

A vertical strip on the left side of the page featuring a detailed, light blue microchip pattern with various circuitry and labels.

ルネサス半導体セミナー

R8Cマイコンコース

テキスト

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りが無いことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。
標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット
高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）
特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

第1章 概要

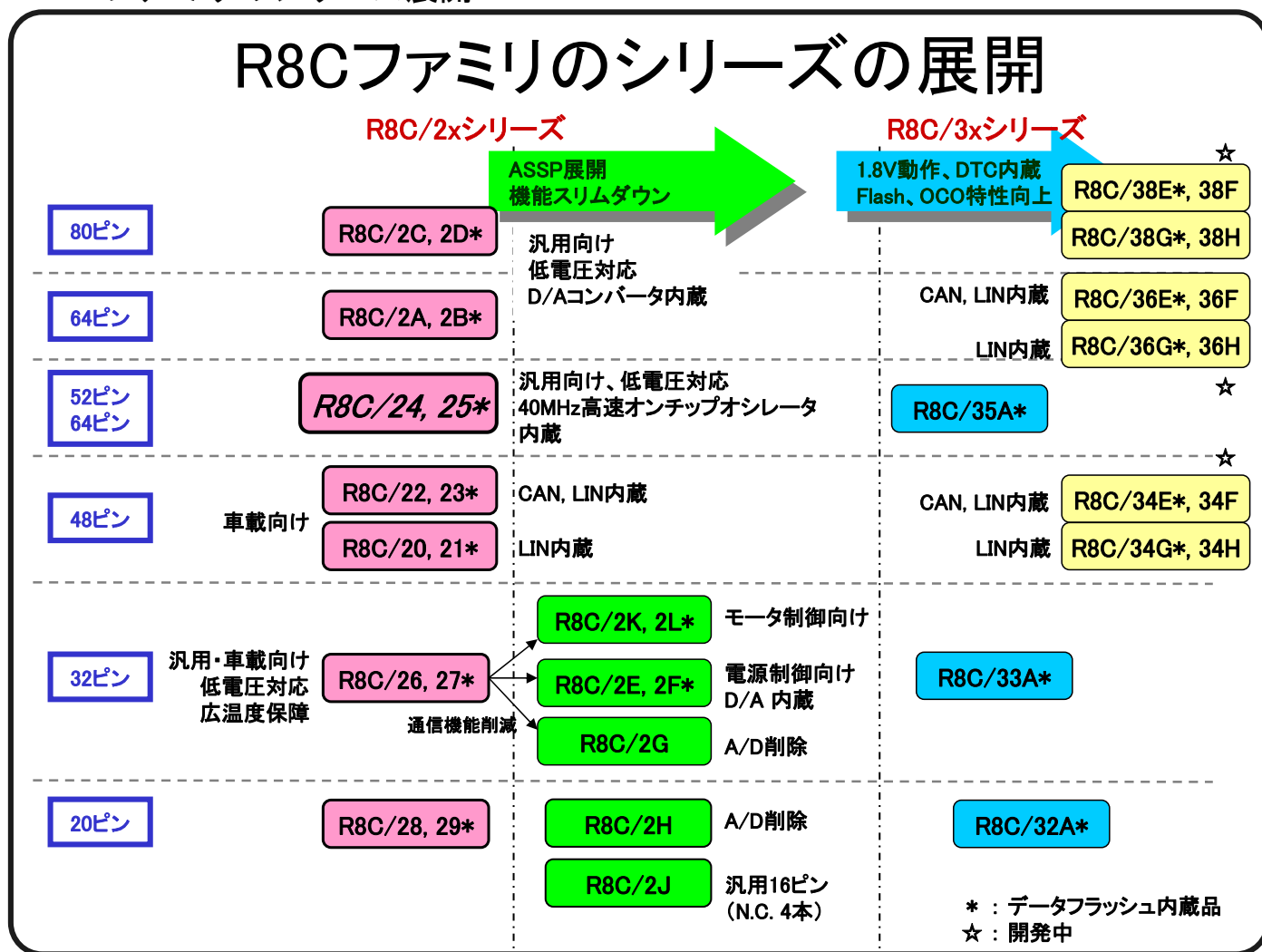
- 1.1 R8Cファミリの特徴
 - 1.2 R8Cファミリのシリーズ展開
 - 1.3 R8C/25グループ概略仕様
 - 1.4 開発ツール
-

1.1 R8Cファミリの特徴

R8Cファミリの特徴

- 高性能・高機能な小型少ピンマイコン
- 専用端子1本でエミュレータ(E8a/E8)と接続可能
- 機能凝縮によりシステムでの外付け部品を削減可能
 - 電圧検出回路内蔵(パワーオンリセット機能あり)
 - 高速オンチップオシレータ内蔵
 - データフラッシュ内蔵
- フェールセーフ機能による高い安全性の確保
 - メインクロック発振停止検出機能
 - 独立クロックで動作可能なウォッチドッグタイマ
- フラッシュメモリ版のみでの製品展開

1.2 R8Cファミリのシリーズ展開



Code : R8Cコース

Date : Rev. 2. 20

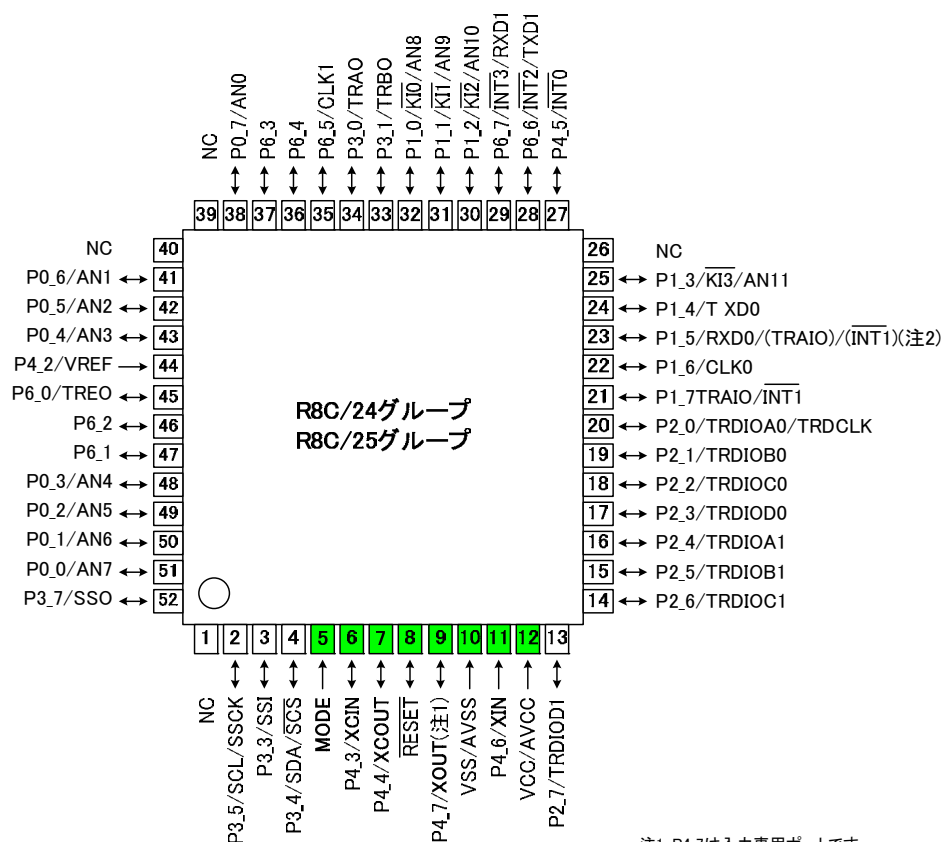
Page: 2 of 6

1.3 R8C/25グループ概略仕様

R8C/25グループ概略仕様

項 目		性 能
電源電圧		3.0~5.5V (f(XIN)=20MHz) 2.7~5.5V (f(XIN)=10MHz) 2.2~5.5V (f(XIN)=5MHz)
パッケージ		52ピンLQFP / 64ピンFLGA
基本バスサイクル		<内部メモリ (RAM, プログラムROM) へのアクセス> バイトアクセス : CPUクロック 1サイクル ワードアクセス : CPUクロック 2サイクル <SFR領域/データフラッシュへのアクセス> バイトアクセス : CPUクロック 2サイクル ワードアクセス : CPUクロック 4サイクル
最短命令実行時間		50ns (f(XIN)=20MHz時、Vcc=3.0~5.5V) 100ns (f(XIN)=10MHz時、Vcc=2.7~5.5V) 200ns (f(XIN)=5MHz時、Vcc=2.2~5.5V)
メモリ容量	ROM	16Kバイト~64Kバイト (2009.3現在)
	RAM	1Kバイト~3Kバイト (2009.3現在)
動作モード		シングルチップモードのみ
クロック発生回路		3回路 ・XINクロック発振回路 (帰還抵抗内蔵) ・XCINクロック発振回路 ・オンチップオシレータ (高速 (周波数調整機能つき)、低速)
内蔵周辺機能		
割り込み		内部 : 11要因、外部 : 5要因、ソフトウェア : 4要因 割り込み優先レベル : 7レベル
タイマ		8bitタイマ (タイマRA, RB) : 各1チャンネル、計2チャンネル 16bitタイマ (タイマRD) : 1チャンネル リアルタイムクロック (タイマRE) : 1チャンネル
シリアルI/O		クロック同期形/クロック非同期形 : 2チャンネル クロック同期形 (I ² Cバス対応) : 1チャンネル
ハードウェアLIN		内蔵 (1チャンネル、タイマRA, UART0使用)
A/D変換器		10ビットA/D 1回路×12チャンネル
ウォッチドッグタイマ		15bit×1チャンネル (リセットスタート機能あり)
発振停止検出機能		メインクロック発振停止検出機能あり
電圧検出回路		内蔵 (検出レベル : 3点)、パワーオンリセット機能あり
入出力ポート		入出力 : 41本 (LED駆動用ポート含む)、入力専用 : 3本
LED駆動用入出力ポート		8本

ピン接続図



注1: P4_7は入力専用ポートです。
注2: プログラムで()の端子に配置できます。

Code : R8Cコース

Date : Rev. 2. 20

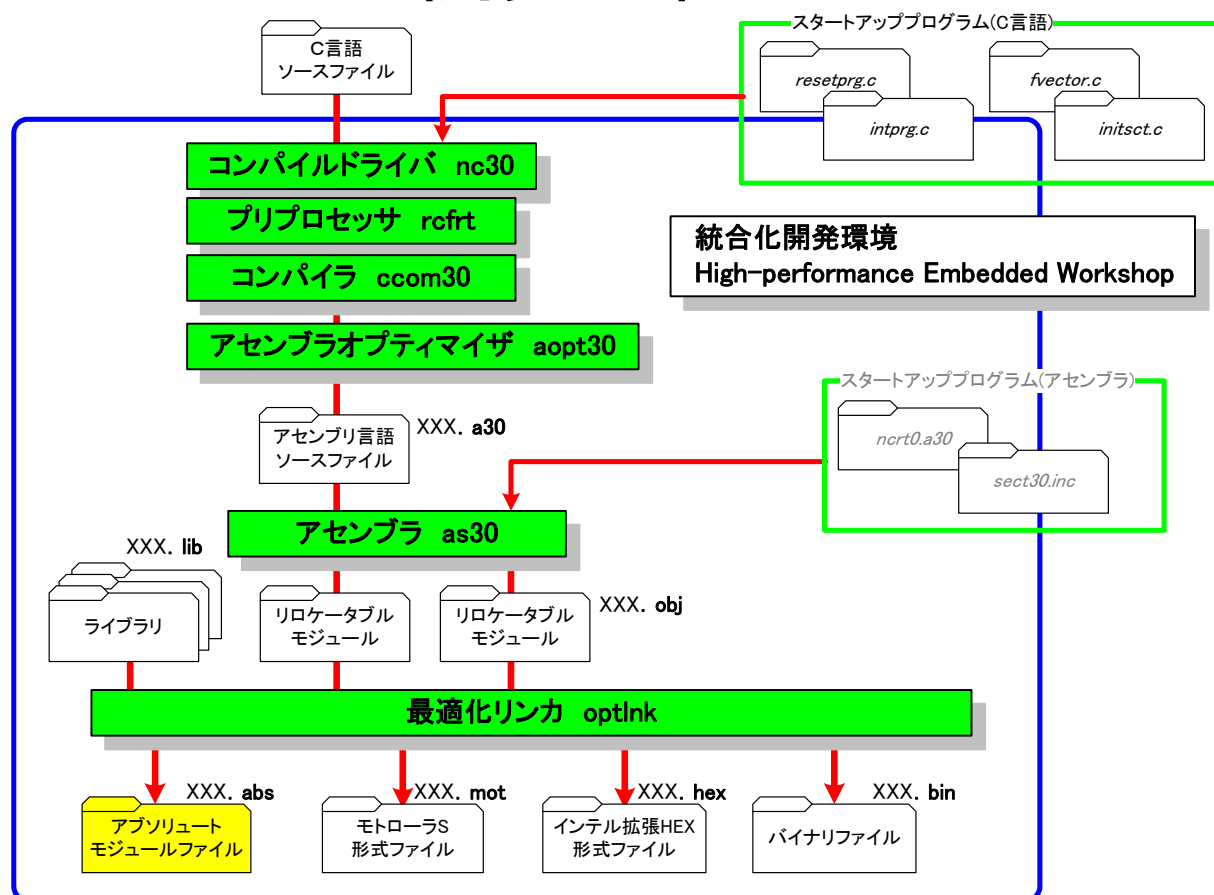
Page: 4 of 6

MODE端子 : マイコン起動時の動作モードを選択する端子

シングルチップモード(通常モード)で起動する場合は“Hレベル”、
標準シリアル入出力モード(ブートモード)で起動する場合は“Lレベル”
を印加する。

1.4 開発支援ツール

開発工程



Code : R8Cコース

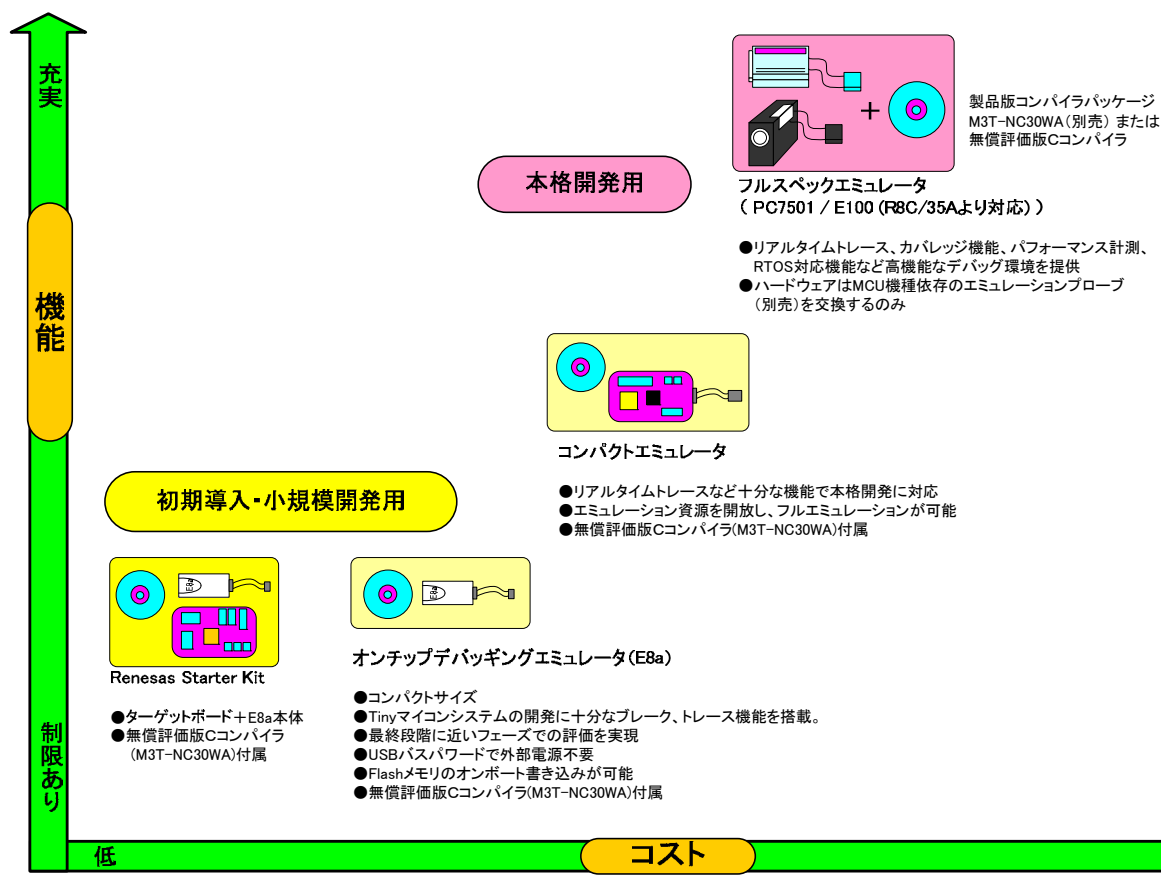
Date : Rev. 2. 20

Page: 5 of 6

M3T-NC30WA (以下NC30とする) 製品構成

- ・ 統合化開発環境 (High-performance Embedded Workshop)
- ・ コンパイルドライバ (nc30)
- ・ プリプロセッサ (rdfrt)
- ・ コンパイラ本体 (ccom30)
- ・ アセンブラオブティマイザ (aopt30)
- ・ 標準ライブラリ構築ツール
- ・ サンプルスタートアッププログラム (C言語／アセンブリ言語)
- ・ スタックサイズ算出ユーティリティ (Call Walker)
- ・ SBDATA宣言 & SPECIALページ関数宣言ユーティリティ (ut130)
- ・ シミュレータデバッグ
- ・ アセンブラドライバ (as30)
- ・ 最適化リンカージェディタ (optlnk)
- ・ ELFフォーマットコンバータ (elfconv) などが含まれる。

R8C/Tinyのデバッグ環境



Code : R8Cコース

Date : Rev. 2. 20

Page: 6 of 6



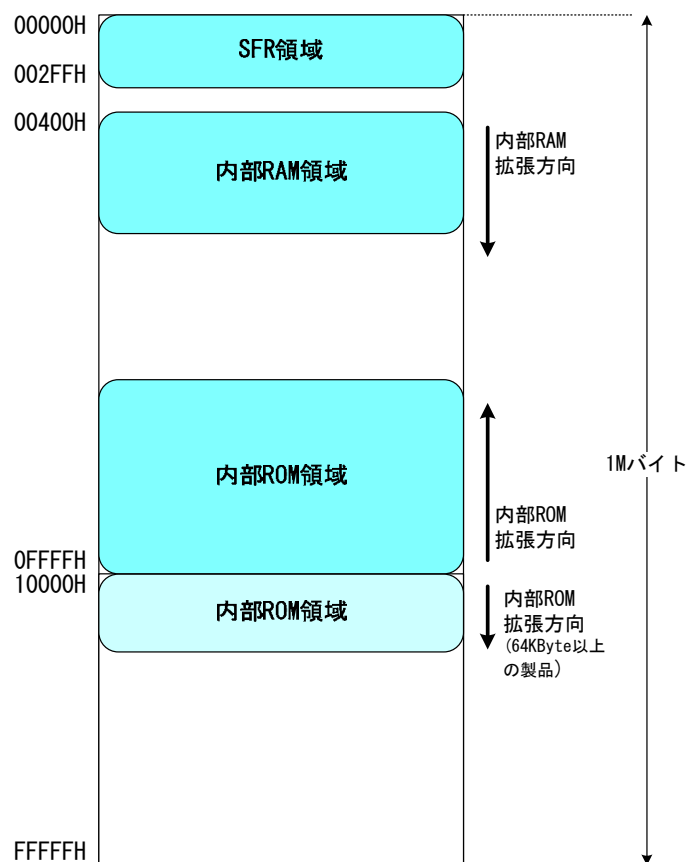
第2章

R8C/2xシリーズのハードウェア

- 2.1 アドレス空間
 - 2.2 リセット動作と電圧検出回路
 - 2.3 発振回路
 - 2.4 パワーコントロール
 - 2.5 プロテクト機能
-

2.1. アドレス空間

内部ROM、RAMの展開



Code : R8Cコース

Date : Rev. 2.20

Page:1 of 22

メモリおよびI/O領域以外の何も配置されていない領域は使用不可。

内蔵メモリが64Kバイト以上の製品は、10000h番地以降にもROMが配置される。

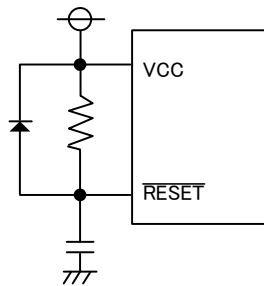
なおR8Cファミリは、シングルチップモード（内部メモリとSFRのみアクセス可能な動作モード）でのみ動作するため、外部メモリなどを拡張領域に割り付けて使用することはできない。

2.2 リセット動作と電圧検出回路

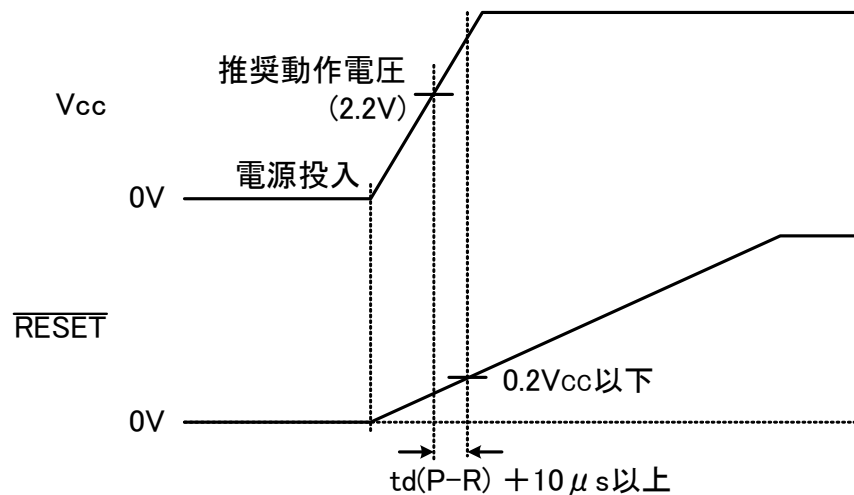
リセット

- **ハードウェアリセット**(外部リセット回路必要)
 - ・RESET端子に印可する電圧レベルによるリセット
- **電圧監視0/1/2リセット**(外部リセット回路不要)
 - ・マイコンに内蔵されている「電圧検出回路」によるリセット
 - ・パワーオンリセット機能あり
- **ソフトウェアリセット**
 - “ソフトウェアリセットビット”によるリセット
- **ウォッチドッグタイマリセット**
 - ウォッチドッグタイマのアンダフローによるリセット

ハードウェアリセット



【リセット回路例】



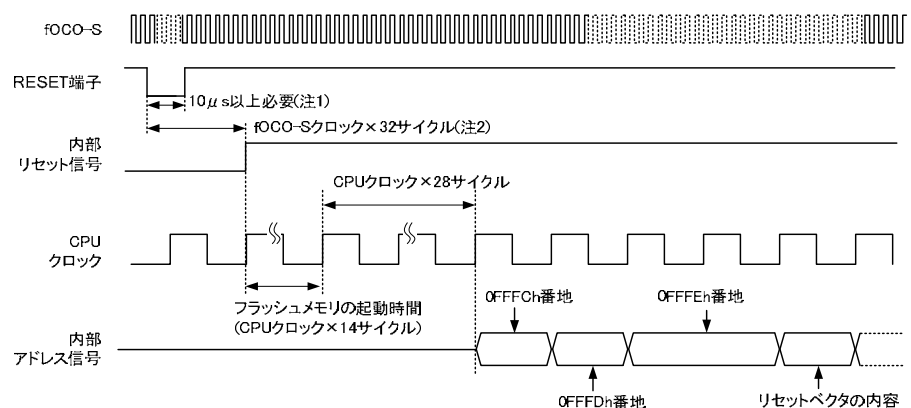
$t_d(P-R)$: 電源投入時の内部電源安定時間 = 2ms(Max)

Code : R8Cコース

Date : Rev. 2.20

Page: 3 of 22

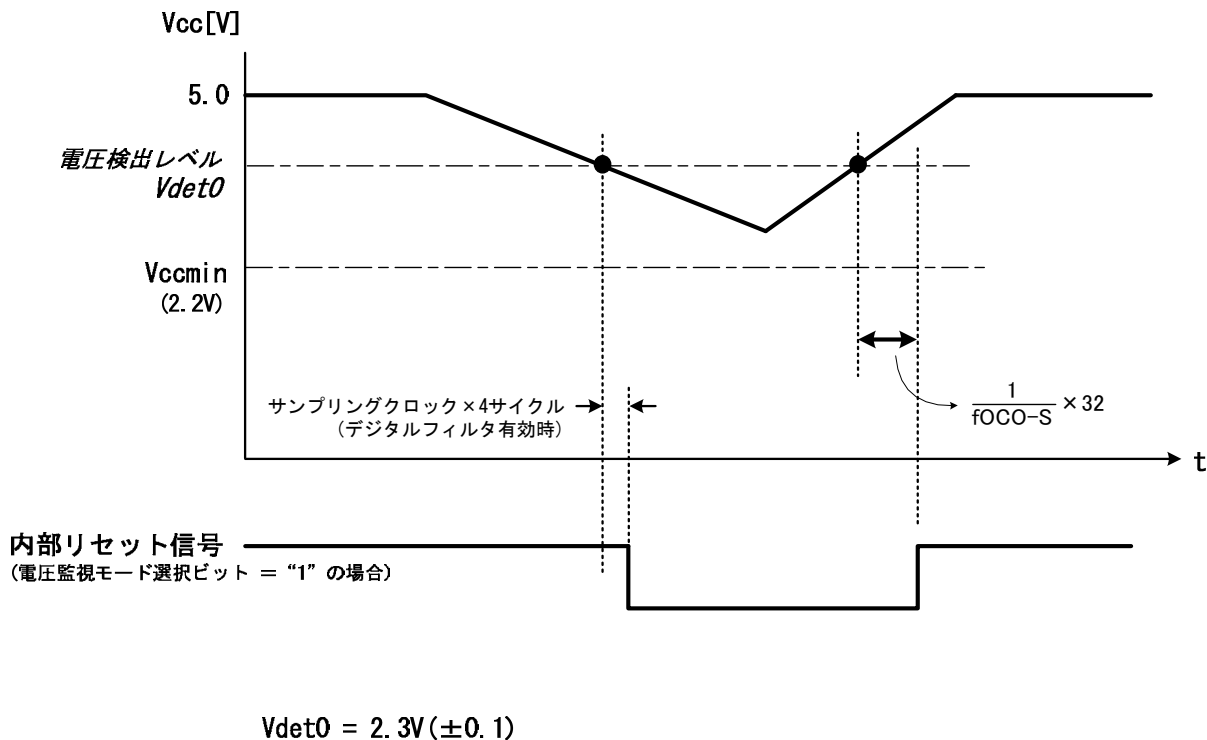
【リセットシーケンス】



注1) ハードウェアリセットの場合

注2) RESET端子への“L”入力幅がfOCO-Sクロック×32サイクル以上の場合には、RESET端子が“H”になると同時に内部リセット信号も“H”になります。

電圧監視0リセット



Code : R8Cコース

Date : Rev. 2.20

Page: 4 of 22

R8C/25の電圧検出レベルは、以下に示す3ポイントがある

- $V_{det0} = 2.3V(\pm 0.1)$
- $V_{det1} = 2.85V(\pm 0.15)$
- $V_{det2} = 3.6V(\pm 0.3)$

電圧検出レジスタ2 (注1)

VCA2 0032H番地

リセット解除後の値: (注5)

b7								b0
			0	0	0	0		

- ① OFSレジスタのLVD00Nビット = "1"で、かつハードウェアリセット時 : 00000000B
- ② パワーオンリセット、電圧監視0リセット、およびOFSレジスタのLVD00Nビット = "0"の場合のハードウェアリセット時 : 00100000B

ビット名/ビットシンボル	機能	R	W
内部電源低消費電力許可ビット/VCA20 (注6)	0 : 低消費電力禁止 1 : 低消費電力許可	○	○
予約ビット	"0"にしてください	○	○
電圧検出0許可ビット (注2)/VCA25	0 : 電圧検出0回路無効 1 : 電圧検出0回路有効	○	○
電圧検出1許可ビット (注3)/VCA26	0 : 電圧検出1回路無効 1 : 電圧検出1回路有効	○	○
電圧検出2許可ビット (注4)/VCA27	0 : 電圧検出2回路無効 1 : 電圧検出2回路有効	○	○

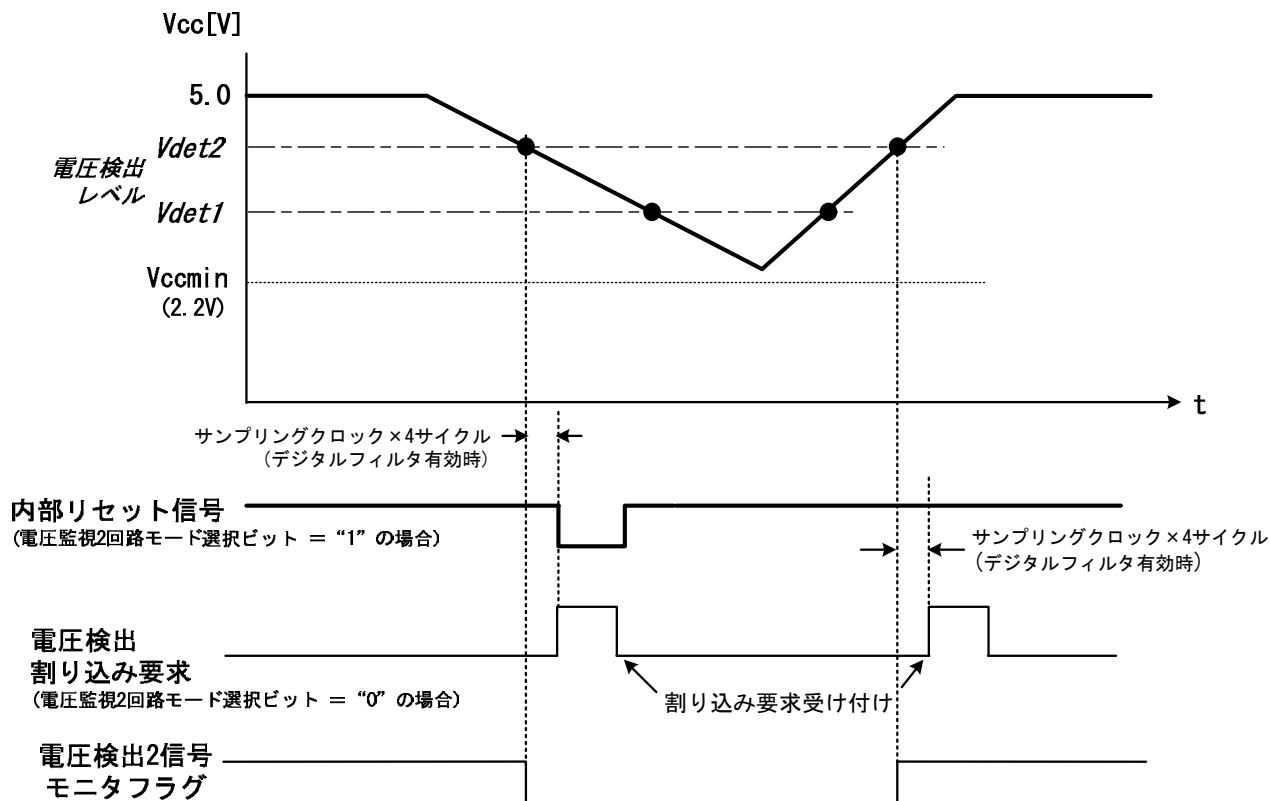
注1. VCA2レジスタはPRCRレジスタのPRC3ビットを"1" (書き込み許可) にした後で書き換えてください。

注2. 電圧監視0リセットを使用する場合、VCA25ビットを"1"にしてください。VCA25ビットを"0"から"1"にした後、 $t_d(E-A)$ 経過してから検出回路が動作します。注3. 電圧監視1割り込み/リセットを使用する場合、またはVW1CレジスタのVW1C3ビットを使用する場合、VCA26ビットを"1"にしてください。VCA26ビットを"0"から"1"にした後、 $t_d(E-A)$ 経過してから検出回路が動作します。注4. 電圧監視2割り込み/リセットを使用する場合、またはVCA1レジスタのVCA13ビットを使用する場合、VCA27ビットを"1"にしてください。VCA27ビットを"0"から"1"にした後、 $t_d(E-A)$ 経過してから検出回路が動作します。

注5. ソフトウェアリセット、ウォッチドッグタイマリセット、電圧監視1リセット、電圧監視2リセット時は変化しません。

注6. FRA0レジスタのFRA00ビットが"0" (高速オンチップオシレータ停止) のときのみ有効です。ウェイトモードへの移行時のみに使用してください。

電圧監視2(または1)リセット



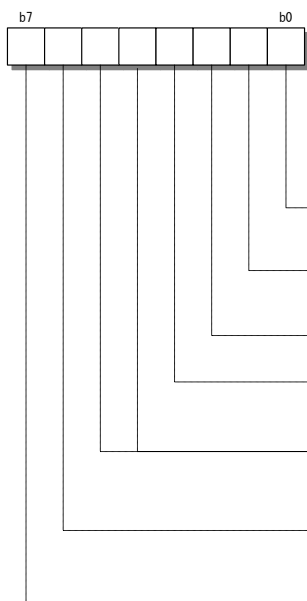
Code : R8Cコース

Date : Rev. 2.20

Page: 5 of 22

電圧監視2回路制御レジスタ

VW2C 0037H番地 リセット時: 00000000B



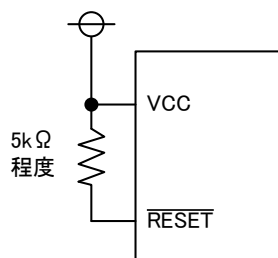
ビット名/ビットシンボル	機能	R	W
電圧監視2監視割り込み/ リセット許可ビット /VW2C0	0 : 禁止 1 : 許可	○	○
電圧監視2デジタルフィルタ 無効モード選択ビット /VW2C1	0 : デジタルフィルタ有効モード (デジタルフィルタ回路有効) 1 : デジタルフィルタ無効モード (デジタルフィルタ回路無効)	○	○
電圧変化検出フラグ /VW2C2	0 : 未検出 1 : Vdet2通過検出	○	○
WDT検出フラグ /VW2C3	0 : 未検出 1 : 検出	○	○
サンプリングクロック 選択ビット /VW2CF0, VW2CF1	00 : fOC0-sの1分周(分周なし) 01 : fOC0-sの2分周 10 : fOC0-sの4分周 11 : fOC0-sの8分周	○	○
電圧監視2回路モード選択 ビット /VW2C6	0 : 電圧監視2割り込みモード 1 : 電圧監視2リセットモード	○	○
電圧監視2割り込み/ リセット発生条件選択 ビット /VW2C7	0 : Vdet2以上になるとき 1 : Vdet2以下になるとき	○	○

・Vdet1=2.85V(±0.15)

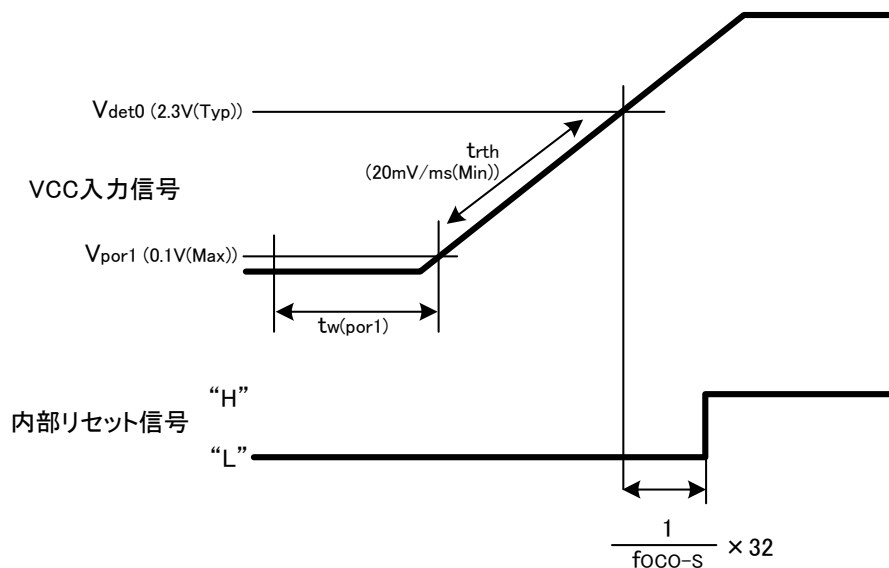
・Vdet2=3.6V(±0.3)

※ 電圧低下時のリセット動作だけでなく、Vccに入力される電圧が下降してVdeti(i=1,2)越えたとき、または上昇してVdeti(i=1,2)を越えたときに“電圧検出1または電圧検出2割り込み”を発生させることができる。

パワーオンリセット機能



注) RESET端子は常に
0.8V_{CC}以上を印加する



V_{por1} : パワーオンリセットが有効になる電圧
tw(por1) : V_{por1}の保持時間
trth(V_{por1}-V_{det1}) : 外部電源VCCの立ち上がり傾き
foco-s : 低速オンチップオシレータ発振周波数

Code : R8Cコース

Date : Rev. 2.20

Page:6 of 22

※ パワーオンリセット機能使用時は、必ず電圧監視0リセットを有効にした状態でシステムを動作させること。

プロセッサモードレジスタ0,1

プロセッサモードレジスタ0 (注1)

b7	PM07	PM06	PM05	PM04	PM03	PM02	PM01	PM00	b0
						0	0	0	
PM0 0004H番地 リセット時：0000 0000B									
ビット名		機能		R	W				
予約ビット		“0” にしてください。		○	○				
ソフトウェアリセットビット /PM03		このビットに“1”を書き込むとマイクロコンピュータはリセットされる。 読み出し時の値は“0”。		○	○				
何も配置されていない。書き込む場合は“0”を書いてください。 読み込んだ場合、その値は“0”。				—	—				

注1. プロセッサモードレジスタ0は、プロテクトレジスタのビット1を“1”（書き込み許可）にした後書き換える。

プロセッサモードレジスタ1 (注1)

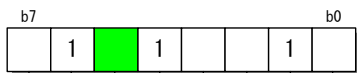
b7	PM17	PM16	PM15	PM14	PM13	PM12	PM11	PM10	b0
	0						0	0	
PM1 0005H番地 リセット時：0000 0000B									
ビット名		機能						R	W
予約ビット		“0”を設定する						○	○
WDT割り込み／リセット切り替えビット		0：ウォッチドッグタイマ割り込み 1：ウォッチドッグタイマリセット(注2)						○	○
何も配置されていない。書き込む場合“0”を書いてください。 読み込んだ場合、その値は“0”。								—	—
予約ビット		“0”を設定する						○	○

注1. プロセッサモードレジスタ1は、プロテクトレジスタのビット1を“1”（書き込み許可）にした後書き換える。

注2. WDT割り込み/リセット切り替えビットはプログラムで“1”のみ書き込める。(“0”を書いても変化しない)

オプション機能選択レジスタ

オプション機能選択レジスタ（注1）



OFS 0FFFFH番地 リセット時：11111111B（注3）

ビット名/ビットシンボル	機能	R	W
ウォッチドッグタイマ起動選択ビット/WDTON	0：リセット後、ウォッチドッグタイマは自動的に起動 1：リセット後、ウォッチドッグタイマは停止状態	○	○
予約ビット	“1”にしてください	○	○
ROMコードプロテクト解除ビット/ROMCR	0：ROMコードプロテクト解除 1：ROMCP1有効	○	○
ROMコードプロテクトビット/ROMCP1	0：ROMコードプロテクト有効 1：ROMコードプロテクト解除	○	○
予約ビット	“1”にしてください	○	○
電圧検出0回路起動ビット（注2）/LVD00N	0：リセット後、電圧監視0リセット有効 1：リセット後、電圧監視0リセット無効	○	○
予約ビット	“1”にしてください	○	○
リセット後カウントソース保護モード選択ビット/CSPRO1NI	0：リセット後、カウントソース保護モード有効 1：リセット後、カウントソース保護モード無効	○	○

注1. OFSレジスタはフラッシュメモリ上にあります。プログラムと一緒に書き込んでください。

注2. パワーオンリセットを使用する場合、LVD00Nビットを“0”（リセット後、電圧監視0リセット有効）にしてください。

注3. OFSレジスタを含むブロックを消去すると、OFSレジスタは“FFh”になります。

Code： R8Cコース

Date： Rev. 2.20

Page: 8 of 22

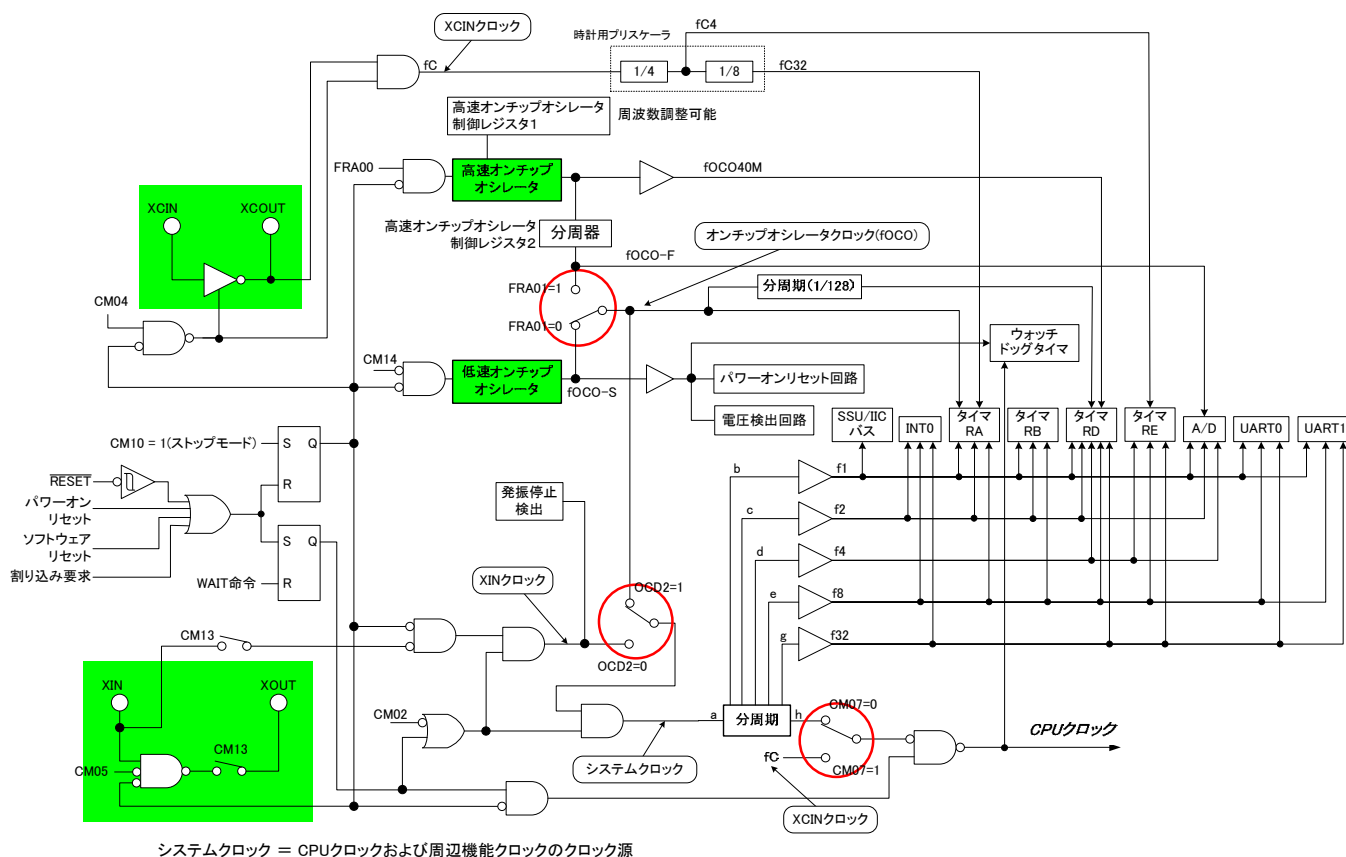
※ オプション機能選択レジスタは内部ROM内に配置されているため、プログラム実行中に値を変更することはできないので注意が必要。

2.3 発振回路

クロック発生回路の仕様

項 目	XINクロック発振回路	XCINクロック発振回路	オンチップオシレータ	
			高速オンチップオシレータ	低速オンチップオシレータ
クロックの用途	・CPUのクロック源 ・周辺機能のクロック源	・CPUのクロック源 ・タイマRAおよび タイマREのクロック源	・CPUのクロック源 ・周辺機能のクロック源 ・XINクロック発振停止 時のCPUおよび周辺 機能のクロック源	・CPUのクロック源 ・周辺機能のクロック源 ・XINクロック発振停止 時のCPUおよび周辺 機能のクロック源
クロック周波数	0 ~ 20MHz	32.768kHz	約40MHz (調整可)	約125kHz
接続できる端子	・セラミック共振子 ・水晶発振子	水晶発振子	—	—
発振子の接続端子	XIN, XOUT	XCIN, XCOUT	—	—
発振の停止/再開機能	あり	あり	あり	あり
リセット解除後の状態	停止	停止	停止	発振
その他	・外部で生成された クロックを入力可能	・外部で生成された クロックを入力可能 ・帰還抵抗内蔵(接続/ 非接続を選択可)	—	—

クロック発生回路



Code : R8Cヨース

Date : Rev. 2. 20

Page:10 of 22

●パワーコントロール

パワーコントロールとはCPUの消費電流を小さくすることで、R8C/Tinyシリーズでは以下の3つのモードがある。

(1) 標準動作モード

- ・高速クロックモード：XINクロックの1～16分周がCPUクロックとなるモード。
- ・低速クロックモード：XCINクロックがCPUクロックとなるモード。
- ・高速オンチップオシレータモード：
高速オンチップオシレータ制御レジスタ2で分周された高速オンチップオシレータクロックの1～16分周がCPUクロックとなるモード。
- ・低速オンチップオシレータモード：
低速オンチップオシレータクロックの1～16分周がCPUクロックとなるモード。

(2) ウェイトモード

CPUクロックを停止させるモード。（外部発振子およびオンチップオシレータは停止しない）

(3) ストップモード

全ての発振が停止するモード。(消費電流がもっとも少なくなるモード)

システムクロック制御レジスタ0

システムクロック制御レジスタ0 (注1)



CM0 0006H番地 リセット時：01101000B

ビット名/ビットシンボル	機能	R	W
予約ビット	“0” にしてください。	○	○
WAIT時周辺機能クロック停止ビット/CM02	0：ウェイトモード時、周辺機能クロック停止しない 1：ウェイトモード時、周辺機能クロック停止する(注8)	○	○
XCIN-XCOUT駆動能力選択ビット(注9)/CM03	0：LOW 1：HIGH	○	○
ポート、XCIN-XCOUT切り替えビット(注6)/CM04	0：入出力ポート機能P4_3, P4_4 1：XCIN、XCOUT端子(注7)	○	○
XINクロック(XIN-XOUT)停止ビット(注2, 4)/CM05	0：発振 1：停止(注3)	○	○
システムクロック分周比選択ビット0(注5)/CM06	0：CM16, CM17有効 1：8分周モード	○	○
CPUクロック選択ビット(注8)/CM07	0：システムクロック 1：XCINクロック	○	○

- 注1. このレジスタを書き替える場合、プロテクトレジスタ(000AH番地)のビット0を“1”にする。
 注2. ストップモードへの移行時およびリセット時、“1”になる。
 注3. このビットは低消費電力モードにするときに、メインクロックを停止させるためのビット。メインクロックを停止させる場合、サブクロックが安定して発振している状態で、システムクロック選択ビット(CM07)を“1”にしてから、このビットを“1”にする。
 注4. 外部クロック入力時には、クロック発振バッファだけ停止し、クロック入力は受け付けられるモードとなる。
 注5. このビットが“1”の場合、XOUTは“H”レベルになる。また、内蔵している帰還抵抗は接続したままなので、XINは帰還抵抗を介して、XOUT (“H”レベル)にプルアップされた状態となる。
 注6. このビットを“0”から“1”にする場合、ポートXC切り替えビット(CM04)を“1”にし、サブクロックの発振が安定した後に行うこと。同時に書き込まないこと。また、このビットを“1”から“0”にする場合は、メインクロック停止ビット(CM05)を“0”にし、メインクロックの発振が安定した後に行うこと。
 注7. 高速モード、中速モードからストップモードへの移行時およびリセット時、このビットは“1”になる。低速モード、低消費電力モードでは保持される。
 注8. fC32は含まない。低速モード、低消費電力モード時は、“1”にしないこと。
 注9. XCIN/XCOUTを使用する場合、ポートP86、P87は入力ポートで、プルアップなしを設定すること。

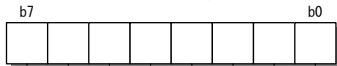
Code : R8Cコース

Date : Rev. 2.20

Page: 11 of 22

システムクロック制御レジスタ1

システムクロック制御レジスタ1 (注1)



CM1 0007H番地 リセット時：00100000B

ビット名/ビットシンボル	機能	R	W
全クロック停止制御ビット /CM10 (注4、7、8)	0: クロック発振 1: 全クロック停止 (ストップモード)	○	○
XIN-XOUT内蔵帰還抵抗選択ビット /CM11	0: 内蔵帰還抵抗有効 1: 内蔵帰還抵抗無効	○	○
XCIN-XCOUT内蔵帰還抵抗選択ビット /CM12	0: 内蔵帰還抵抗有効 1: 内蔵帰還抵抗無効	○	○
ポートXIN-XOUT切り替えビット (注7、9) /CM13	0: 入力ポートP4_6、P4_7 1: XIN-XOUT端子	○	○
低速オンチップオシレータ発振停止ビット (注5、6、8) /CM14	0: 低速オンチップオシレータ発振 1: 低速オンチップオシレータ停止	○	○
XIN-XOUT駆動能力選択ビット (注2) /CM15	0: LOW 1: HIGH	○	○
システムクロック分周比選択ビット1 (注3) /CM16、CM17	00: 分周なしモード 01: 2分周モード 10: 4分周モード 11: 16分周モード	○	○

- 注1. CM1レジスタはPRCRレジスタのPRC0ビットを“1”（書き込み許可）にした後で書き換えてください。
- 注2. ストップモードへの移行時、CM15ビットは“1”（駆動能力HIGH）になります。
- 注3. CM06ビットが“0”（CM16、CM17ビット有効）の場合、CM16～CM17ビットは有効となります。
- 注4. CM10ビットが“1”（ストップモード）の場合、内蔵している帰還抵抗は無効となります。
- 注5. CM14ビットはOCD2ビットが“0”（XINクロック選択）のとき、“1”（低速オンチップオシレータ停止）にできます。OCD2ビットを“1”（オンチップオシレータクロック選択）にすると、CM14ビットは“0”（低速オンチップオシレータ発振）になります。“1”を書いても変化しません。
- 注6. 電圧監視1割り込み、電圧監視2割り込みを使用する場合（デジタルフィルタを使用する場合）、CM14ビットを“0”（低速オンチップオシレータ発振）にしてください。
- 注7. CM10ビットが“1”（ストップモード）の場合、CM13ビットが“1”（XIN-XOUT端子）のとき、XOUT（P4_7）端子は“H”になります。CM13ビットが“0”（入力ポートP4_6、P4_7）のとき、P4_7（XOUT）は入力状態になります。
- 注8. カウントソース保護モード有効時は、CM10、CM14ビットへ書いても値は変化しません。
- 注9. CM13ビットはプログラムで一度“1”にすると、“0”にはできません。

Code : R8Cコース

Date : Rev. 2.20

Page: 12 of 22

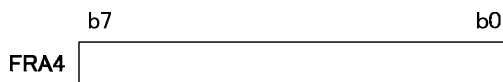
2-13

周波数調整用レジスタ

高速オンチップオシレータ制御レジスタ4

0029h番地

リセット後の値: 出荷時の値(周波数が40MHzとなる値)



----- VCC2.7~5.5V時の周波数補正用データが格納されています。

※ この値をFRA1レジスタに転送することにより、電圧条件に応じた最適補正ができます。

高速オンチップオシレータ制御レジスタ6

002Bh番地

リセット後の値: 出荷時の値(周波数が40MHzとなる値)



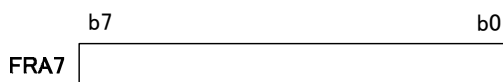
----- VCC2.2~5.5V時の周波数補正用データが格納されています。

※ この値をFRA1レジスタに転送することにより、電圧条件に応じた最適補正ができます。

高速オンチップオシレータ制御レジスタ7

002Ch番地

リセット後の値: 出荷時の値



----- 高速オンチップオシレータの周波数を36.864MHzに調整するデータが格納されています

※ シリアルインタフェースをクロック非同期形で使用する場合、システムクロックに高速オンチップオシレータを選択することで、ビットレートの計算誤差を0%にすることができます。

Code : R8Cコース

Date : Rev. 2.20

Page: 14 of 22

FRA4レジスタ、FRA6レジスタおよびFRA7レジスタは読み出し専用レジスタです。

【 クロック非同期形でのビットレート設定例(内部クロック選択時) 】

ビットレート (bps)	カウントソース	システムクロック=18.432MHz(注)		
		設定値(n)	実時間(bps)	誤差(%)
1200	f8	119(77h)	1200	0
2400	f8	59(3Bh)	2400	0
4800	f8	29(1Dh)	4800	0
9600	f1	119(77h)	9600	0
14400	f1	79(4Fh)	14400	0
19200	f1	59(3Bh)	19200	0
28800	f1	39(27h)	28800	0
38400	f1	29(1Dh)	31250.00	0
57600	f1	19(13h)	37878.79	0
115200	f1	9(09h)	52083.33	0

注: FRA2で2分周を選択し、システムクロックに高速オンチップオシレータを選択した場合。

XINクロックへの切り替え手順

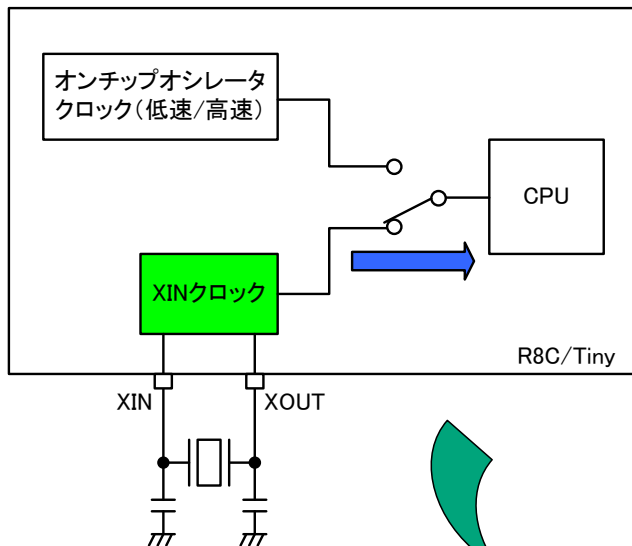
リセット解除後、オンチップオシレータモード → 高速モードへ切り替える場合

- (1) ポートXIN-XOUT切り替えビット(システムクロック制御レジスタ1のビット3)を“1”(XIN-XOUT端子)にした後、XINクロック停止ビット(システムクロック制御レジスタ0のビット5)を“0”(XINクロック発振)にする。
- (2) システムクロック分周比選択ビット1(システムクロック制御レジスタ1のビット6, 7)を“00”(分周なしモード)にした後、システムクロック分周比選択ビット0(システムクロック制御レジスタ0のビット6)を“0”にする。
- (3) XINクロックの発振が安定する時間ウェイトした後、システムクロック選択ビット(発振停止検出レジスタのビット2)を“0”(XINクロック選択)にする。

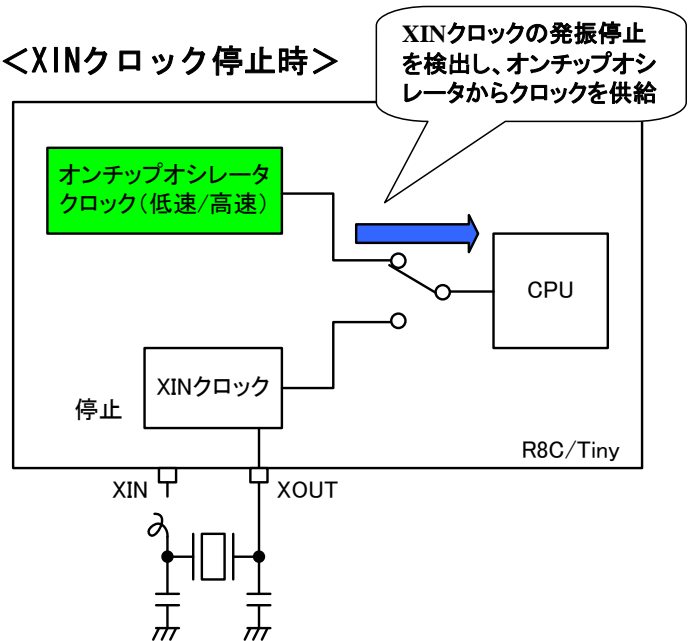
※ CPUクロック選択ビット(システムクロック制御レジスタ0のビット7)は“0”(システムクロック)になっている必要があります。

発振停止検出機能

<XINクロック発振時>



<XINクロック停止時>



Code : R8Cコース

Date : Rev. 2.20

Page:16 of 22

発振停止検出機能

XINクロックを常に監視し、XINクロックが停止すると オンチップオシレータ(低速)が発振を開始する。

CPUはこの信号を動作クロックとし、発振停止検出割り込みを発生させることが可能。

発振停止検出レジスタ

発振停止検出レジスタ（注1）



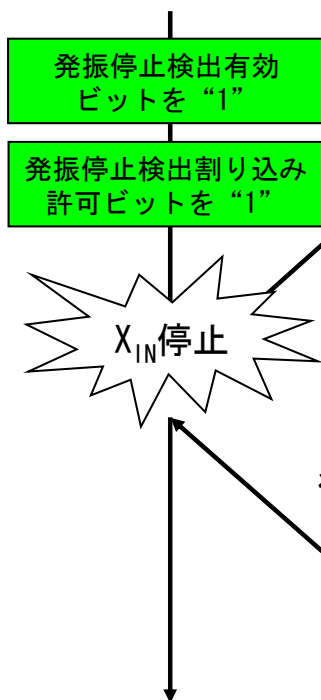
リセット時：0000 0100B

ビット名/ビットシンボル	機能	R	W
発振停止検出有効ビット /OCD0	0：発振停止検出機能無効（注2） 1：発振停止検出機能有効	○	○
発振停止検出割り込み許可ビット /OCD1	0：禁止（注2） 1：許可	○	○
システムクロック選択ビット（注4）/OCD2	0：XINクロック選択（注7） 1：オンチップオシレータクロック選択（注3）	○	○
クロックモニタビット（注5、注6）/OCD3	0：XINクロック発振 1：XINクロック停止	○	—
予約ビット	“0”にしてください	○	○

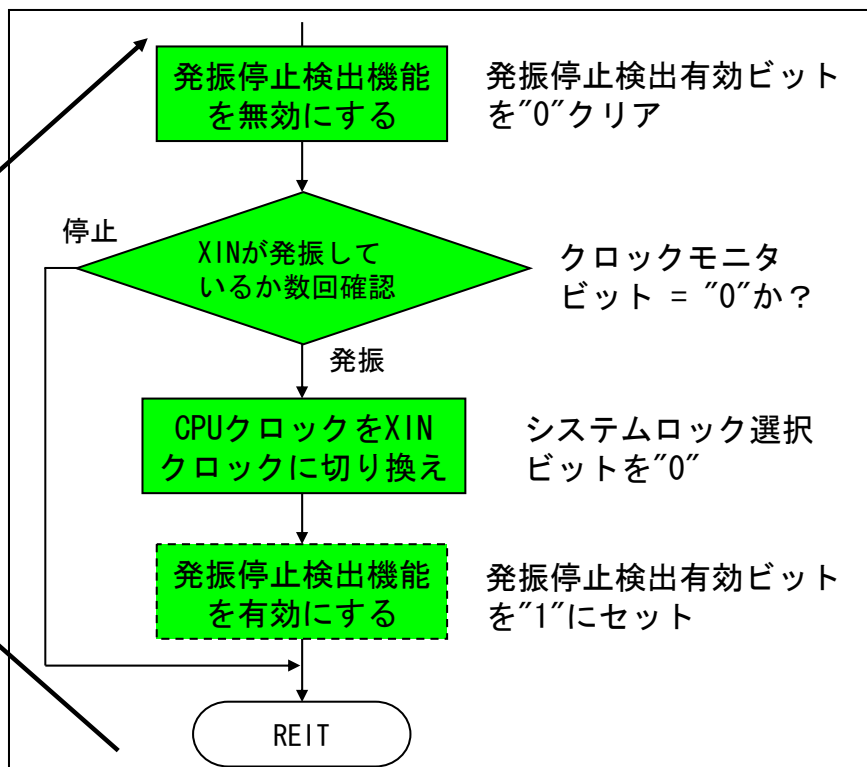
- 注1. このレジスタは、プロテクトレジスタのビット0を“1”（書き込み許可）にした後、書き換える。
- 注2. ストップモード、高速オンチップオシレータモード、低速オンチップオシレータモード（XINクロック停止）に移行する前にOCD1～OCD0ビットを“00b”に設定してください。
- 注3. OCD2ビットを“1”（オンチップオシレータ選択）にすると、CM14ビットは“0”（低速オンチップオシレータ発振）になります。
- 注4. OCD2ビットは、OCD1～OCD0ビットが“11b”のときにXINクロック発振停止を検出すると、自動的に“1”（オンチップオシレータクロック選択）に切り替わります。また、OCD3ビットが“1”（XINクロック停止）のとき、OCD2ビットに“0”（XINクロック選択）を書いても変化しません。
- 注5. OCD3ビットはOCD0ビットが“1”（発振停止検出機能有効）のとき有効です。
- 注6. OCD1～OCD0ビットが“00b”のときOCD3ビットは“0”（XINクロック発振）になり、変化しません。

発振停止検出から復帰まで

<通常動作>



<発振停止検出割り込み>



Code : R8Cコース

Date : Rev. 2.20

Page:18 of 22

発振停止検出割り込み、電圧監視1割り込み、電圧監視2割り込み、およびウォッチドッグタイマ割り込みは割り込みベクタを共有しているため、発生した割り込み ルーチン内で割り込み要因の判別が必要となる。

以下の表に示すように、各レジスタのフラグを参照することで割り込み要因の判別ができる。

発生した割り込み要因	割り込み要因の判定ビット
発振停止検出 ((a)または(b)のとき)	(a) 発振停止検出レジスタのクロックモニタビット(OCD3) = 1 (XINクロック停止)のとき
	(b) 発振停止検出レジスタの発振停止検出有効ビット(OCD0~1) = 11 かつシステムクロック選択ビット(OCD2) = 1 (オンチップオシレータクロック選択)のとき
電圧監視1	電圧監視1回路制御レジスタの電圧変化検出フラグ(VW1C2) = 1 (Vdet1通過検出)のとき
電圧監視2	電圧監視2回路制御レジスタの電圧変化検出フラグ(VW2C2) = 1 (Vdet2通過検出)のとき
ウォッチドッグタイマ	電圧監視2回路制御レジスタのWDT検出フラグ(VW2C3) = 1 (検出)のとき

2.4 パワーコントロール

ストップモードとウェイトモード

■ ストップモード

全ての発振(XINクロック、XCINクロック、オンチップオシレータクロック)が停止するモード。

- ・全クロック停止制御ビット(0007H番地のビット0)を“1”にすることでストップモードとなる。
なおストップモード時にVCCが2V以上であれば、内部RAMの内容は保持される。
- ・ストップモード移行時に、システムクロック分周比選択ビット0が“1”(8分周モード)になる。
- ・ストップモード時の端子の状態は、ストップモードに入る直前の状態を保持する。
- ・ストップモード解除は、ハードウェアリセットまたは以下の周辺機能割り込みにより行なう。
 - ①キー入力割り込み
 - ②INT0～INT3割り込み
 - ③タイマRA割り込み(イベントカウンタモードで外部パルスをカウント時)
 - ④シリアルI/O割り込み(外部クロック選択時)
 - ⑤電圧監視1、電圧監視2割り込み

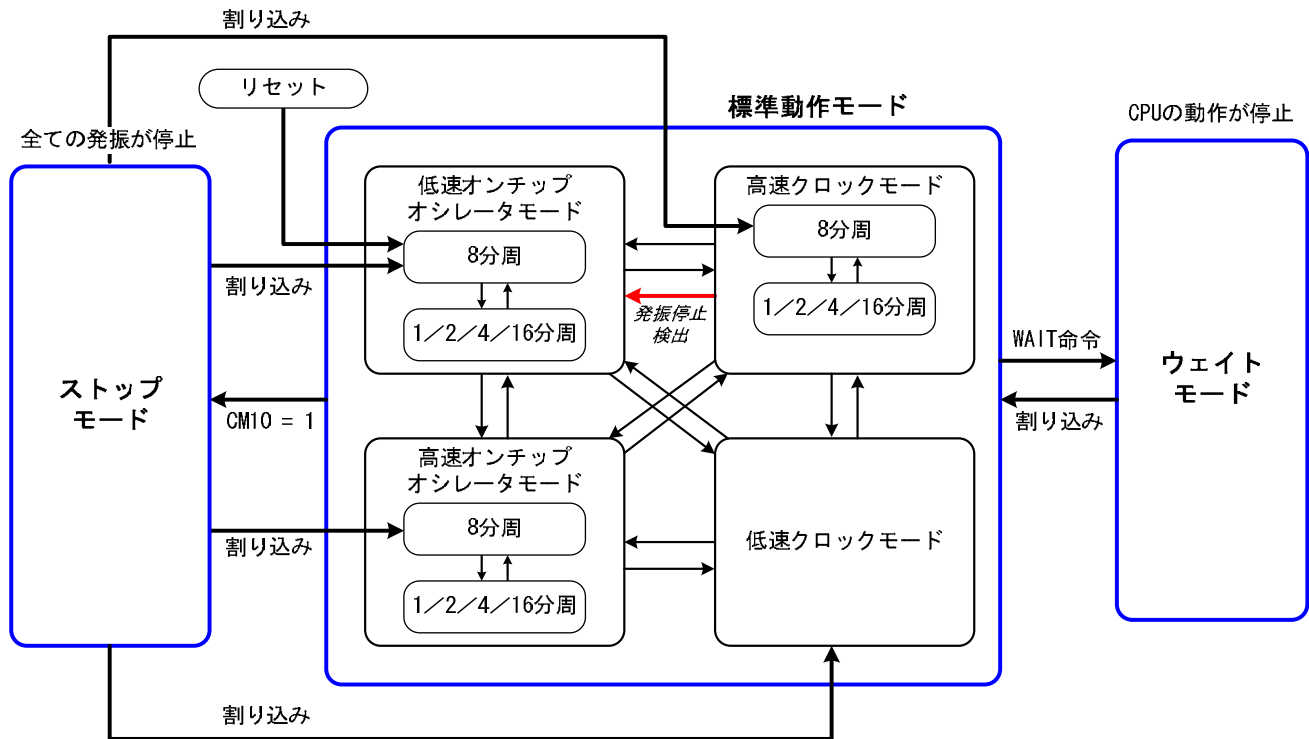
■ ウェイトモード

CPUクロックとカウントソース保護モード無効時のウォッチドッグタイマが停止するモード。

XINクロック、XCINクロック、およびオンチップオシレータクロック は停止しない。

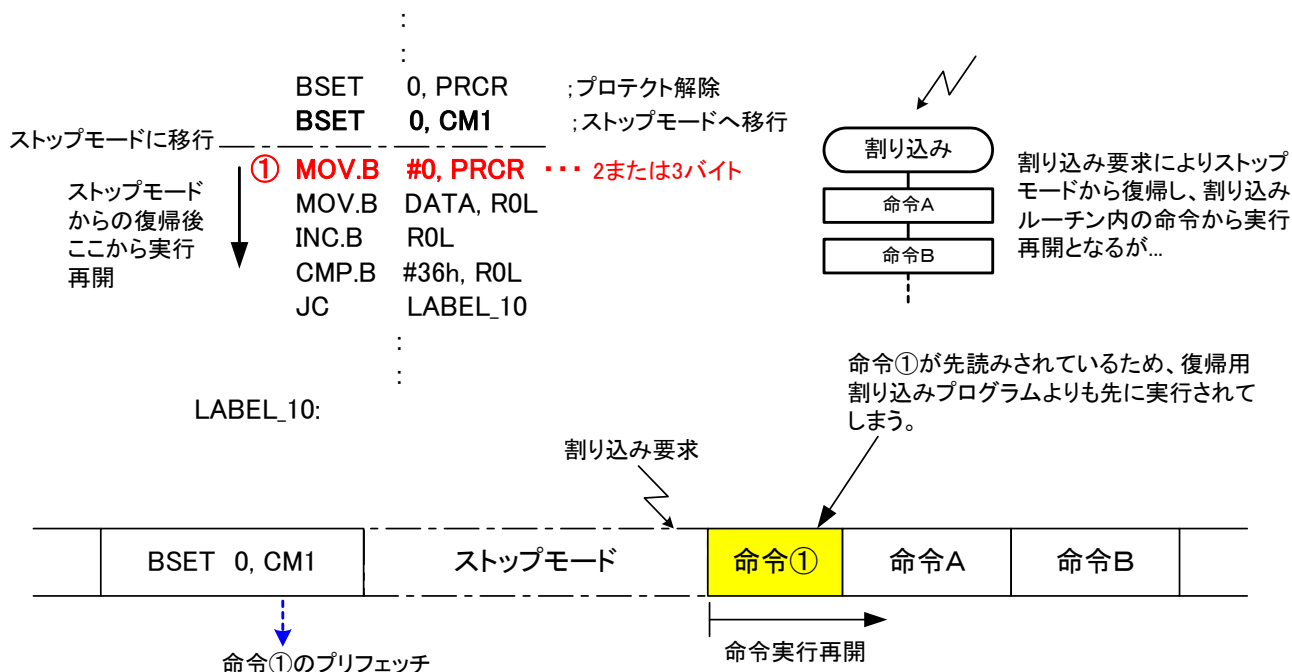
- ・WAIT命令を実行することでウェイトモードとなる。WAIT時周辺機能クロック停止ビットを“1”にしてWAIT命令を実行すると、周辺機能クロックが停止するため、消費電力を低減することができる。
- ・ウェイトモード時の端子の状態は、ウェイトモードに入る直前の状態を保持する。
- ・ウェイトモードの解除は、ハードウェアリセットまたは周辺機能割り込みにより行う。

ストップモード/ウェイトモードへの移行



※ CM10 : システムクロック制御レジスタ1のbit0(全クロック停止制御ビット)

ストップモード移行時の注意点



※ 命令キューの状態、および「BSET命令」の後続命令の種類によって「BSET命令」実行後に命令①が実行され、その後ストップモードに移行する場合もある。

Code : R8Cコース

Date : Rev. 2.20

Page: 21 of 22

ストップモード移行時の記述例

ストップモード移行時は、以下のようにBSET命令の後に4命令以上NOP命令を挿入すること。

```

BCLR 1, FMRO ; CPU書き換えモード無効
FSET 1 ; 割り込み許可
:
BSET 0, PRCR ; プロテクト解除
BSET 0, CM1 ; ストップモードへ移行
JMP.B LABEL
LABEL:
NOP
NOP
NOP
NOP
MOV ...

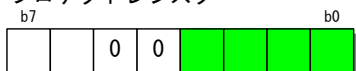
```


2.5 プロテクト機能

プロテクトレジスタ

プログラムが暴走した時に備え、重要なレジスタは簡単に書き換えができないよう保護されている

プロテクトレジスタ



PRCR 000AH番地 リセット時：0000 0000B

ビット名/ビットシンボル	機能	R	W
CM0、CM1、OCD、FRA0、FRA1FRA2レジスタへの書き込み許可 / PRC0	0：書き込み禁止 1：書き込み許可	○	○
PM0、PM1レジスタへの書き込み許可 / PRC1	0：書き込み禁止 1：書き込み許可	○	○
PD0レジスタへの書き込み許可 / PRC2	0：書き込み禁止 1：書き込み許可 (注1)	○	○
VCR2、VW0C、VW1C、VW2Cレジスタへの書き込み許可 / PRC3	0：書き込み禁止 1：書き込み許可	○	○
予約ビット	書く場合“0”を書くこと	○	○
予約ビット	読んだ場合、その値は“0”	○	—

注1. このビットは“1”を書き込んだ後、任意の番地に書き込みを実行すると“0”になる。
他のビットは“0”にならないので、プログラムで“0”にすること。



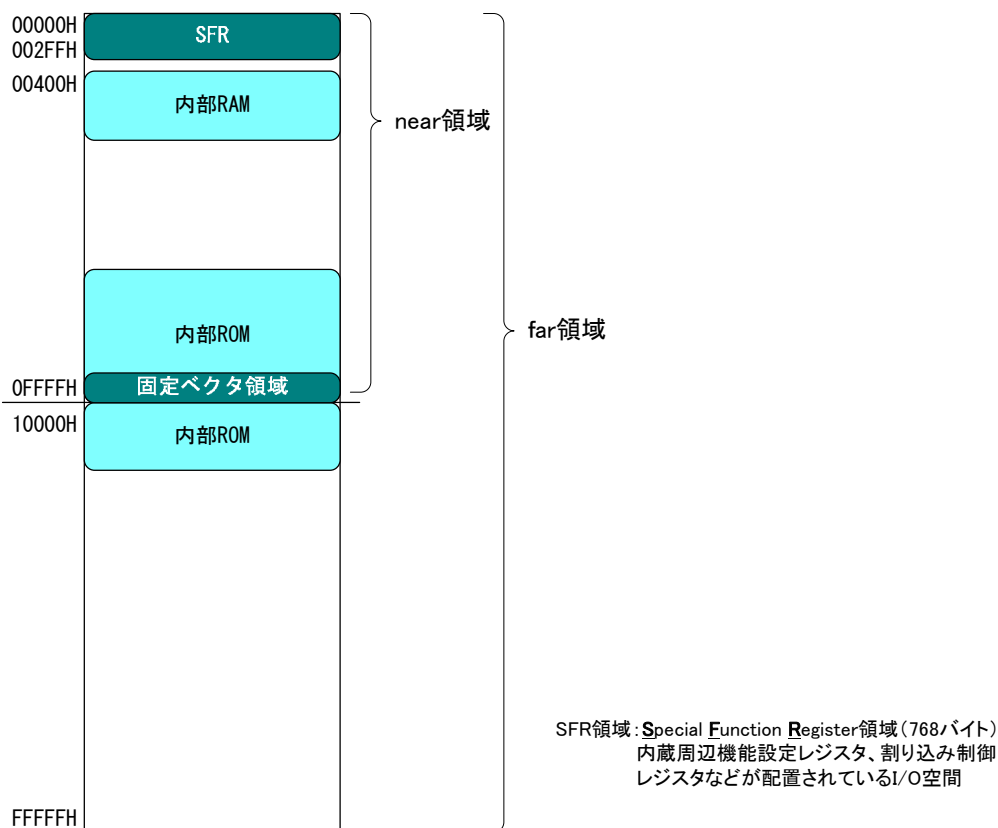
第3章

*R8C*ファミリのアーキテクチャ

- 3.1 SFR領域と固定ベクタテーブル
 - 3.2 レジスタセット
 - 3.3 扱えるデータタイプ
 - 3.4 メモリとレジスタ上のデータ配置
 - 3.5 アドレッシングモード
 - 3.6 命令セット
-

3.1 SFR領域と固定ベクタテーブル

R8Cファミリのアドレス空間



Code : R8Cコース

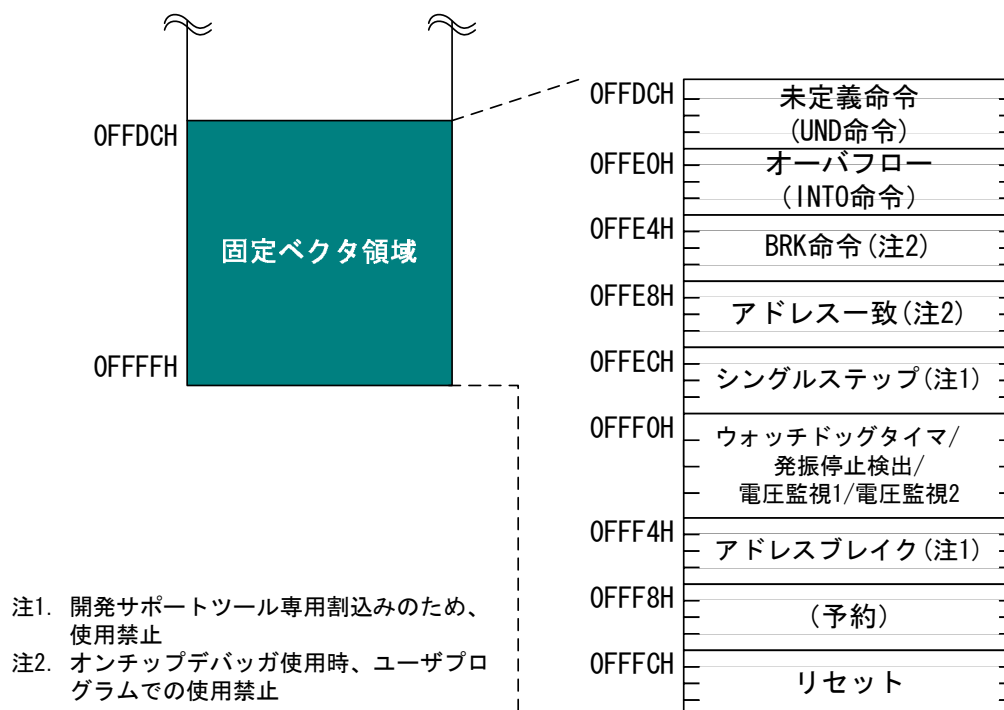
Date : Rev. 2.20

Page: 1 of 21

SFR領域 (Special Function Register)

R8Cファミリでは、00000番地～002FFH番地の768バイトを占有している。すべて予約領域のため、何も配置されていない領域についてもワークRAMとしては使用できない。

固定ベクタ領域の配置



Code : R8Cコース

Date : Rev. 2. 20

Page: 2 of 21

固定ベクタテーブル

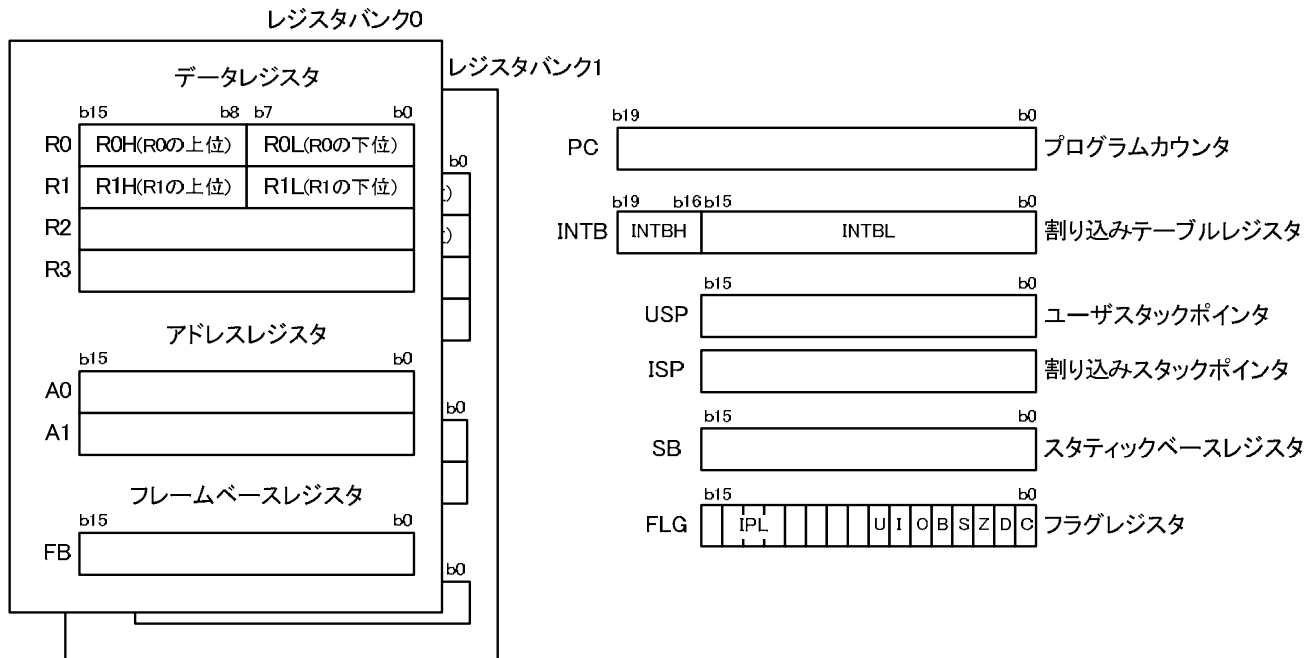
1ベクタに対して4バイトで構成されているテーブル。

各ベクタテーブルには割り込みルーチンの先頭アドレスを設定する。

なお、BRK命令ベクタとリセットベクタを除く各ベクタの最上位バイトは、IDコードチェック機能で判定される「IDコード」の格納領域となる。

3.2 レジスタセット

R8Cファミリのレジスタ構成



Code : R8Cコース

Date : Rev. 2.20

Page:3 of 21

データレジスタ (R0, R1, R2, R3)

16ビットで構成されており、主に転送や算術、論理演算に使用する。

アドレスレジスタ (A0, A1)

16ビットで構成されており演算以外に、間接／相対アドレッシング時のベースアドレス格納用レジスタとして使用する。

フレームベースレジスタ (FB)

R8C/Tinyシリーズ用Cコンパイラ (NC30) で使用する16ビットのレジスタ。auto変数と引数を相対アドレッシングでアクセスする際のベースアドレス格納用として使用する。

スタティックベースレジスタ (SB)

変数のアクセス効率を向上させるために用意されている16ビットのレジスタ。変数の相対アドレッシング時にベースアドレスを格納するレジスタとして使用する。

フラグレジスタ (FLG)

1ビット単位のフラグと3ビットのプロセッサ割り込み優先レベル (IPL) で構成。

割り込みテーブルレジスタ (INTB)

可変割り込みベクタの先頭アドレスを格納する20ビットのレジスタ。

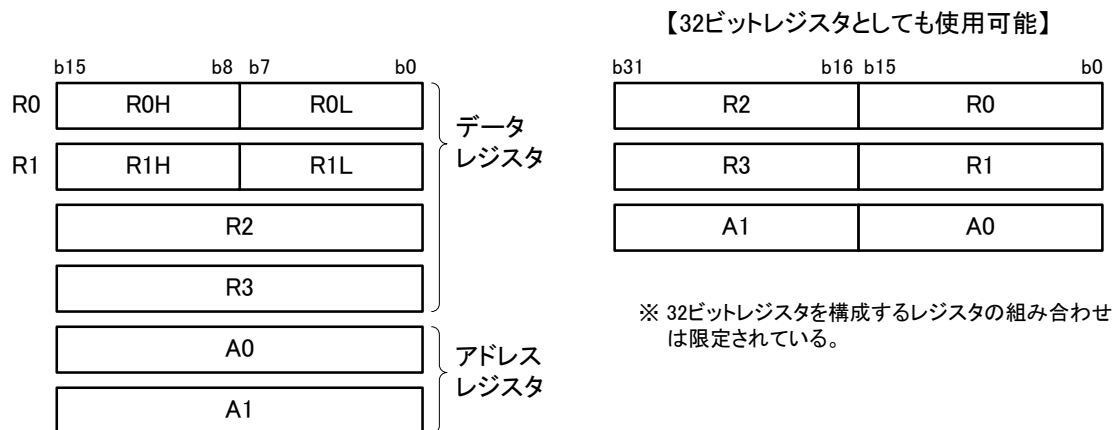
ユーザスタックポインタ (USP)／割り込みスタックポインタ (ISP)

スタックポインタは2種類あり、スタックポインタ指定フラグ (Uフラグ) で切り換える。

プログラムカウンタ (PC)

次に実行する命令のアドレスを示す20ビットのレジスタ。

汎用レジスタの種類



Code : R8Cコース

Date : Rev. 2.20

Page:4 of 21

データレジスタ (R0, R1, R2, R3) の扱い

標準は16ビット構成のレジスタであるが、R0とR1は上位 (R0H, R1H) と下位 (R0L, R1L) を別々に8ビットのレジスタとして使用できる。

また、R2とR0、R3とR1を組み合わせることで32ビットのレジスタとしても使用可能。

フラグレジスタのビット構成



IPL : プロセッサ割り込み優先レベル (Interrupt Priority Level)
レベル0～レベル7を指定可能。数値が大きいほど優先レベルが高くなる。

U :スタックポインタ指定フラグ
U = '0' で ISP、U = '1' で USP を使用する。

I : 割り込み許可フラグ
I = '0' で割り込み禁止、I = '1' で割り込み許可。

0 : オーバフローフラグ
演算結果がオーバフローしたとき(8ビット時: -128~+127、
16ビット時: -32768~+32767)に、0='1'となる。

B :レジスタバンク指定フラグ
B = '0' のときレジスタバンク0を使用、B = '1' のときレジスタバンク1を使用する。

S :サインフラグ
演算結果が負のとき S = '1' になる。

Z : ゼロフラグ
演算結果が '0' のとき Z = '1' になる。

D :デバッグフラグ
シングルステップ割り込みを許可するフラグ。D = '1' で1命令実行後にシングルス
ステップ割り込みが発生。割り込み要求受け付け後、D = '0' となる。

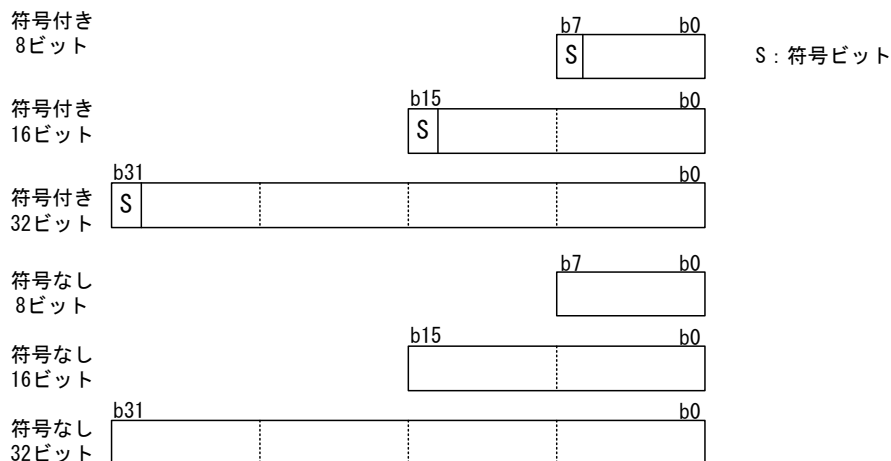
C : キャリーフラグ
算術演算ユニットで発生したキャリー(桁上がり時C='1')、ボロー(桁借り時C='0')、シフトアウトしたビットを保持する。

3.3 扱えるデータタイプ

扱えるデータタイプ

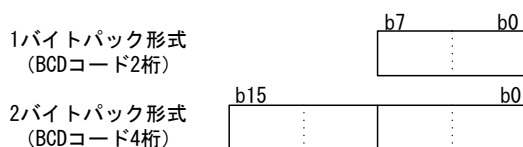
【整数 (Integer)】

整数は符号付きと符号なしがあり、符号付きの場合 最上位ビットがサインビット。



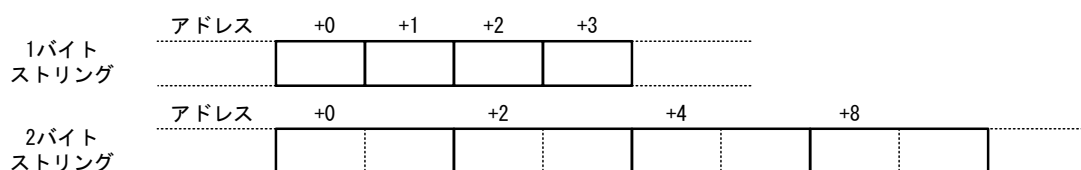
【10進 (BCDコード)】

10進演算命令 DADC, DADD, DSBB, DSUB の4命令で使用可能。



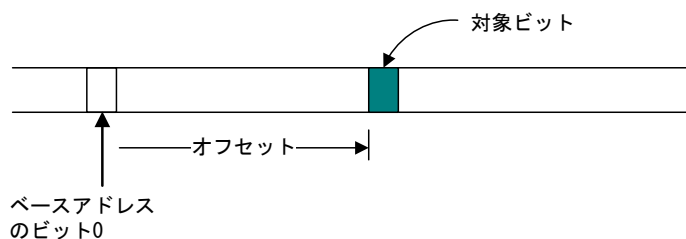
【STRING (String)】

任意の長さだけ連続して並べたデータ。STRING命令 SMOVB, SMOVF, SSTR の3命令で使用可能。



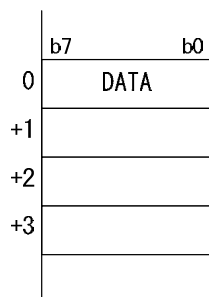
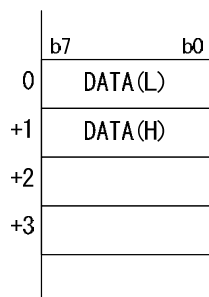
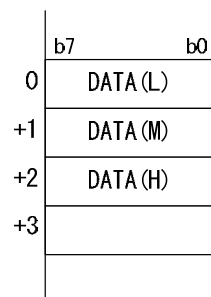
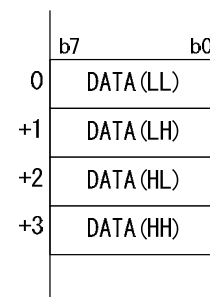
【ビット (Bit)】

ビットはレジスタ上のビットとメモリ上のビットがあり、メモリ上のビットについてはアドレッシングモードによってビットの指定方法と指定範囲が異なる。



3.4 メモリとレジスタ上のデータ配置

メモリ上のデータ配置

バイト(8ビット)
データワード(16ビット)
データアドレス(20ビット)
データロングワード(32ビット)
データ

R8Cファミリのエンディアンは「リトルエンディアン」である。
したがって2バイト以上のデータは、データの下位バイト側からメモリ上に配置される。

R8Cファミリのアドレッシングモード

アドレッシング名称	アドレッシング		
	一般命令	特定命令	ビット命令
即値	○	×	—
レジスタ直接	○	○	○
絶対	○	○	○
アドレスレジスタ間接	○	○	○
アドレスレジスタ相対	○	○	○
SB相対	○	×	○
FB相対	○	×	○
スタックポインタ相対	○	×	×
プログラムカウンタ相対	×	○	×
専用レジスタ直接	×	○	×
FLG直接	×	×	○

○: 該当アドレッシングあり

×: 該当アドレッシングなし

Code : R8Cコース

Date : Rev. 2.20

Page: 8 of 21

一般命令アドレッシング

00000H番地～0FFFFH番地の領域 (near領域) をアクセスするアドレッシング。

特定命令アドレッシング

00000H番地～FFFFFH番地のフルアドレス空間 (far領域)、および専用レジスタをアクセスするアドレッシング。一般命令アドレッシングと異なり、使用できる命令が限定される。

ビット命令アドレッシング

ビット命令でのみ指定可能なアドレッシングで、00000H番地～0FFFFH番地のnear領域内で有効。ベースアドレスのbit0から最大64Kビット先までアクセスが可能。

※ 内蔵ROMが64Kバイト未満の製品は、near領域にのみメモリが配置されるため、特定命令アドレッシングでfar領域 (10000h番地以降) をアクセスすることはない。

3.5.1 間接、相対アドレッシング

アドレスレジスタ間接、相対

(一般命令アドレッシング)

アドレスレジスタ間接		
[A0] [A1]	<p>アドレスレジスタ (A0/A1) の内容で示した値が演算対象の実行アドレスとなる。</p> <p>実行アドレスの範囲は、00000H ~ 0FFFFH番地</p>	<p>レジスタ address</p> <p>メモリ</p>
アドレスレジスタ相対		
dsp:8[A0] dsp:8[A1] dsp:16[A0] dsp:16[A1]	<p>ディスプレイースメント (dsp) で示した値 (0~FFFFH) にアドレスレジスタ (A0/A1) の内容を符号なしで加算した結果が演算対象の実行アドレスとなる。</p> <p>ただし、演算結果が0FFFFH番地を超える場合、17ビット以上は無視され、00000H番地に戻る。</p>	<p>レジスタ address</p> <p>dsp</p> <p>+</p> <p>メモリ</p>

Code : R8Cコース

Date : Rev. 2. 20

Page:9 of 21

使用例を以下に示す。

```

int *sym1_n;
void main(void)
{
    *sym1_n = 1;

    *(sym1_n+1) = 1;
}

```

→

```

/* アドレスレジスタ間接を使用 */
mov.w _sym1_n:16, A0
mov.w #0001H, [A0]

/* アドレスレジスタ相対を使用 */
mov.w _sym1_n:16, A0
mov.w #0001H, 02H[A0]

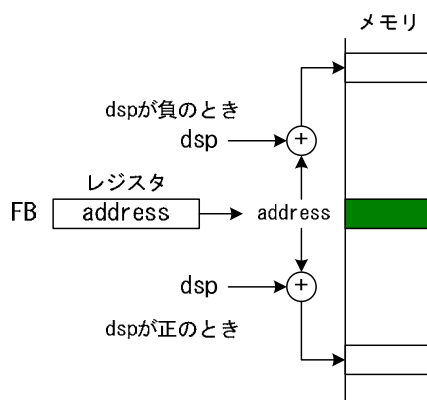
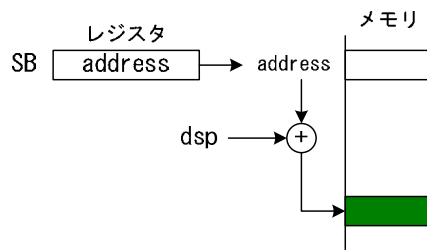
```

3.5.2 SB、FB相対アドレッシング

SB、FB相対

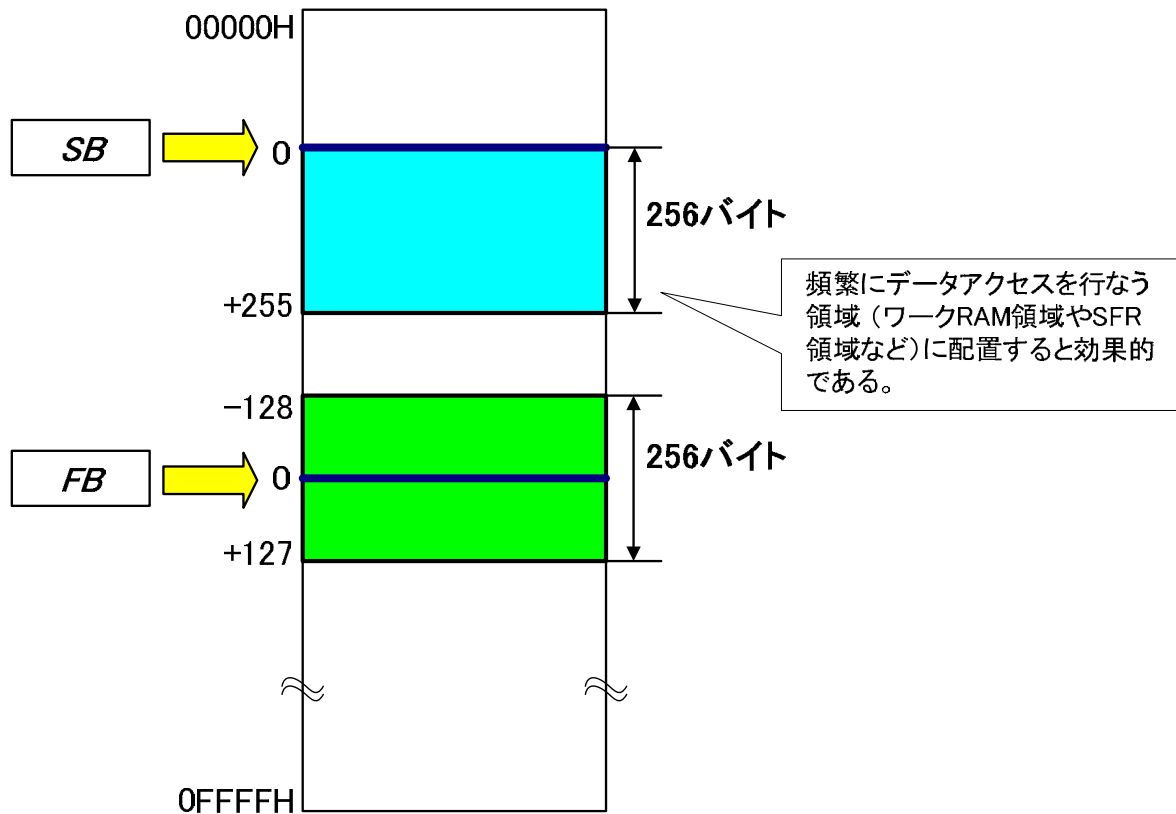
(一般命令アドレッシング)

SB相対		メモリ
dsp:8[SB] dsp:16[SB]	<p>スタティックベースレジスタ (SB) の内容で示したアドレスにディスプレースメント (dsp) で示した値 (0~FFFFHまで) を符号なしで加算した結果が演算対象の実行アドレスとなる。</p> <p>ただし、加算結果が0FFFFH番地を超える場合、17ビット以上は無視され、00000H番地に戻る。</p>	
FB相対		メモリ
dsp:8[FB]	<p>フレームベースレジスタ (FB) の内容で示したアドレスにディスプレースメント (dsp) で示した値 (-128~+127まで) を符号付きで加算した結果が演算対象の実行アドレスとなる。</p> <p>ただし、加算結果が00000H~0FFFFH番地を超える場合、17ビット以上は無視され、00000H番地側、または0FFFFH番地側に戻る。</p>	



3.5.2 SB、FB相対アドレッシング

SB相対、FB相対アドレッシング使用方法(1)



Code : R8Cコース

Date : Rev. 2.20

Page: 11 of 21

SB相対アドレッシングの使用例

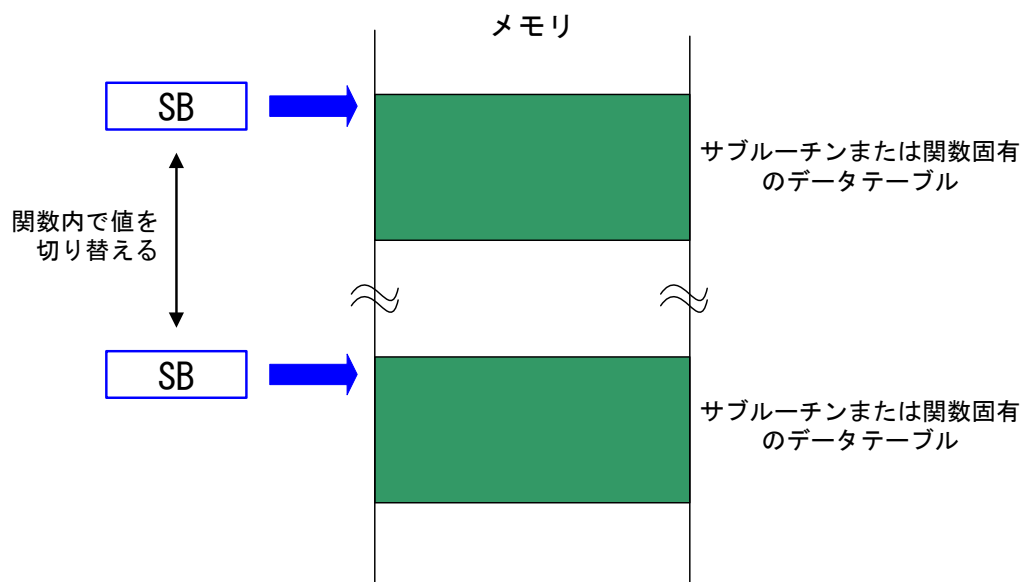
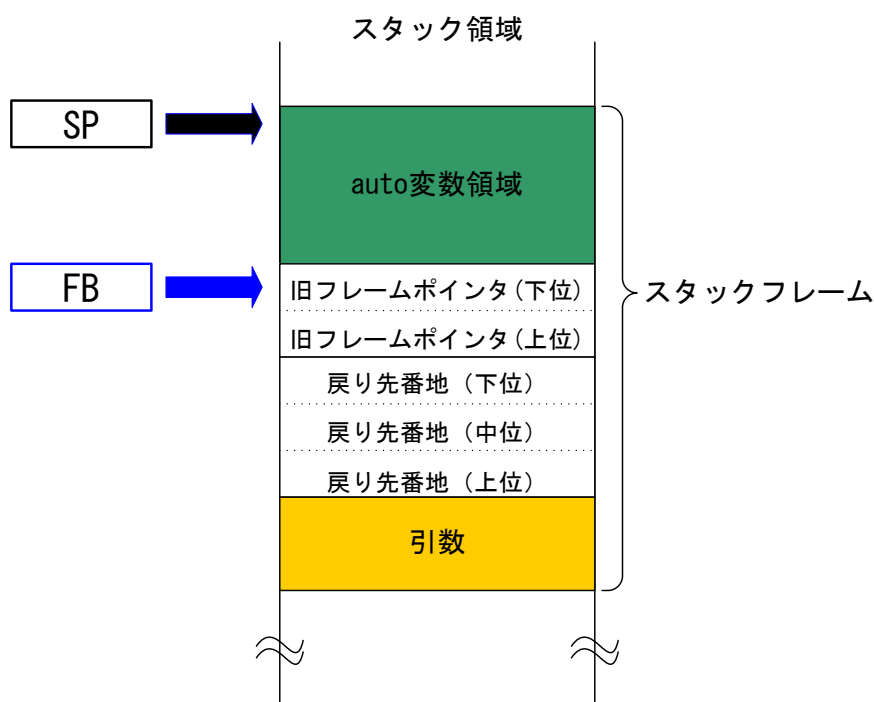
R8Cファミリ用アセンブラ (AS30) では、SBレジスタで指定したアドレスから256バイト内の領域 に存在する変数をプログラム中でアクセスした場合、SBレジスタ相対アドレッシングでアクセスするコード(機械語)に展開する最適化を行う。

※ NC30の拡張機能によりC言語で記述した場合でも、SB相対アドレッシングでの変数アクセスが可能。

3.5.2 SB、FB相対アドレッシング

SB、FB相対アドレッシング
使用方法 (2)

◆ SBレジスタの動的制御

◆ FBレジスタを使用したauto変数および引数領域へのアクセス
(Cプログラミング時、コンパイラがFB相対アドレッシングを占有する)

3.6 命令セット

R8Cファミリの命令セット

命令数:89命令

大分類	中分類	大分類	中分類
転送命令 17命令	転送命令 実行アドレス転送命令 4ビットデータ転送命令 プッシュ/ポップ命令 拡張データ領域転送命令 条件付き転送命令 データ交換命令 ストリング命令	分岐命令 8命令	無条件分岐命令 条件分岐命令 間接分岐命令 加算/減算&条件分岐命令 サブルーチンコール命令 間接分岐サブルーチンコール命令 サブルーチン復帰命令
	演算命令 31命令	ビット処理命令 14命令	ビット論理演算命令 ビットセット/ビットクリア命令 ビットテスト命令 条件ビット転送命令
	加算命令 減算命令 乗算命令 除算命令 10進加算命令 10進減算命令 インクリメント/デクリメント命令 積和演算命令 論理演算命令 シフト/ローテート命令 テスト命令 絶対値, 2の補数, 符号拡張命令	その他の命令 19命令	専用レジスタ転送命令 フラグレジスタ操作命令 C言語サポート命令 OSサポート命令 デバッグサポート命令 ソフトウェア割り込み命令 割り込み復帰命令 ウェイト命令 ノーオペレーション命

Code : R8Cコース

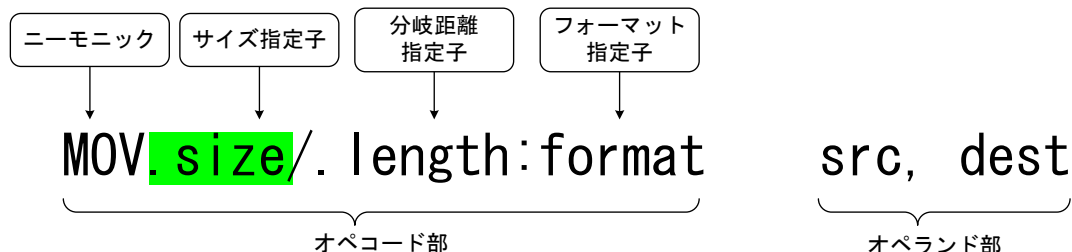
Date : Rev. 2.20

Page:13 of 21

R8Cファミリでは、C言語でプログラミングした場合のオブジェクト効率向上のため、様々な命令が用意されている。

3.6.1 命令の記述方法

命令の記述形式



記述例) **MOV. W R0, R1**

- **ニーモニック** : 命令の動作を示す。
- **サイズ指定子** : ニーモニックの処理対象となるメモリやレジスタのデータサイズを指定する。
- **分岐距離指定子** : 分岐命令及びサブルーチンコール命令の分岐先への距離を指定する。
(通常は省略。)
- **命令フォーマット指定子** : オペコード形式を指定する。オペコード形式によって、オペコード及びオペランドのコード長が異なる。(通常は省略)

Code : R8Cコース

Date : Rev. 2.20

Page: 14 of 21

各指定子の種類

サイズ指定子	内容
.B	バイト (8ビット) サイズを示す
.W	ワード (16ビット) サイズを示す
.L	ロングワード (32ビット) サイズを示す

分岐距離指定子	内容
.S	分岐距離 : +2~+9 (3ビット前方PC相対)
.B	分岐距離 : -128~+127 (8ビットPC相対)
.W	分岐距離 : -32768~+32767 (16ビットPC相対)
.A	分岐距離 : 0~FFFFFFH (20ビット絶対)

フォーマット指定子	内容
:Z	ゼロ形式 (オペコード1バイト展開)
:S	ショート形式 (オペコード1バイト展開)
:Q	クイック形式 (オペコード2バイト展開)
:G	ジェネリック形式 (オペコード2バイト展開)

選択優先順位

高

↑

↓

低

3.6.2 転送命令

転送命令

ニーモニック	記述形式
MOV	MOV. size src, dest

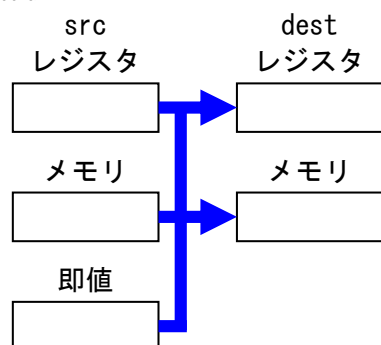
◆ 記述例1) サイズ指定子が. B

MOV. B R0L, R1L
 MOV. B Ram1, Ram2
 MOV. B #12H, R0L

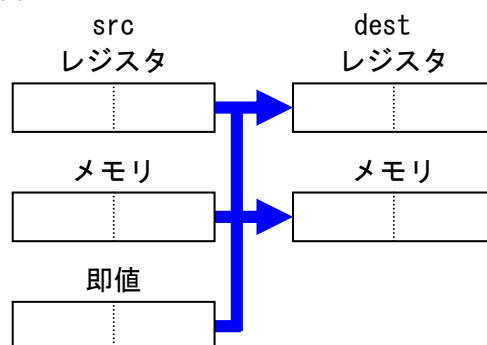
◆ 記述例2) サイズ指定子が. W

MOV. W R0, R1
 MOV. W Ram1, Ram2
 MOV. W #1234H, R0

◆ 動作例



◆ 動作例



Code : R8Cコース

Date : Rev. 2.20

Page:15 of 21

転送命令

メモリーメモリー間の直接転送も4サイクルで実行可能。

3.6.2 転送命令

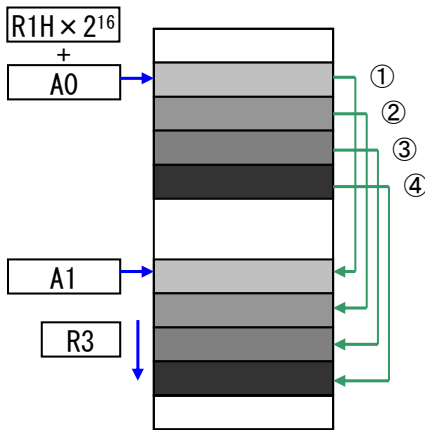
連続データの転送

◆ スtring命令

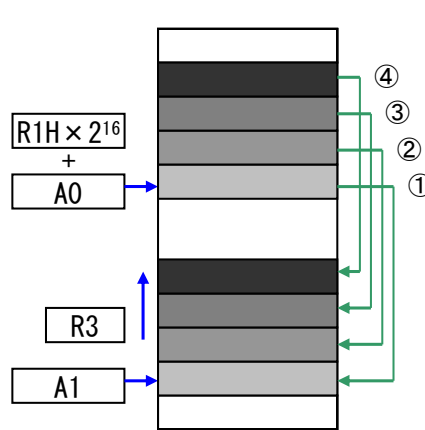
ニーモニック	記述形式
SMOVF	SMOVF. size
SMOVB	SMOVB. size
SSTR	SSTR. size

転送元アドレス : $R1H \times 2^{16} + A0$
 転送先アドレス : A1
 転送回数 : R3
 転送サイズ : .B → バイト転送
 : .W → ワード転送

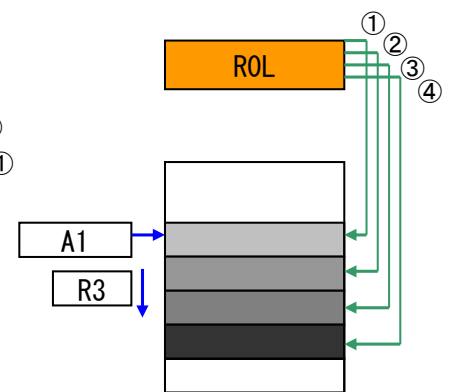
◆ 動作例 SMOVF. B



◆ 動作例 SMOVB. B



◆ 動作例 SSTR. B



Code : R8Cコース

Date : Rev. 2. 20

Page: 16 of 21

利用例

ブロック転送や初期値転送

上記の命令は、スタートアッププログラムの変数領域初期化関数で使用されている。

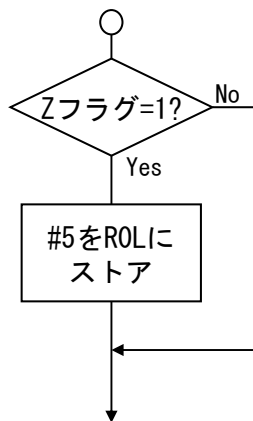
3.6.2 転送命令

条件付き転送

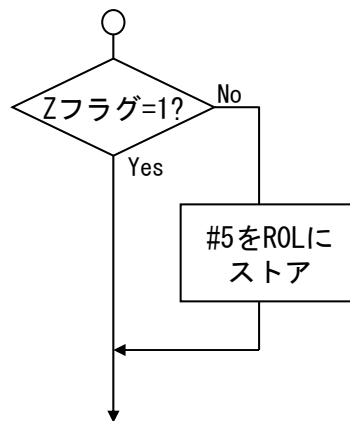
◆ 条件ストア命令

ニーモニック	記述形式
STZ	STZ src, dest
STNZ	STNZ src, dest
STZX	STZX src1, src2, dest

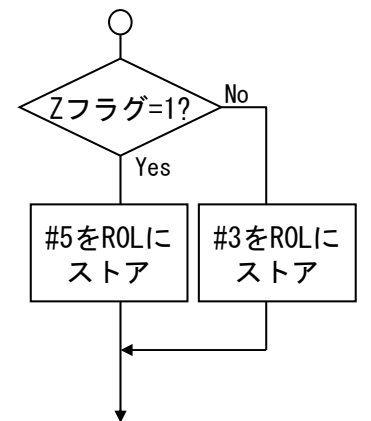
◆ 動作例 STZ #5, ROL



◆ 動作例 STNZ #5, ROL



◆ 動作例 STZX #5, #3, ROL



Code : R8Cコース

Date : Rev. 2.20

Page:17 of 21

上記の命令は、C言語の制御文の展開に使用される。

※ 上記の命令に展開された場合、ソースラインデバッグ時に、分岐先にブレイクポイントが設定できなくなるので、注意が必要。以下に例を示す。

```

void main(void)
{
    char data1, data2;

    data1++;
    if(data1 ==5){ ----->    cmp. b    #05H, _data1
                                stz      #00H, _data2
    }

    if(data1 == 10){ ---->    cmp. b    #0AH, _data1
                                stzx     #00H, #03H, _data2
    }else{
        data2 = 3;
    }
}
  
```

分岐先に“Break Point”が設定できない

3.6.3 演算命令

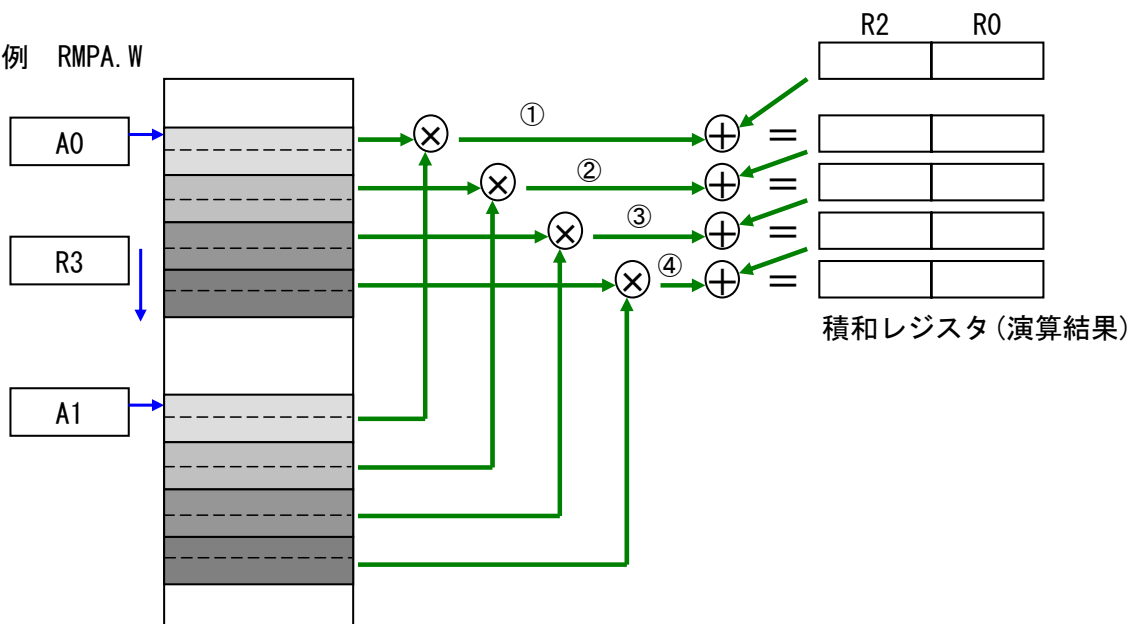
積和演算

◆ 積和演算命令

ニーモニック	記述形式
RMPA	RMPA. size

被乗数アドレス : A0
 乗数アドレス : A1
 積和回数 : R3
 積和レジスタ : R2R0 (.W指定時) または
 (演算結果) R0 (.B指定時)

◆ 動作例 RMPA.W



Code : R8Cコース

Date : Rev. 2.20

Page:18 of 21

NC30では以下の条件を満たしているC言語の記述であれば、RMPA命令を使用したコードに展開される。また、アセンブラマクロ関数を使用することでもRMPA 命令を使用できる (詳細は後述)。

- (1) ループ回数が明確である積和演算処理である。
- (2) signed char 型のint 型へのキャスト指定がある積和演算、またはint 型のlong 型へのキャスト指定がある積和演算である。
- (3) 被乗数、乗数ともにvolatile 修飾されていない。
- (4) 最適化オプション -O3 以上、-OS, -OR のいずれかを指定している。

【記述例】

```
void main(void)
{
    static int  str1[] = {0x1234, 0x5678, 0x9abc, 0xdef0, ... };
    static int  str2[] = {0x0012, 0x0034, 0x0056, 0x0078, ... };
    static long  data = 0;

    for(i=0 ; i<10 ; i++) {
        data = (long)str1[i] * str2[i] + data; /* 積和演算 */
    }
}
```

3.6.4 分岐命令

サブルーチンコール

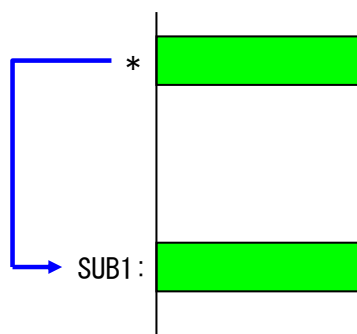
◆ サブルーチンコール命令

ニーモニック	記述形式
JSR	JSR (.length) label

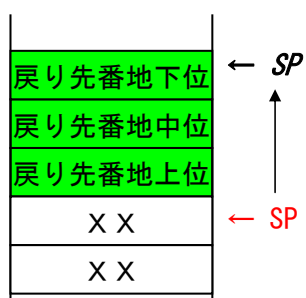
分岐距離指定子 (.length) は、アセンブラが最適なものを選択する。
(JSR命令については .Wと .Aのみ選択可)

◆ 動作例

JSR SUB1



JSR命令実行後の
スタックの状態



3.6.4 分岐命令

条件分岐

◆ 条件分岐命令

ニーモニック	記述形式
JCnd	JCnd label

Cndで示した条件が
真であればlabelへ分岐

◆ 単独フラグに対する判断

Cnd	条件		式	Cnd	条件		式
GEU/C	C=1	等しいまたは大きい	≤	LTU/NC	C=0	小さい	>
EQ/Z	Z=1	等しい	=	NE/NZ	Z=0	等しくない	≠
PZ	S=0	正またはゼロ	0≤	N	S=1	負	0>
O	O=1	○フラグが “1”		NO	O=0	○フラグが “0”	

◆ 複数のフラグに対する一括判断

Cnd	条件		式	Cnd	条件		式
GTU	$C \wedge \overline{Z}=1$	大きい	$<$	LEU	$C \wedge \overline{Z}=0$	等しいまたは 小さい	\geq
GE	$S \vee 0=0$	等しいまたは 符号付きで大きい	\leq	LE	$(S \vee 0) \vee Z=1$	等しいまたは 符号付きで小さい	\geq
GT	$(S \vee 0) \vee Z=0$	符号付きで大きい	$<$	LT	$S \vee 0=1$	符号付きで小さい	$>$

Code : R8Cコース

Date : Rev. 2.20

Page: 20 of 21

C言語の制御文で使用される。<、≥ など二つ以上のフラグ判断が必要な場合でも、上記一命令で判断分岐が可能となる。

```

void main(void)
{
    char data1, data2;

    /* data1が5以下なら分岐 */
    if(5 < data1) {
        data2 = 0;
    }

    /* data1が5より大きいなら分岐 */
    if(5 >= data1) {
        data2 = 3;
    }
}

```

→

```

cmp.b    #05H, _data1
jleu     L13
mov.b    #00H, _data2
L13:

```

→

```

cmp.b    #05H, _data1
jgtu     L18
mov.b    #03H, _data2
L18:

```

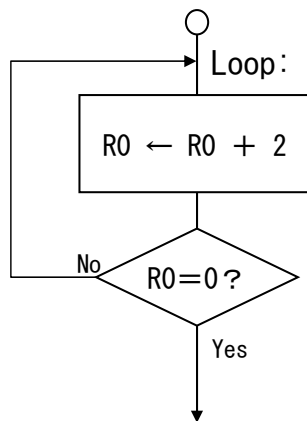
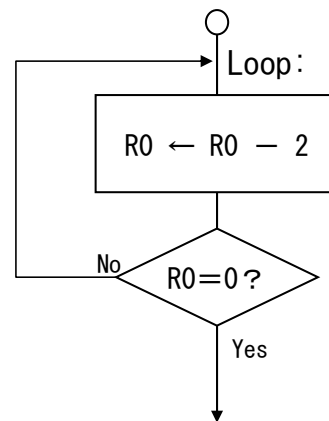
3.6.4 分岐命令

加算／減算 & 条件分岐

◆ 加算／減算 & 条件分岐命令

ニーモニック	記述形式		機能
ADJNZ	ADJNZ. size	src, dest, label	加算&条件分岐
SBJNZ	SBJNZ. size	src, dest, label	減算&条件分岐

◆ 動作例

ADJNZ. W #2, R0, Loop*SBJNZ. W #2, R0, Loop*

Code : R8Cコース

Date : Rev. 2. 20

Page: 21 of 21

以下に示すように、C言語の繰り返し文を記述した場合に上記の命令で展開されることがある。

```

void main(void)
{
    int i = 0xff;

    /* iが'0'になるまで減算 */
    do{
        i -= 3;
    }while(i != 0);
}

```

L18: *adjnz. w #-3, _i, L18*

第4章

NC30の実装仕様

- 4.1 概要
 - 4.2 データ型と修飾子
 - 4.3 絶対番地へのアクセス
 - 4.4 メモリ配置
-

4.1 概要

コンパイラの特長

- ANSI-C※ 仕様をベースにしたC/C++コンパイラ
- 最適化機能、CPUアーキテクチャによる優れたコード生成効率
- 関数毎に使用スタックサイズの算出が可能
- システム内のデータ使用状況情報の出力が可能
- 豊富な組み込み用拡張機能
 - ◆変数の絶対アドレス指定
 - ◆インラインアセンブル機能
 - ◆アセンブラ関数への引数のレジスタ渡し指定
 - ◆割込み処理関数の直接記述 etc
- μ ITRON M3T-MR30対応拡張機能(M16Cシリーズ対象)

Code : R8Cコース

Date : Rev. 2.20

Page:1 of 22

NC30は市販されている標準的なC言語の使用に加え、ROM化を容易に行うための拡張機能を備えている。

※ ANSI ; American National Standards Instituteの略称

4.2 データ型と修飾子

基本データ型

データ型	サイズ	符号指定のない場合の扱い
_Bool	8 ビット	指定不可（符号なし）
char	8 ビット	unsigned
short (int)	16 ビット	signed
int	16 ビット	signed
long (int)	32 ビット	signed
long long (int)	64 ビット	signed
float	32 ビット	signed
double	64 ビット	signed
long double	64 ビット	signed

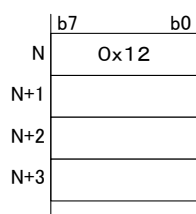
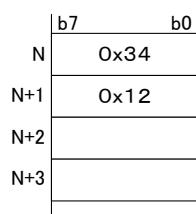
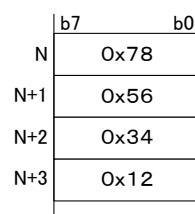
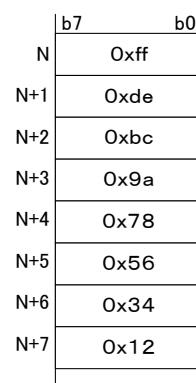
Code : R8Cコース

Date : Rev. 2.20

Page:2 of 22

_Bool型は符号指定はできない。その他のデータ型で符号指定がない場合、char型のみ unsigned型と解釈し、それ以外の型ではsigned型として解釈する。

【データ型】

*char = 0x12;**int = 0x1234;**long = 0x12345678;**long long =
0x123456789abcdeff;*_Bool
charshort
int
nearポインタlong
farポインタ
floatlong long
double
long double

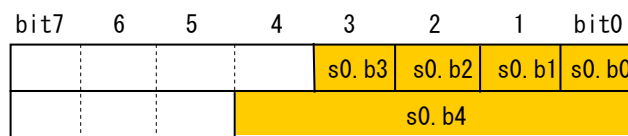
ビットフィールド構造体

割り付けルール

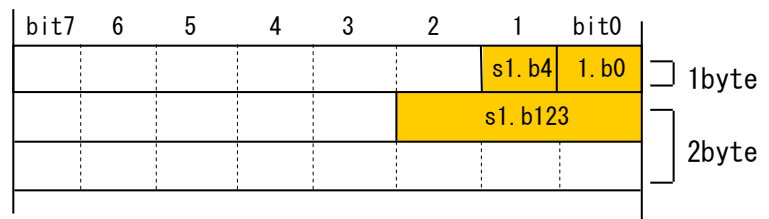
- ◆ メンバの宣言順に下位ビットから割り付け
- ◆ データ型の異なるメンバは次のアドレスに配置

割り付け例

```
struct ex0 {
    char    b0 : 1 ;
    char    b1 : 1 ;
    char    b2 : 1 ;
    char    b3 : 1 ;
    char    b4 : 5 ;
} s0 ;
```



```
struct ex1 {
    char    b0 : 1 ;
    int     b123: 3 ;
    char    b4 : 1 ;
} s1 ;
```



Code : R8Cコース

Date : Rev. 2. 20

Page:3 of 22

NC30では、型によって確保するフィールドのバイト数が異なる。

char型 → 1バイト

int型 → 2バイト

long int型 → 4バイト

※ メンバの型は、符号指定がない場合“unsigned”とみなす。

※ long long (int)型のビットフィールドは宣言できない。

ポインタ・列挙型

データ型	サイズ	指定のない場合の符号
near ポインタ	16 ビット	unsigned
far ポインタ	32 ビット	unsigned
列挙型	16 ビット	unsigned

列挙型はunsigned intと同様に扱われる。
-fchar_enumerator (-fCE) オプションにより
unsigned charと同様に扱うことも可能。

Code : R8Cコース

Date : Rev. 2.20

Page:4 of 22

※ 内部ROMが64Kバイト未満の製品は、10000H番地以降にメモリは配置されないため、farポインタを使用することはない。

near/far修飾子について

near/far領域

- near領域 — 0H~0FFFFH番地の64Kバイトの領域（一般命令アドレッシングでアクセスできる空間）。
- far領域 — 0H~FFFFFFH番地の全メモリ空間（特定命令アドレッシングでアクセスする空間）。
 (R8C/TinyのROM 64Kバイト未満の製品の場合、10000H~FFFFFFH番地は使用不可)

near/far修飾された変数の扱い

- near修飾子 —
- ① near属性のセクションに割り当てる
 - ② near属性を操作する命令（一般命令アドレッシング）を出力
 - ③ near属性のセクションをnear領域に割り付ける
- far修飾子 —
- ① far属性のセクションに割り当てる
 - ② far属性を操作する命令（特定命令アドレッシング）を出力
 - ③ far属性セクションをfar領域に割り付ける

Code : R8Cコース

Date : Rev. 2.20

Page: 5 of 22

near/far修飾された変数に対して、①、②はコンパイラが自動的に行う。
 ③については、ユーザーがスタートアッププログラム内で行う必要がある。

【near/far修飾子を省略した場合のデフォルト属性】

分類	属性
変数 (RAM データ)	near
auto 変数／引数 (スタックデータ)	near 固定
定数/const 修飾された変数 (ROM データ)	far ※
関数	far 固定

※ “-R8C” オプションを指定した場合、nearとなる。

静的変数の修飾例

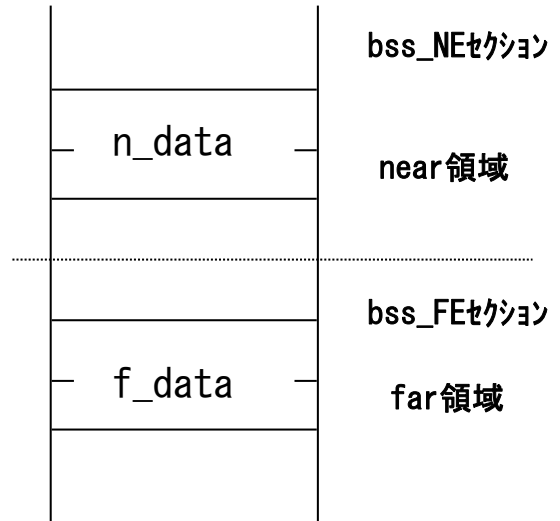
宣 言

near属性セクション
(bss_NE)に割り付け
られる

```
static int n_data ;  
static int far f_data ;
```

far属性セクション
(bss_FE)に割り付
けられる

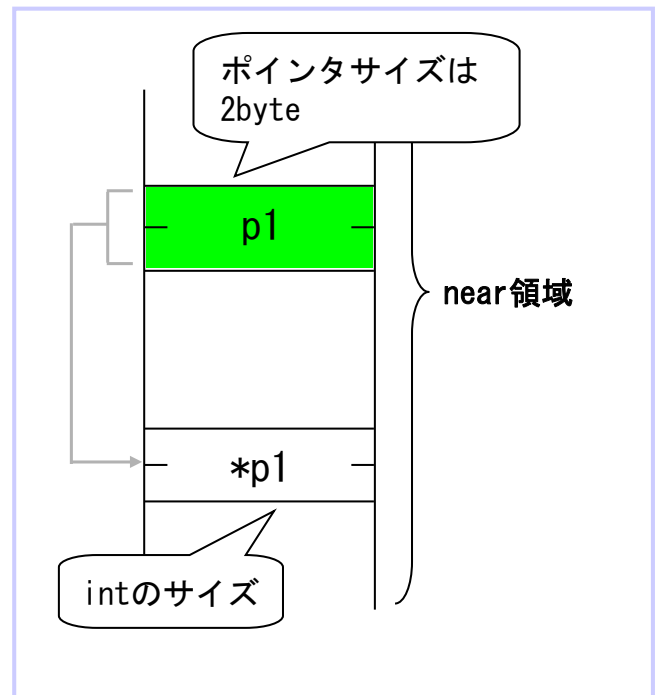
配 置



nearポインタ

```
int *p1 ;  
||  
(int near * near p1 ;)
```

near領域に存在する変数を指す
ポインタが、*near*ポインタとなる

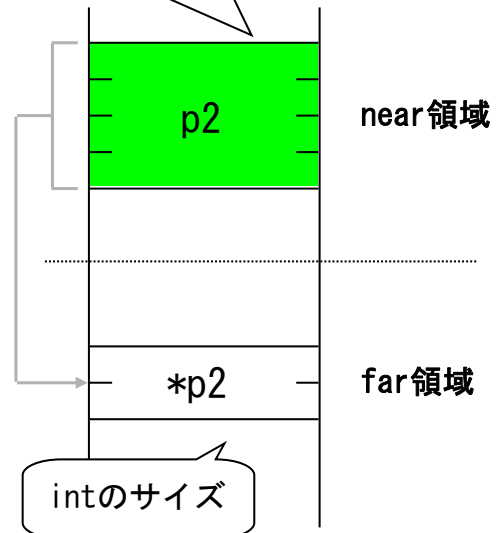


farポインタ

```
int far * p2 ;  
||  
(int far * near p2 ;)
```

far領域に存在する変数を指す
ポインタが、*far*ポインタとなる

ポインタサイズ
は4byte



Code : R8Cコース

Date : Rev. 2.20

Page:8 of 22

R8Cファミリの場合、far領域(10000H番地以降)には内部ROMのみが配置され、外付けRAMの割り付けはできない。そのため、farポインタを使用するのは以下の場合に限られる。

10000H番地以降のROM領域に配置されている文字列または定数データをポインタを使用してアクセスする場合。

const型修飾子

const修飾子の意味

修飾子のついたオブジェクトに対する明示的な代入をチェックする。
(値の変更ができなくなる。)

NC30での扱い

- ◆明示的な代入に対してウォーニングを出力
- ◆指定された変数をROM属性のセクションに配置し、RAMには領域を確保しない。

プログラム例

【例 1】

```
const char c = 10 ;
```

```
:
```

```
c = 5 ;
```

コンパイル時に
チェックされる

【例 2】

```
const char *p ;
```

```
:
```

```
p = ( char * ) 0x30 ;
```

ok

```
*p = 0x40 ;
```

ワーニング

Code : R8Cコース

Date : Rev. 2. 20

Page: 9 of 22

変数を定数データの扱いとして宣言したい場合などにconst修飾子を用いる。

例)

```
const char seg_data[]
    = {0x01, 0x02, 0x03, .....};    /* LEDセグメントデータ */
```

volatile型修飾子

volatile修飾子の意味

プログラム実行以外の要因（割り込み等）で値が変化するオブジェクトに対する指定。（コンパイラの最適化を抑止し、記述通りの命令コードを生成する）

```
char count;  
volatile char port1 ;  
  
void main( void )  
{  
    port1 = 0;  
    if(port1 == 0) {  
        count++;  
        :  
    }  
}
```

volatile指定がない場合コンパイラにより最適化され、比較命令が出力されない可能性がある。

4.3 絶対番地へのアクセス

ポインタによるアクセス

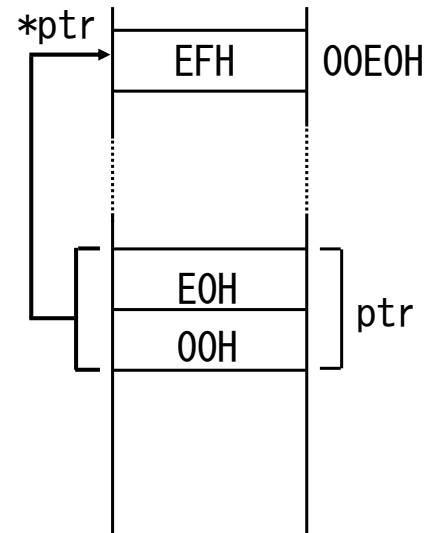
■00E0H番地にEFHを代入する

ポインタ変数を用いたアクセス

```
char *ptr ;  
ptr = ( char * )0x00e0 ;  
*ptr = 0xef ;
```

マクロ定義を使用した場合

```
#define PORT0 (*(char *)0x00e0)  
PORT0 = 0xef ;
```



Code : R8Cコース

Date : Rev. 2.20

Page:11 of 22

通常、C言語で絶対番地をアクセスする場合は、ポインタ変数を用いる。

拡張機能による絶対番地へのアクセス(1)

絶対アドレスの指定方法

`#pragma ADDRESS 変数名 アドレス`

記述例

`#pragma ADDRESS PORT0 00e0h`
`char PORT0 ;`

変数名

数値の記述形式はアセンブラ
(AS30)の数値表現に従う
※ 0x00e0での記述も可能

通常の変数と同じように宣言する

```
void func( void )
{
    :
    PORT0 = 0x0f ;
    :
}
```

展開イメージ

```
_PORT0 .equ 00e0h
:
MOV.B #0f0h, _PORT0
```

`#pragma ADDRESS`は、関数外で宣言された変数に対してのみ有効

`#pragma ADDRESS`で指定した変数は`volatile`修飾されたものとみなされる

Code : R8Cコース

Date : Rev. 2.20

Page:12 of 22

`#pragma ADDRESS`により指定した絶対アドレスは、文字列としてアセンブリ言語ファイルに展開され、アセンブラ指示命令`“ . equ ”`により定義される。
したがって、絶対アドレスはAS30の数値表現に従って記述すること。

拡張機能による絶対番地へのアクセス (2)

絶対アドレスの指定方法

#pragma BITADDRESS 変数名 ビット位置, アドレス

記述例

変数名

ビット位置は0～65535
の範囲で、10進数のみ

```
#pragma BITADDRESS TSTART 0, 0100h
```

```
_Bool TSTART ;
```

_Bool型変数の宣言

数値の記述形式はアセンブラ
(AS30)の数値表現に従う

```
void func( void )
{
    :
    TSTART = 1 ;
    :
}
```

展開イメージ

```
_TSTART .btequ    0, 0100h
:
BSET    _TSTART
```

#pragma BITADDRESSは、関数外で宣言された *_Bool* 型変数に対してのみ有効

#pragma ADDRESSで指定した変数はvolatile修飾されたものとみなされる

Code : R8Cコース

Date : Rev. 2.20

Page:13 of 22

#pragma BITADDRESSにより指定した絶対アドレスは、文字列としてアセンブリ言語ファイルに展開され、アセンブラ指示命令“.btequ”により定義される。
したがって、絶対アドレスはAS30の数値表現に従って記述すること。

SFR領域のアクセス方法

共用体 定義例

```
union SFR {  
    struct {  
        char  b0 : 1 ;  
        char  b1 : 1 ;  
        char  b2 : 1 ;  
        char  b3 : 1 ;  
        char  b4 : 1 ;  
        char  b5 : 1 ;  
        char  b6 : 1 ;  
        char  b7 : 1 ;  
    } bit ;  
    char  byte ;  
} ;
```

プログラム例

```
#pragma ADDRESS PORT0  00e0h  
#pragma ADDRESS DIR0   00e2h  
#pragma BITADDRESS TSTART 0,0100h  
union SFR PORT0, DIR0 ;  
_Bool TSTART ;  
  
void main(void)  
{  
    PORT0.byte = (char)0xc0 ;  
    DIR0.byte  = (char)0xff ;  
    PORT0.bit.b5 = 1 ;  
    :  
    TSTART = 1 ;  
    :  
}
```

Code : R8Cコース

Date : Rev. 2.20

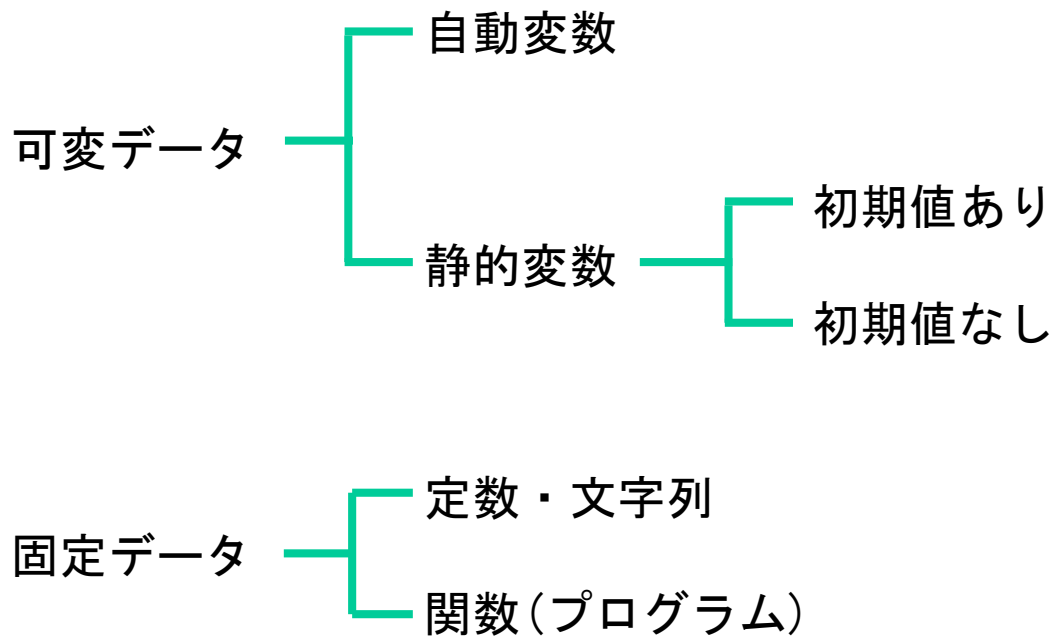
Page:14 of 22

SFR領域(SFR : Special Function Register)には、内蔵周辺機能の制御レジスタ、割り込み制御レジスタ等が配置されている。

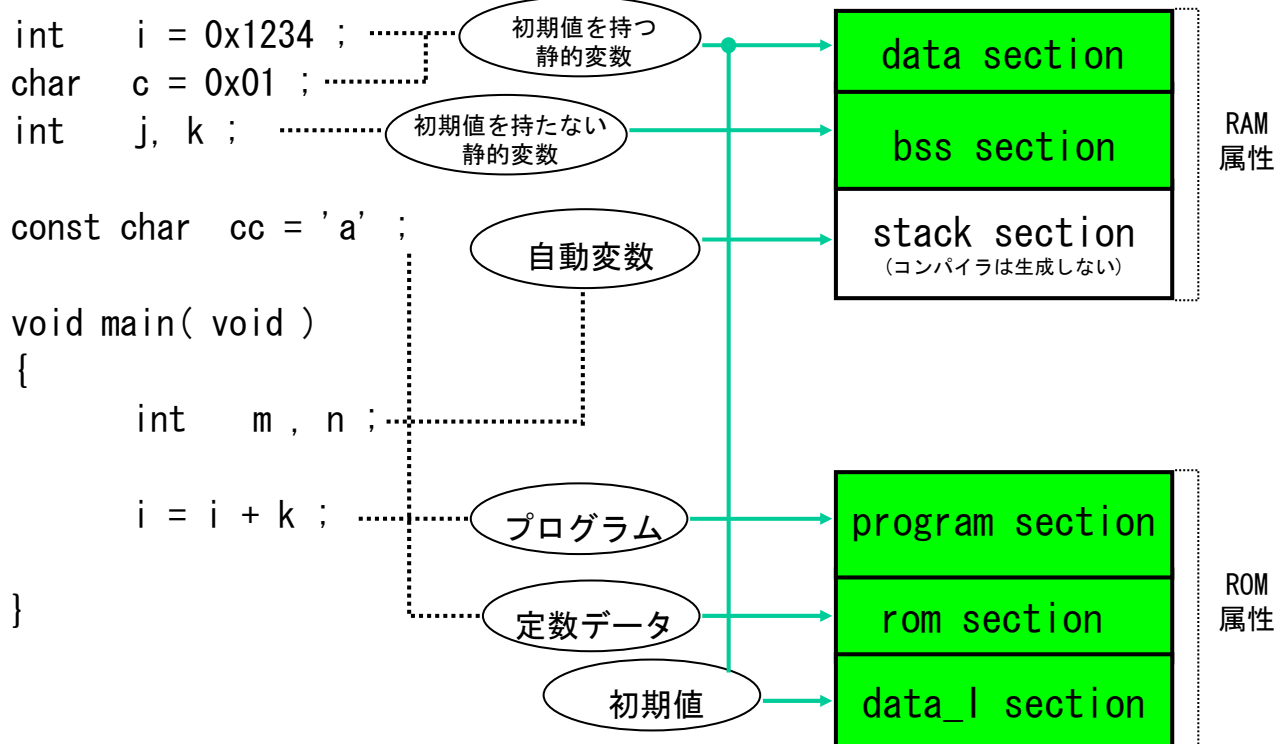
制御レジスタを効率よく操作するには、SFR領域へのアクセスをビット単位／バイト単位で使い分けた方がよい。そのために、SFR領域用の共用体を用意する。

4.4 メモリ配置

プログラムの構成要素



NC30の生成するセクション



Code : R8Cコース

Date : Rev. 2.20


Page:16 of 22

NC30では変数やプログラムを機能ごとに分類し、セクションとして管理している。

初期値をもつ静的変数については、RAM領域 (data section) とROM領域 (data_l section) の両方に配置され、スタートアッププログラム内でROM領域の初期値をRAM領域へコピーする。

セクションの構成

セクションベース名	意 味
data	初期値を持つ静的変数の領域
bss	初期値を持たない静的変数の領域
rom	文字列／定数／const 修飾された変数の領域
program	プログラム領域
stack	スタック領域
heap	ヒープ領域（メモリ管理関数により動的に確保される領域）
vector ※	可変ベクタ領域
svector ※	スペシャルページベクタ領域（R8C/Tiny では未使用）
fvector	割り込みベクタ領域（固定ベクタ）

 : NC30が生成するセクション

※ リンカで自動生成される

Code : R8Cコース

Date : Rev. 2.20

Page:17 of 22

“#pragma SECTION”により、上記のコンパイラが生成するセクション名を変更することができる。

記述例)

```
#pragma SECTION program pro1 ← この宣言以降に記述された関数の
                               セクション名がprogramからpro1
                               に変更される。

void func(void) ← 関数“func”はprogramセクションではなく、
{                pro1セクションに配置される。
    :
    :
}
```

注1) 本機能でセクション名を変更した場合、セクション名の追加あるいは変更を統合環境のリンカオプション設定で行うこと。

注2) データセクション（data, bss, rom）の名称を変更した場合は、セクションベース名の後に属性（_NE, _FE 等）が自動的に付加される。


セクション名の命名規則

セクション名

セクションベース名_属性

属性	意 味
N	near データを格納
F	far データを格納
S	#pragma SBDATA で指定された変数を格納
E	偶数サイズのデータを格納
O	奇数サイズのデータを格納
I	初期値を持つ変数の初期値を格納

セクションベース名		
data	bss	rom

 : 指定可能な属性

Code : R8Cコース

Date : Rev. 2.20

Page:18 of 22

例)

```
int a;
void main(void)
{
    :
}

.section    bss_NE
_a:        .blkb 2
    :
```

【 セクション指示命令 】

セクションの先頭番地を
偶数番地へアライメントする

.SECTION セクション名 [, セクションタイプ, ALIGN]

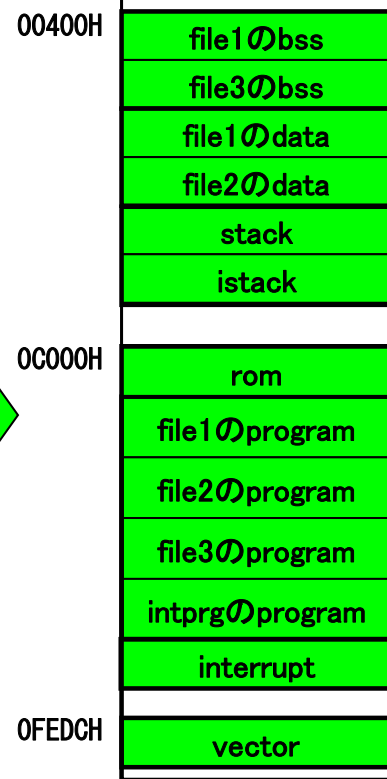
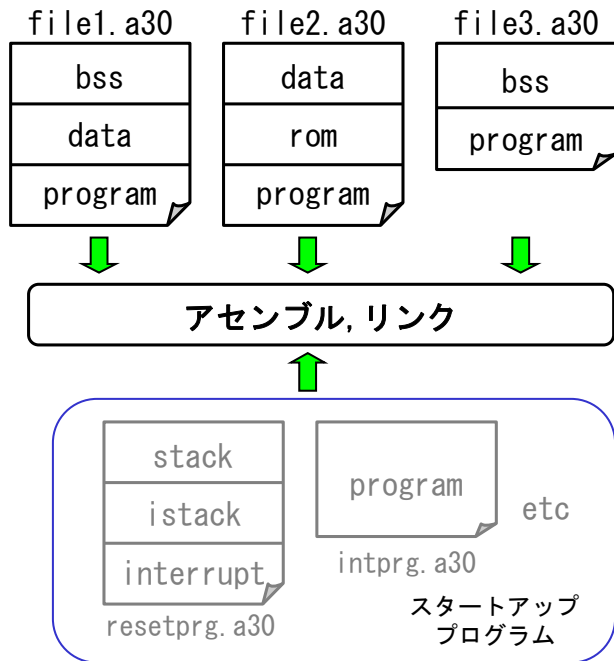
タイプ	内容
CODE (プログラム領域)	<ul style="list-style-type: none"> ・ プログラムを記述する領域を示す ・ CODE タイプのセクションは、ROM 領域に配置すること
DATA (可変データ領域)	<ul style="list-style-type: none"> ・ 可変データを記述する領域を示す ・ DATA タイプのセクションは、RAM 領域に配置すること
ROMDATA (固定データ領域)	<ul style="list-style-type: none"> ・ プログラム以外の固定データを記述する領域を示す ・ ROMDATA タイプのセクションは、ROM 領域に配置すること

セクションの配置規則

リンカオプション指定例

```
optlnk file1 file2 file3 resetprg intprg
-o asb_file -start = bss, data, stack, istack/400,
rom, program, interrupt/0c000, vector/0fedc
```

<abs_fileのセクション配置>



Code : R8Cコース

Date : Rev. 2. 20

Page:19 of 22

SB相対アドレッシングによる変数のアクセス

SB相対(8ビット相対)アドレッシング使用変数宣言

#pragma SBDATA 変数名

記述例

```
#pragma SBDATA x
int x, y;

void main( void )
{
    x = x + y;
}
```

注意点

- ① 外部変数に対してのみ有効
- ② ROMデータに対しては無効

展開イメージ

```
.SBSYM _x
.section program
.glb _main
_main:
    add.w _y, _x
    rts

.section bss_NE, DATA
_y: .blkb 2

.section bss_SE, DATA
_x: .blkb 2
```

変数xに対して指示命令“.SBSYM”を生成

SB相対(8ビット相対)アドレッシングでアクセス可能かどうかは、リンク時に判定

変数xはSBDATA属性のセクションに配置

Code : R8Cコース

Date : Rev. 2. 20

Page: 20 of 22

R8Cファミリでは、SB相対アドレッシングを使用するとコード効率が向上する。
SBレジスタ値の設定、およびコンパイラに対するSBレジスタ値の定義はスタートアッププログラム内で行なう。

ユーティリティ (utl30) によるSBDATA宣言

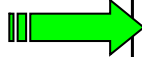
機能

コンパイルオプション“-finfo”の指定で生成されるインスペクタ情報より、**使用頻度の高い変数(外部変数のみ)から順番に“SBDATA宣言(#pragma SBDATA)”を行い、結果をファイルとして出力する。**

プログラム例

```
int data1;
char data2;
long data3;
void main(void)
{
    data2 = data2 + 3;
    data1++;
    :
    if(data1 == 0x00ff){
        data3 = 10;
    }
    :
}
```

使用頻度の高い
ものから順番に
SBDATA宣言される



ヘッダファイル出力例

```
/*
 * #pragma SBDATA Utility
 */
/* SBDATA Size [255] */
#pragma SBDATA data1 /* size = (2) / ref = [ 3] */
#pragma SBDATA data2 /* size = (1) / ref = [ 2] */
#pragma SBDATA data3 /* size = (4) / ref = [ 1] */
:
:
/*
 * End of File
 */
```

SB相対アドレッシングで
アクセスされる外部変数

データのサイズ

データの使用頻度

Code : R8Cコース

Date : Rev. 2.20

Page: 21 of 22

処理対象となる変数の型

- ・Bool, char, short, int, long, longlong

処理対象外となる変数

- ・#pragma ADDRESS で宣言された変数
- ・#pragma ROM で宣言された変数
- ・#pragma SECTION で変更されたセクションに配置された変数
(-fsectionの指定で処理対象となる)

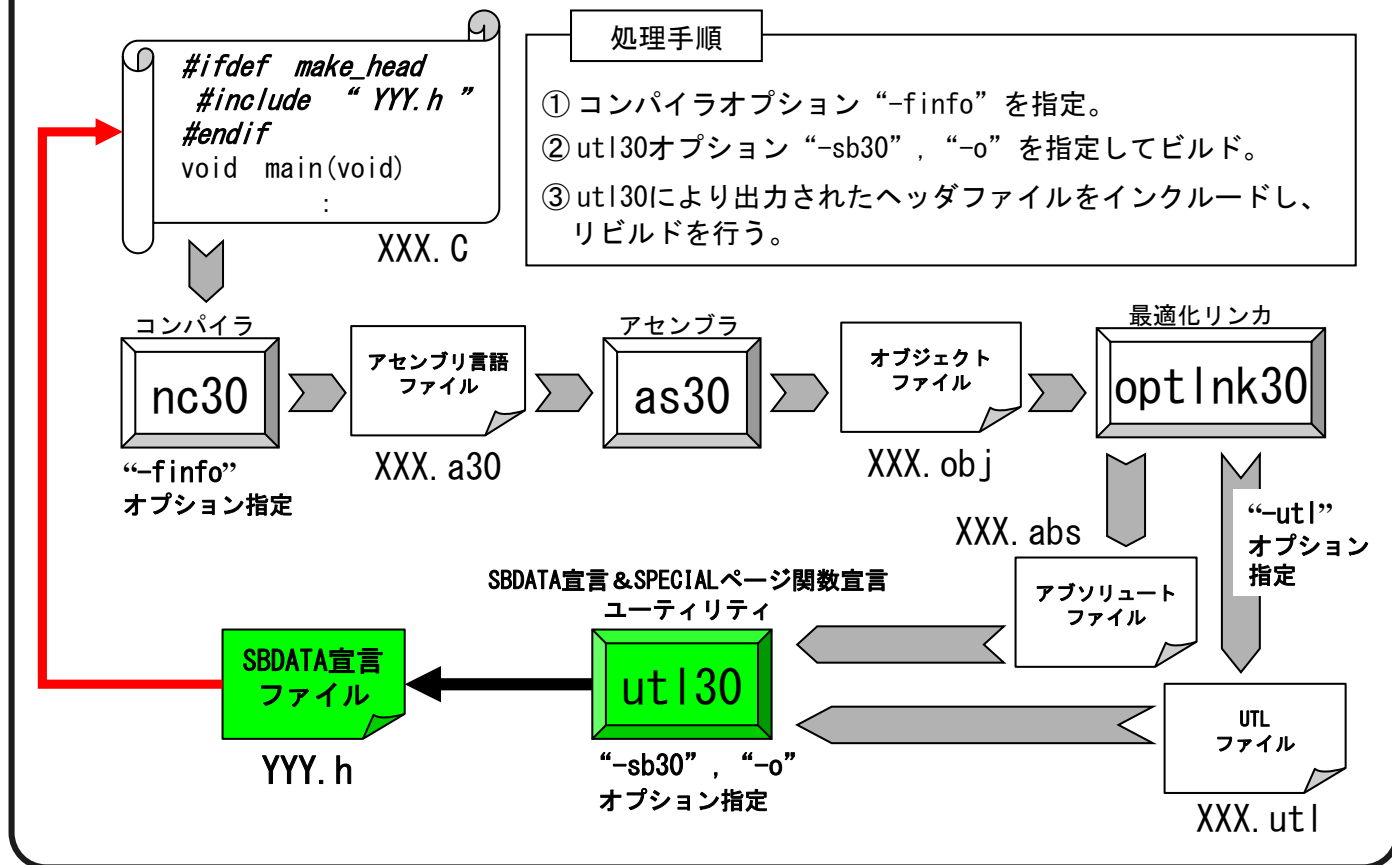
アセンブリ言語関数とリンクする場合

アセンブリ言語関数とリンクする場合、utl30ではアセンブリ言語ファイル中で記述されている“.SBSYM”で指定された変数については、カウントしない。

したがって、C言語ファイル側でSB領域(256バイト)を全て使用している状態でアセンブリ言語ファイル内で“.SBSYM”の宣言がある場合は、utl30実行後の生成された結果に対してSB領域内(SBレジスタで示したアドレスから256バイトの範囲内)に入るよう調整が必要となる。

なおデータの使用頻度が“0”のときは、その変数がプログラム内で使用されていないことを示す。したがってこの場合、変数宣言を削除することによりメモリの使用量を削減できる。

ユーティリティの使用手順



Code : R8Cコース

Date : Rev. 2. 20

Page: 22 of 22

SBDATA宣言 & SPECIALページ関数宣言ユーティリティ (utl30) のオプションを以下に示す。

- | | |
|--------------------|--|
| —sb30 | SBDATA宣言を出力する。 |
| —sp30 | SPECIALページ関数宣言を出力する。(R8Cファミリでは指定できない) |
| —o<ファイル名> | SBDATA宣言、またはSPECIALページ関数宣言の結果を指定した指定したファイルに出力する。 |
| —fover_write(-fow) | —o オプションで指定された出力ファイルに対して、強制的に上書きする。 |
| —all | 全てのグローバル変数に対してSBDATA宣言、またはSPECIALページ関数宣言する。 |
| —Wstdout | エラー及びウォーニングメッセージを標準出力に出力する。 |
| —fsection | #pragma SECTIONで指定された変数および関数に対してもSBDATA宣言、またはSPECIALページ関数宣言する。 |

(起動手順例)

> utl30 -sb30 sample.abs -o sbdata

└─ “sbdata.h”にSBDATA宣言結果を出力



第5章 スタートアッププログラム

- 5.1 役割と構成
 - 5.2 プログラム例
 - 5.3 スタック使用量の算出
-

5.1 役割と構成

スタートアッププログラムの構成と役割

◆ 広義的な意味で以下の処理がスタートアッププログラムとなる

① MCUの動作環境の設定

② C言語で使用するデータ領域の初期化

③ 標準ライブラリの初期化

④ メモリ管理関数使用時の初期化

⑤ main関数の呼び出し

⑥ 終了処理 (部分は必須)

◆ NC30でのスタートアッププログラムの役割

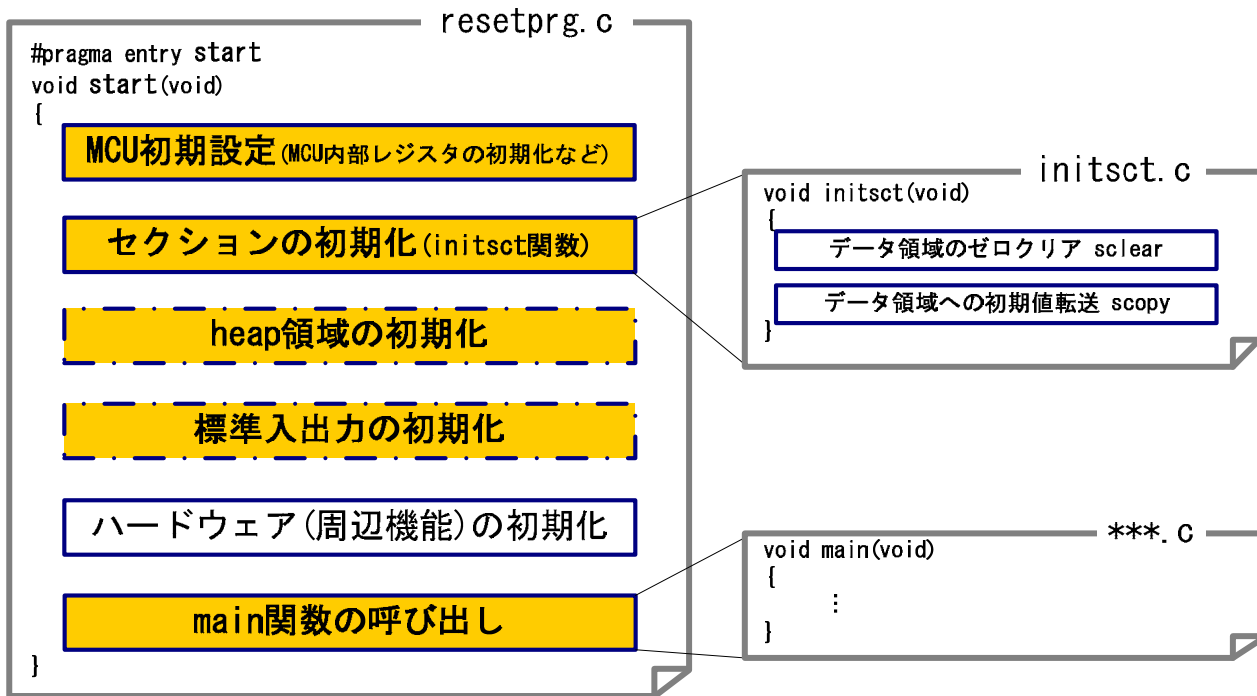
① マイコンの初期化

② スタック領域の確保と静的変数領域の初期化

③ 割り込みベクタの設定

④ main関数の呼び出し

サンプルスタートアッププログラム(C言語用)



Code : R8Cコース

Date : Rev. 2. 20

Page: 2 of 13

📖 #pragma entry 関数名

指定した関数の出入り口で、レジスタの退避／復帰を行わない指定

サンプルスタートアッププログラムの変更

組み込むシステムに合わせて、以下を変更します

ルネサス統合開発環境上で設定

スタック (SP, ISP) サイズ、heapサイズの設定※

各セクションの配置と先頭アドレスの設定

※ プロジェクト生成時のみ。プロジェクト生成後の変更はユーザプログラム (cstartdef.h) で行う

ユーザプログラムで設定

システムクロックやCPU内部レジスタの設定 (resetprg.c)

可変／固定割り込みベクタの設定 (intprg.c、fvector.c)

Code : [R8Cコース](#)

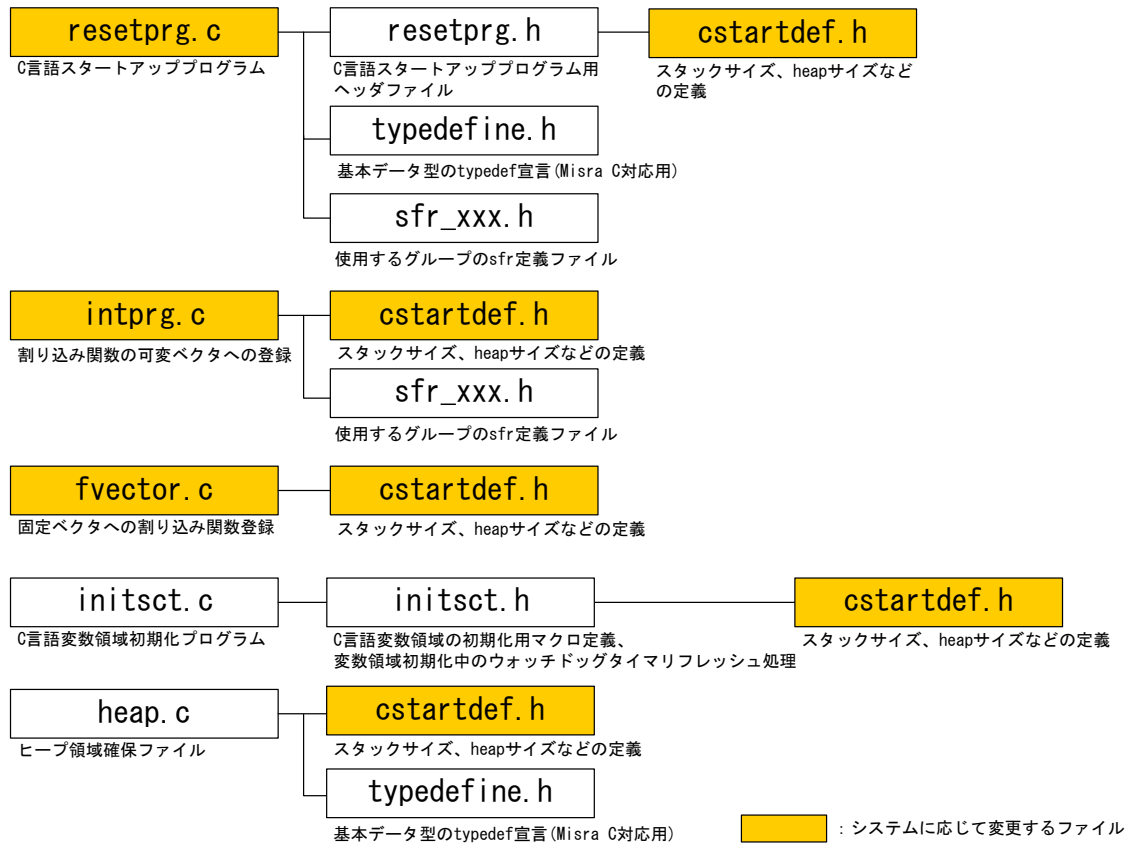
Date : [Rev. 2.20](#)

Page: 3 of 13

※ 以下については、HEWの新規プロジェクト作成時に指定が可能。

- ① 割り込みスタック領域およびユーザスタック領域のサイズ定義
- ② heap領域の使用の有無、およびheap領域のサイズ定義
- ③ 標準入出力関数の使用の有無

C言語スタートアッププログラムのファイル構成



Code : R8Cコース

Date : Rev. 2. 20

Page: 4 of 13

上記のファイルを“High-performance Embedded Workshop”で自動生成します。

5.2 プログラム例

resetprg.c

```

1.  #include "sfr_r825.h"
2.  #include "typedefine.h"
3.  #include "resetprg.h"
4.
5.  DEF_SBREGISTER;
6.  extern _UINT _stack_top,_istack_top
7.  #pragma entry start
8.  void start(void);
9.  extern void initsct(void);
10. extern void _init(void);
11. void exit(int);
12. void main(void);
13.
14. #pragma section program interrupt
15. void start(void)
16. {
17.     _isp_ = &_istack_top; // set interrupt stack pointer
18.     prcr = 0x02; // change protection register
19.     pm0 = 0x00; // set protection register
20.     prcr = 0x00; // change protection register
21.     _flag_ = _F_value; // set flag register
22. #if _STACKSIZE_ != 0
23.     _sp_ = &_stack_top; // set user stack pointer
24. #endif
25.     _sb_ = 0x400; // 400H fixate SBL register's initial value
26.     _intbh_ = 0x00; // set variable vector's address
27.     _asm(" ldc #(topof vector)&0FFFFh, INTBL");
28.
29.     initsct(); // initialize interrupt vector
30. #if _HEAPSIZE_ != 0
31.     heap_init(); // initialize heap
32. #endif
33. #if _STANDARD_IO_ != 0
34.     _init(); // initialize standard I/O
35. #endif
36.     _fb_ = 0; // initialize FB register for debugger
37.     main(); // call main routine
38.
39.     exit(0); // call exit
40. }
41.
42. void exit(int rc)
43. {
44.     while(1); //infinite loop
45. }

```

リセット解除後のプログラム開始位置

割り込みスタックポインタの設定

プロセッサモードレジスタなどのプロテクトされているレジスタの設定 (システムに応じて変更/追加する)

フラグレジスタの初期化

ユーザスタックポインタの設定

SBLレジスタの初期値設定

統合開発環境で設定した可変割り込みベクタの先頭アドレスを割り込みテーブルレジスタに設定

main関数の呼び出し

resetprg.h

```

:
: 省略
:
1.  _UINT _flg_;
2.  _UINT _sb_;
3.  _UINT _fb_;
4.  _UINT *_sp_;
5.  _UINT *_isp_;
6.  _UINT *_intbl_;
7.  _UINT *_intbh_;
8.
9.  #define DEF_BANKSELECT  _asm(" .glb  _BankSelect¥n"¥
10.                          "_BankSelect  .equ
11.  #define DEF_SBREGISTER  _asm(" .glb  SB  ¥n"¥
12.                          "_SB_  .equ  0400H")
13.
14.  #if _STACKSIZE_ != 0
15.  #define _F_value_  0x0080
16.  #else
17.  #define _F_value_  0x0000
18.  #endif
19.
20.  #if _HEAPSIZE_ != 0
21.  //extern  _UBYTE _far * _mbase;
22.  extern  _UBYTE _far * _mnext;
23.  extern  _UDWORD _msize;
24.
25.  :
26.  : 以下省略
27.  :
28.  :

```

start関数内でSBレジスタの値を変更した場合は、この値も変更する。

統合開発環境上でUseUser Stack をチェック (ユーザスタックを使用) した場合は、Uフラグを“1”としてユーザスタックポインタ(USP)と割り込みスタックポインタ(ISP)の両方を使用する

統合開発環境上でUseUser Stack をチェック (ユーザスタックを未使用) しなかった場合は、Uフラグを“0”として割り込みスタックポインタ (ISP)のみを使用する

cstartdef.h

```

1.  #define _STACKSIZE_  0x80  //user stack
2.  #define _ISTACKSIZE_  0x80  //interrupt stack
3.  #define _HEAPSIZE_  0x80  //heap size
4.  #define _STANDARD_IO_  0  // use standard I/O (ex printf)
5.  #define _WATCH_DOG_  0  // When watchdog is made effective after reset,
6.                          // here is set to 1.

```

プロジェクト生成後にスタックサイズを変更したい場合は、この値を変更する

initsect.c

```
1.  #include "initsect.h"
2.  void initsect(void);
3.
4.  void initsect(void)
5.  {
6.      sclear("bss_SE", "data", "align");
7.      sclear("bss_SO", "data", "noalign");
8.      sclear("bss_NE", "data", "align");
9.      sclear("bss_NO", "data", "noalign");
10. #ifndef _NEAR_
11.     sclear_f("bss_FE", "data", "align");
12.     sclear_f("bss_FO", "data", "noalign");
13. #endif
14.     // add new sections
15.     // bss_clear("new section name");
16.
17.     scopy("data_SE", "data", "align");
18.     scopy("data_SO", "data", "noalign");
19.     scopy("data_NE", "data", "align");
20.     scopy("data_NO", "data", "noalign");
21. #ifndef _NEAR_
22.     scopy_f("data_FE", "data", "align");
23.     scopy_f("data_FO", "data", "noalign");
24. #endif
25. }
```

near RAM領域 (bssセクション) の初期化

far RAM領域 (bssセクション) の初期化
(R8C/Tinyでは未使用)

near RAM領域 (dataセクション) の初期化

far RAM領域 (dataセクション) の初期化
(R8C/Tinyでは未使用)


```
#include "cstartdef.h"
#pragma section program interrupt
#if _WATCH_DOG_ != 1
#define scopy(X,Y,Z)    _asm(".initsect \"X\",\"Y\",\"Z\"¥n¥n¥n\n".initsect "X"I,rom"Y",noalign¥n"¥n¥n\n".mov.w      #(topof "X")&0ffffH,A0¥n"¥n¥n\n".mov.b      #00H,R1H¥n"¥n¥n\n".mov.w      #(topof "X")&0ffffH,A1¥n"¥n¥n\n".mov.w      #sizeof      "X",R3¥n"¥n¥n\n".smovf.b");
#define sclear(X,Y,Z)   _asm(".initsect \"X\",\"Y\",\"Z\"¥n¥n¥n\n".mov.b      #00H,R0L¥n"¥n¥n\n".mov.w      #(topof      "X"),A1¥n"¥n¥n\n".mov.w      #sizeof      "X",R3¥n"¥n¥n\n".sstr.b");
#else
#define scopy(X,Y,Z)     _asm(".initsect \"X\",\"Y\",\"Z\"¥n¥n¥n\n".initsect "X"I,rom"Y",noalign¥n"¥n¥n\n".scopy      "X"I,"X""");
#define sclear(X,Y,Z)    _asm(".initsect \"X\",\"Y\",\"Z\"¥n¥n¥n\n".N_BZERO    "X""");
#endif
:
省略
:

#if _WATCH_DOG_ == 1
#pragma ASM
scopy .macro            FROM_TO_
    mov.b              #00,0DH
    mov.b              #0ffH,0DH
    mov.w              #0000H,R0
    .local M1
M1:
        cmp.w          #sizeof TO_,R0
        mov.b          #0ffH,0DH
:
省略
:
M3:
    mov.w              #(topof      SECT_)&0ffffH,
    add.w              R0,A0
    mov.b              #00H,[A0]
    add.w              #1H,R0
    jmp                M1
M2:
    .endm
#pragma ENDASM
#endif
```

変数領域の初期化処理実行中に
ウォッチドッグタイマをリフレッシュする
処理のアセンブラマクロ定義(リセット
解除後にウォッチドッグタイマを動作
させる設定をした場合にのみ展開)

intprg.c

```

1.      / When you want to use BANK1 registers
2.      // please define interrupt using /B switch as follows.
3.      //
4.      // #pragma interrupt/B xxxx
5.      //
6.      // BRK                (software int 0)
7.      #pragma interrupt      _brk(vect=0)
8.      void _brk(void){ }
9.      // vector 1 reserved
10.     // vector 2 reserved
11.     // vector 3 reserved
12.     // vector 4 reserved
13.     // vector 5 reserved
14.     // vector 6 reserved
15.     // vector 7 reserved
16.     // timer RD (channel 0) (software int 8)
17.     #pragma interrupt      _timer_rd0(vect=8)
18.     void _timer_rd0(void){ }
19.
20.     // timer RD (channel 1) (software int 9)
21.     #pragma interrupt      _timer_rd1(vect=9)
22.     void _timer_rd1(void){ }
23.
24.     // timer RE                (software int 10)
25.     #pragma interrupt      _timer_re(vect=10)
26.     void _timer_re(void){ }
27.     // vector 11 reserved
28.     // vector 12 reserved
29.     // input_key                (software int 13)
30.     #pragma interrupt      _input_key(vect=13)
31.     void _input_key(void){ }
32.
33.     // A-D converter            (software int 14)
34.     #pragma interrupt      _ad_converter(vect=14)
35.     void _ad_converter(void){ }
36.
37.     // SSU IIC                  (software int 15)
38.     #pragma interrupt      _ssu(vect=15)
39.     void _ssu(void){ }
40.     // vector 16 reserved
41.
42.     #if defined (_STANDARD_IO_) && (defined(_FOUSB_) || defined(_E8_))
43.     // uart0 can't be used
44.     #else
45.     // uart0 trance              (software int 17)
46.     :
47.     以下省略
48.     :

```

割り込み関数の指定

エントリ関数

fvector.c

```

1.  #include "cstartdef.h"
2.  #pragma sectaddress   fvector,ROMDATA 0xffdc
3.

```

```

4.  //////////////////////////////////////
5.

```

```

6.  #pragma interrupt/v _dummy_int
7.  #pragma interrupt/v _dummy_int
8.  #pragma interrupt/v _dummy_int
9.  #pragma interrupt/v _dummy_int
10. #pragma interrupt/v _dummy_int
11. #pragma interrupt/v _dummy_int
12. #pragma interrupt/v _dummy_int
13. #pragma interrupt/v _dummy_int
14. #pragma interrupt/v start

```

_dummy_int部分を割り込み関数名に変更する

[記述例] #pragma interrupt /v *wdt_int*

```

//address_match
//single_step
//wdt
//reserved

```

```

15.
16.  #if _WATCH_DOG_ != 0
17.  _asm(".ofsreg 0FEH");
18.  #else
19.  _asm(".ofsreg 0FFH");
20.  #endif

```

「オプション機能選択レジスタ(0FFFFh番地)」の値設定部。

※リセット解除後にウォッチドッグタイマをスタートさせる設定をする場合は、「オプション機能選択レジスタ」のbit0を'0'として設定する。
(マクロ名"_WATCHDOG_"は、cstartdef.hでデフォルト'0'に定義されている)

```

21.
22.  _asm(".id      \"\"#\"#FFFFFFFFFFFFFF#\"");
23.

```

```

24.  #pragma interrupt _dummy_int()
25.  void _dummy_int(void){}

```

ダミー割り込み

フラッシュメモリ書き換えの「IDコードチェック機能」使用時で、「.id」命令でIDコードを設定する場合は、このIDコードを変更する。
デフォルトは、FFFFFFFFFFFFFFh(コード未設定)で設定される。

※ IDコード(文字列/即値) 格納番地 =

0FFDFh, 0FFE3h, 0FFEBh, 0FFEfh, 0FFF3h, 0FFF7h, 0FFFBh

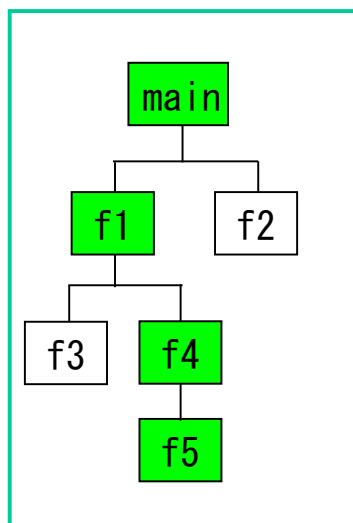
5.3 スタック使用量の算出

スタックを使用するもの

- ① 局所変数領域（自動変数）
- ② 演算などで使用するテンポラリ領域
- ③ 関数への引数
- ④ 関数呼び出し時のリターンアドレス
- ⑤ 旧フレームポインタ（FBレジスタ）

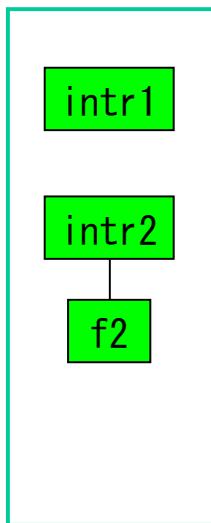
最大スタック使用量の算出方法

<file1.c>



計算結果 A

<file2.c>

計算結果 B
または
計算結果 C

算出手順

<STEP1>

- ① 関数毎の最大スタック使用量を求める
- ② 関数の呼び出し関連を考慮し、スタック消費量が最大となるパスを求め、関数毎の最大スタック使用量の総和を求める → 算出結果 A

<STEP2>

- ③ 割り込み関数毎の最大スタック使用量を求める
- ④ 最もスタックを消費している割り込み関数を割り出す → 算出結果 B
- ⑤ 多重割り込みを許可している場合は、割り込みがネスティングした場合の最悪値を求める → 算出結果 C
- ⑥ 算出結果 A + B (または C) がシステム全体での最大スタック使用量となる

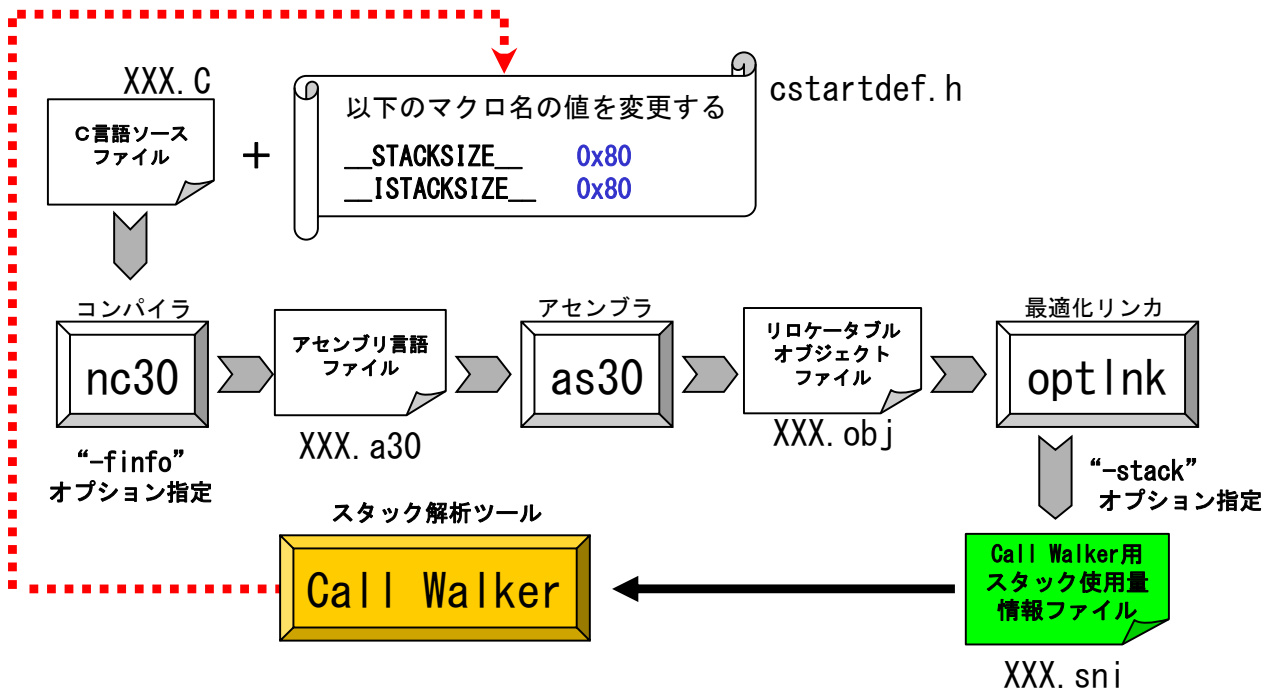
Code : R8Cコース

Date : Rev. 2.20

Page:12 of 13

NC30では、Renesas Call Walker により、自動的にシステム全体の最大スタック使用量を算出することができる。

Call Walkerによるスタック使用量の解析



Code : R8Cコース

Date : Rev. 2.20

Page:13 of 13

Call Walkerは、High-performance Embedded Workshopから起動します。起動後、各関数のスタック使用量を算出し、結果をウィンドウに表示します。

【処理手順】

- ① “-finfo”オプションを指定してビルド。
- ② High-performance Embedded Workshop から Call Walkerを起動し、以下のスタック使用量の自動算出と表示を行う。
 - a. スタートアップルーチン “start()” から下の階層で使用する最大スタック使用量
 - b. 割り込み関数毎の最大スタック使用量
- ③ ②のaの値を『ユーザースタックの使用量』として、cstartdef.hファイル内の識別子 “__STACKSIZE__” の値を変更する。
- ④ 多重割り込みを禁止している場合は、②のb中の最も大きいサイズを『割り込みスタックの使用量』として、cstartdef.hファイル内の識別子 “__ISTACKSIZE__” の値を変更する。
多重割り込みを許可している場合は、割り込みネスティング時の最大値を机上計算し、cstartdef.hファイル内の識別子 “__ISTACKSIZE__” の値を変更する。
- ⑤ スタック領域のサイズ変更後にリビルドを行う。

※ Call Walker では、以下に示す関数については算出できないため、別途 算出して結果を加算してください。

- ・再帰呼び出しされている関数
- ・間接呼び出しにより呼び出されている関数
- ・アセンブリ言語関数
- ・asm関数から呼び出されている関数

第6章 *R8Cファミリの割り込み制御*

- 6.1 割り込みの動作と使用方法
 - 6.2 $\overline{\text{INT}}$ 割り込み
-

6.1 割り込みの動作と使用方法

R8C/25グループの割り込み要因

ハードウェア割り込み	
周辺I/O	
キー入力	
A/D変換	
シリアルインタフェース	UART0送信
	UART0受信
	UART1送信
	UART1受信
	チップセレクト付き クロック同期シリアルI/O ／I2Cバスインタフェース
タイマ	タイマRA
	タイマRB
	タイマRD(チャンネル0)
	タイマRD(チャンネル1)
	タイマRE
外部端子	INT0～INT3
特殊	
リセット	
ウォッチドッグタイマ	
発振停止検出	
電圧監視1、電圧監視2	
シングルステップ(注1)	
アドレスブレイク(注1)	
アドレス一致	

ソフトウェア割り込み
BRK命令
INT命令
INT0命令(オーバーフロー/0除算)
UND命令(未定義命令)

- ハードウェア割り込み
 - 周辺I/O: マスカブル
 - 特殊: ノンマスカブル
- ソフトウェア割り込み: ノンマスカブル

注1: 開発サポートツール専用割り込みのため、使用禁止。

Code: R8Cコース

Date: Rev. 2.20

Page: 1 of 17

ノンマスカブル割り込み

割り込み許可フラグ(Iフラグ)による割り込みの許可および
割り込み優先レベルによる割り込み優先順位の変更不可

マスカブル割り込み

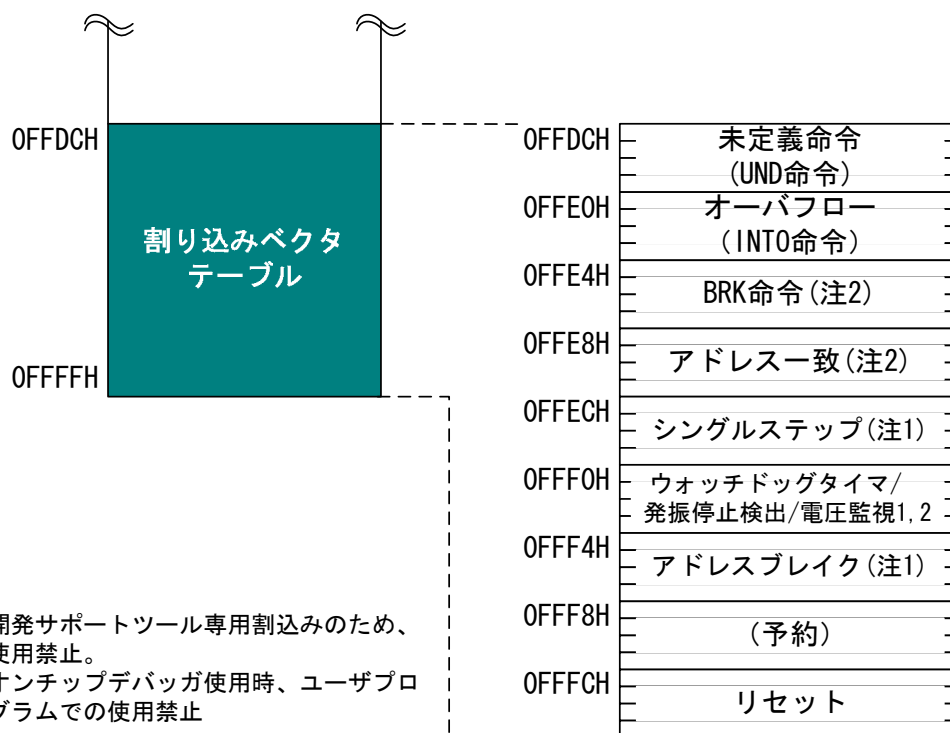
割り込み許可フラグ(Iフラグ)による割り込みの許可および
割り込み優先レベルによる割り込み優先順位の変更可

固定ベクタと可変ベクタ

特殊割り込みとソフトウェア割り込みは「固定ベクタ」に、周
辺I/O割り込みは可変ベクタにそれぞれ割り込み要求受け付け
後の飛び先アドレスを格納する。

固定ベクタテーブル

1ベクタは4バイトで構成され、各ベクタの最上バイトは「IDコード格納領域」となっている。

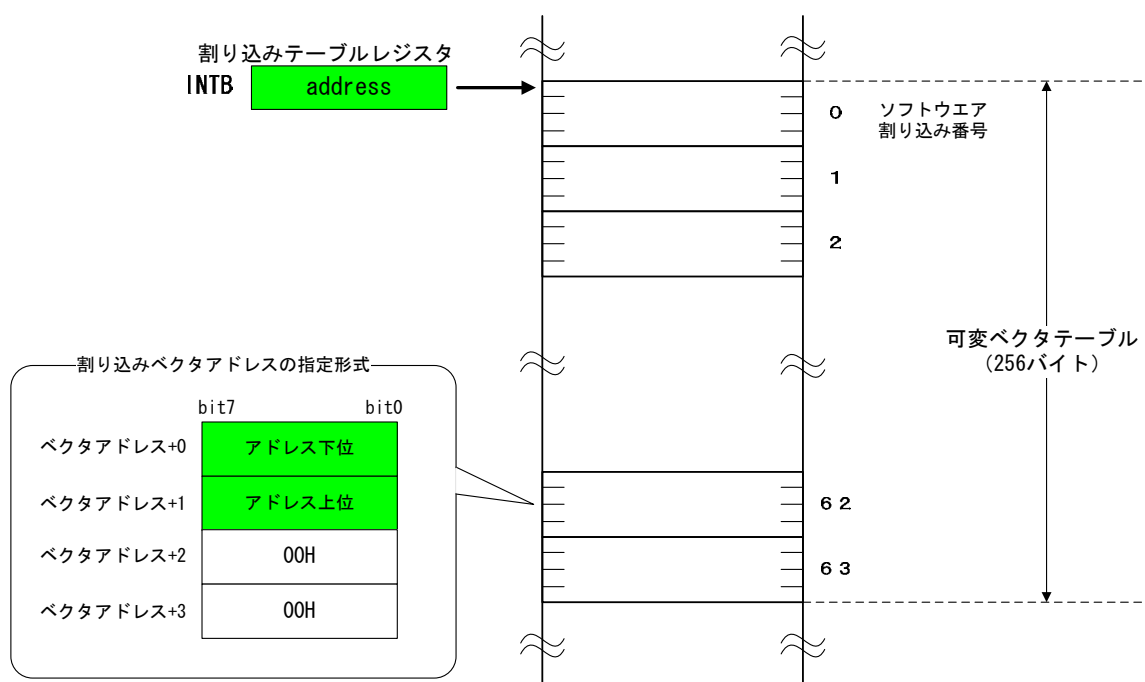


注1. 開発サポートツール専用割り込みのため、
使用禁止。

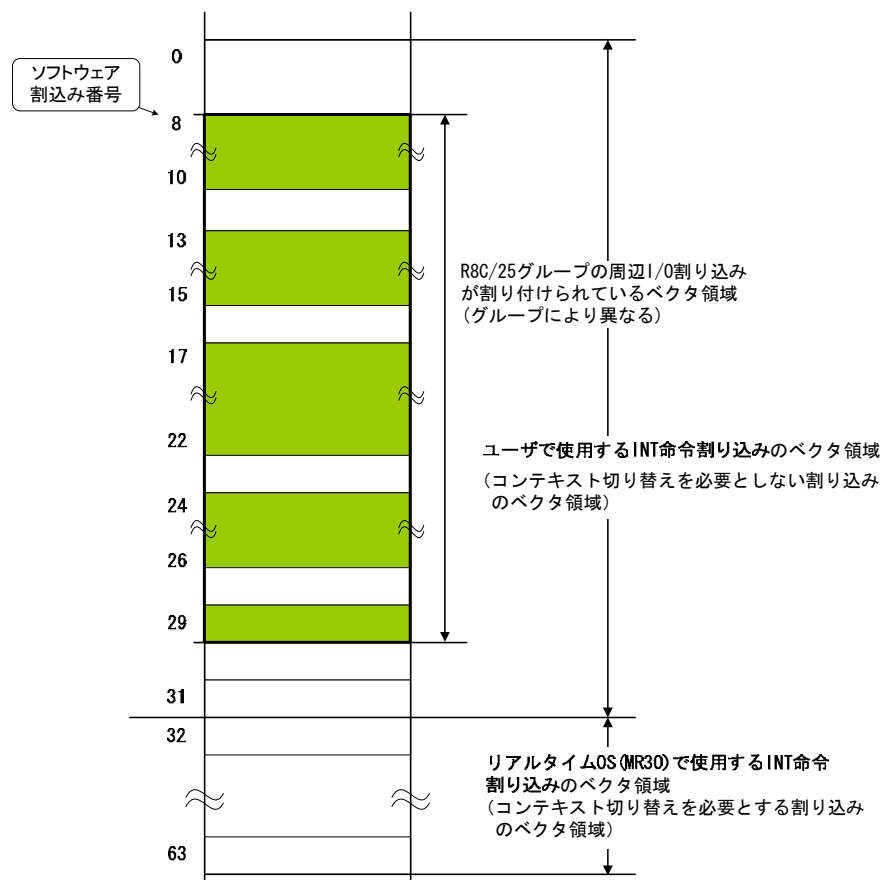
注2. オンチップデバッガ使用時、ユーザプロ
グラムでの使用禁止

可変ベクタテーブル

可変ベクタテーブルは、割り込みテーブルレジスタ（INTB）の内容で示された値を先頭アドレスとする256バイトの割り込みベクタテーブル。ベクタテーブルは全メモリ空間（SFR領域を除く）の任意の空間に配置が可能。各ベクタは4バイトで構成され、0～63までのソフトウェア割り込み番号が割り付けられている。



ソフトウェア割り込み番号の割り当て



Code : R8Cコース

Date : Rev. 2.20

Page: 4 of 17

割り込み要因とベクタ番地 (R8C/25グループ)

優先 レベル	割り込み 番号	割り込み要因	ベクトル番地 ※1	Iフラグに よるマスク
1	-	リセット	0FFFCH~0FFFFH (固定)	不可
2	-	アドレスブレイク ※2	0FFF4H~0FFF7H (固定)	不可
3	-	ウォッチドッグタイマ、 発振停止検出、電圧監視1, 2	0FFF0H~0FFF3H (固定)	不可
5	-	シングルステップ ※2	0FFECH~0FFE FH (固定)	不可
6	-	アドレス一致	0FFE8H~0FFEBH (固定)	不可
4	0	BRK命令	0~+3	不可
	1~7	(予約)	+4~+28	-
	8	タイマRD(チャンネル0)	+32~+35	可
	9	タイマRD(チャンネル1)	+36~+39	可
	10	タイマRE	+40~+43	可
	11~12	(予約)	+44~+51	-
	13	キー入力	+52~+55	可
	14	A/D変換	+56~+59	可
	15	チップセレクト付きクロック 同期形シリアルI/O ／I2Cバスインタフェース	+60~+63	可
	16	(予約)	+64~+67	可
	17	UART0送信	+68~+71	可
	18	UART0受信	+72~+75	可
	19	UART1送信	+76~+79	可
	20	UART1受信	+80~+83	可
	21	TNT2	+84~+87	可
	22	タイマRA	+88~+91	可
	23	(予約)	+92~+95	-
	24	タイマRB	+96~+99	可
	25	TNTT	+100~+103	可
	26	TNT3	+104~+107	可
	27	(予約)	+108~+111	-
	28	(予約)	+112~+115	-
	29	TNT0	+116~+119	可
	30	(予約)	+120~+123	-
	31	(予約)	+124~+127	-
ソフト ウェア 割り込み	32 ~ 63	INT命令	+128~+131 ~ +252~+255	不可
	-	BRK命令 ※3	0FFE4H~0FFE7H (固定)	不可
	-	INT0命令 (オーバフロー/0除算)	0FFE0H~0FFE3H (固定)	不可
	-	UND命令 (未定義命令)	0FFDCH~0FFDFH (固定)	不可

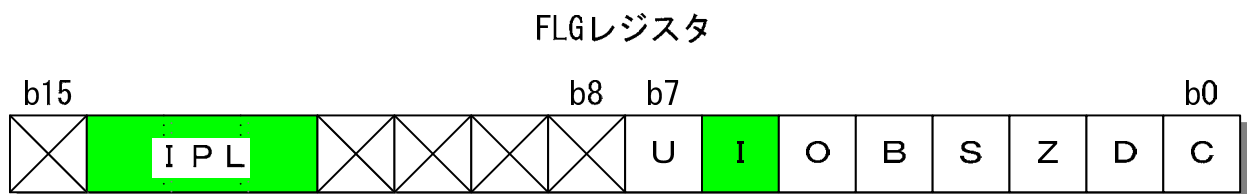
※1：INTBレジスタが示すアドレスからの相対アドレス値。

※2：開発サポートツール専用割り込みのためユーザは使用禁止。

※3：0FFE7H番地の内容が“FFH”の場合にのみ可変ベクタとなる。

割り込み受け付け条件

- ① 割り込み要求ビット(割り込み制御レジスタのbit3) = 1 (要求あり)
- ② $IPL < \text{発生した割り込みの割り込み優先レベル}$
(割り込み制御レジスタのbit0~bit2)
- ③ 割り込み許可フラグ(Iフラグ) = 1 (割り込み許可状態)

Code : R8CコースDate : Rev. 2. 20Page: 6 of 17

②、③はユーザがプログラムで設定し、①はハードウェアによってセットされる。

割り込み許可フラグ(1フラグ)

マスカブル割り込みの禁止／許可を制御する。

FSET 1 ; 1で割り込み許可

FCLR 1 ; 1=0で割り込み禁止

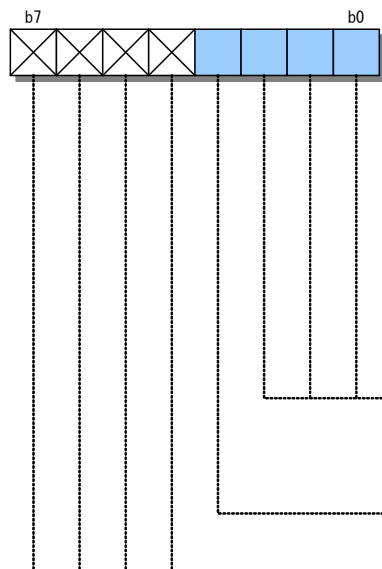
プロセッサ割り込み優先レベル (IPL)

レベル0～レベル7でプロセッサ割り込み優先レベルを指定。

割り込み要求があった割り込みの優先レベルが、IPLより大きい場合にその割り込みを受け付ける。

割り込み制御レジスタの構成

(1) INT_i以外の割り込み制御レジスタ (注2) (i = 0~3)

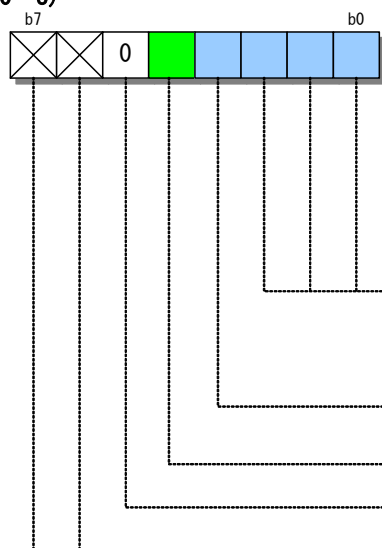


シンボル	アドレス	リセット時
TREIC	004AH番地	XXXXX000B
KUPIC	004DH番地	XXXXX000B
ADIC	004EH番地	XXXXX000B
SOTIC	0051H番地	XXXXX000B
SORIC	0052H番地	XXXXX000B
SITIC	0053H番地	XXXXX000B
SIRIC	0054H番地	XXXXX000B
TRAIC	0056H番地	XXXXX000B
TRBIC	0058H番地	XXXXX000B
TRDOIC	0048H番地	XXXXX000B
TRD1IC	0049H番地	XXXXX000B
SSUIC/IICIC (注3)	004FH番地	XXXXX000B

ビット名	機能	R	W
割り込み優先レベル選択ビット /ILVLO, 1, 2	000 : レベル0 (割り込み禁止) 001 : レベル1 010 : レベル2 011 : レベル3 100 : レベル4 101 : レベル5 110 : レベル6 111 : レベル7	○	○
割り込み要求ビット (注4) /IR	0 : 割り込み要求なし 1 : 割り込み要求あり	○	○ (注1)
何も配置されていない。書き込む場合、“0”を書き込む。読み出した場合、その値は不定。		—	—

- 注1. “0”だけ書き込み可 (“1”を書き込まないこと)。
 注2. 割り込み制御レジスタの変更は、そのレジスタに対応する割り込み要求が発生しない箇所で行うこと。
 注3. PMRレジスタのIICSELビットで選択。
 注4. TRDOIC, TRD1IC, SSUIC/IICICレジスタの場合は、読み出しのみで、書き込みは不可。

(2) INT_i割り込み制御レジスタ (注2) (i = 0~3)



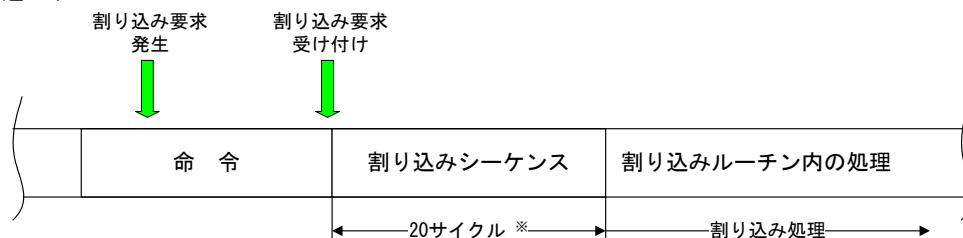
シンボル	アドレス	リセット時
INT0IC	0055H番地	XX00X000B
INT1IC	0059H番地	XX00X000B
INT2IC	005AH番地	XX00X000B
INT3IC	005DH番地	XX00X000B

ビット名	機能	R	W
割り込み優先レベル選択ビット /ILVLO, 1, 2	000 : レベル0 (割り込み禁止) 001 : レベル1 010 : レベル2 011 : レベル3 100 : レベル4 101 : レベル5 110 : レベル6 111 : レベル7	○	○
割り込み要求ビット /IR	0 : 割り込み要求なし 1 : 割り込み要求あり	○	○ (注1)
極性切り替えビット /POL (注4)	0 : 立ち下がりエッジを選択 1 : 立ち上がりエッジを選択 (注3)	○	○
予約ビット	必ず“0”を設定すること	○	○
何も配置されていない。書き込む場合、“0”を書き込む。読み出した場合、その値は不定。		—	—

- 注1. “0”だけ書き込み可 (“1”を書き込まないこと)。
 注2. 割り込み制御レジスタの変更は、そのレジスタに対応する割り込み要求が発生しない箇所で行うこと。
 注3. 外部入力許可レジスタのTINT0入力極性選択ビットが“1” (両エッジ) の場合、極性切り替えビット (POLビット) を“0” (立ち下がりエッジを選択) にすること。
 注4. 極性切り替えビット (POLビット) を変更すると割り込み要求ビットが“1”になる場合があるため、変更後に割り込み要求ビットを“0”にする。

割り込み受け付けタイミング

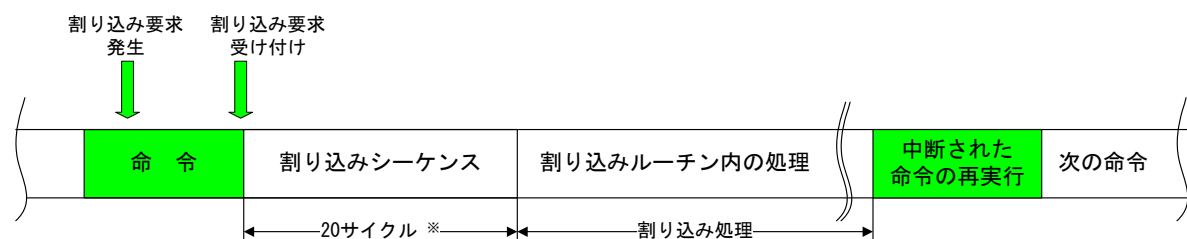
< 通 常 >



< 例 外 >

以下の命令を実行中に割り込みが発生した場合、命令実行途中で割り込み要求が受け付けられる。

- ① ストリング転送命令 (SMOVF, SMOVB, SSTR)
- ② 積和演算命令 (RMPIA)



※ アドレス一致割り込み、シングルステップ割り込みは21サイクル

ハードウェア割り込み優先レベル

優先順位

高



低



INT3
タイマRB
タイマRA
INT0
INT1
UART1受信
UART0受信
A/D変換
タイマRE
タイマRD0
INT2
UART1送信
UART0送信
SSU/I ² Cバス
キー入力
タイマRD1

優先順位

高



低



リセット
アドレスブレイク
ウォッチドッグタイマ 発振停止検出 電圧監視1, 電圧監視2
周辺I/O
シングルステップ
アドレス一致

割り込みシーケンス

- ① 0000H番地を読み込み、割り込み情報(割り込み番号など)を取得する。その後、受け付けた割り込みの割り込み要求ビットを“0”にする。
- ② FLGレジスタの内容をCPU内部の一時レジスタに退避する。
- ③ FLGレジスタの U, I, Dフラグをクリアする。
(ただし Uフラグについては、ソフトウェア割り込み番号32～63のINT命令を実行した場合は変化しない。)
- ②の動作により、
 - ・スタックポインタは強制的に割り込みスタックポインタ (ISP) に切り替わる。
(ただし、ソフトウェア割り込み番号32～63のINT命令を実行した場合は、割り込み発生時のスタックポインタ (ISPまたはUSP) を使用する。)
 - ・多重割り込みは禁止となる。
 - ・シングルステップ割り込みは禁止となる。
- ④ CPU内部の一時レジスタ (FLGを退避しておいたレジスタ) とプログラムカウンタの内容をスタック領域(割り込みスタック/ユーザスタック)に退避する。
- ⑤ プロセッサ割り込み優先レベル(IPL)に受け付けた割り込みの割り込み優先レベルを設定する。
- ⑥ 割り込みベクタに設定された割り込みルーチンの先頭アドレスをプログラムカウンタに転送する。

Code : R8Cコース

Date : Rev. 2.20

Page:10 of 17

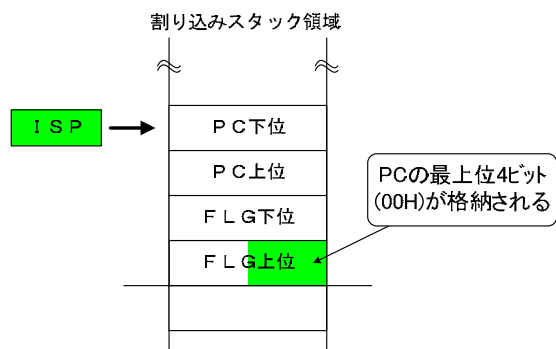
割り込み受付時のIPLの変化

割り込み要因	設定されるIPLの値
ウォッチドッグタイマ、発振停止検出、電圧監視1、電圧監視2、アドレスブレイク	7
リセット	0
周辺 I / O	受け付けた割り込みの優先レベル
ソフトウェア割り込み、アドレス一致、シングルステップ	変化しない

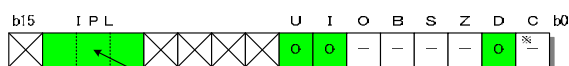
割り込み要求受け付け後のスタックとフラグレジスタの状態

周辺I/O割り込み発生時 または
ソフトウェア割り込み番号 0～31のINT命令実行時

《割り込み要求受け付け後のスタックの状態》



《割り込み要求受け付け後のFLGの状態》

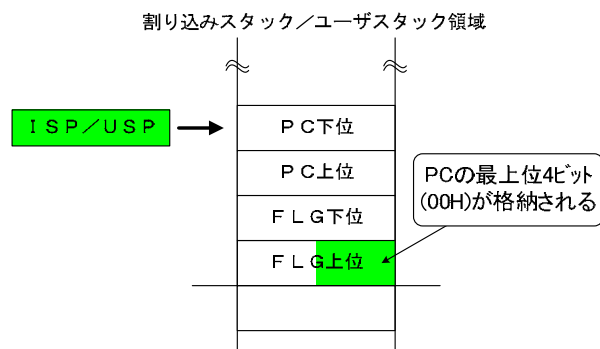


受け付けた割り込みの
優先レベルが格納される

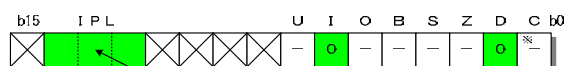
“—” は、変化なし(割り込み受け付け前の状態と同じ)

ソフトウェア割り込み番号 32～63のINT命令実行時

《割り込み要求受け付け後のスタックの状態》



《割り込み要求受け付け後のFLGの状態》



受け付けた割り込みの
優先レベルが格納される

“—” は、変化なし(割り込み受け付け前の状態と同じ)

Code : R8Cコース

Date : Rev. 2.20

Page:11 of 17

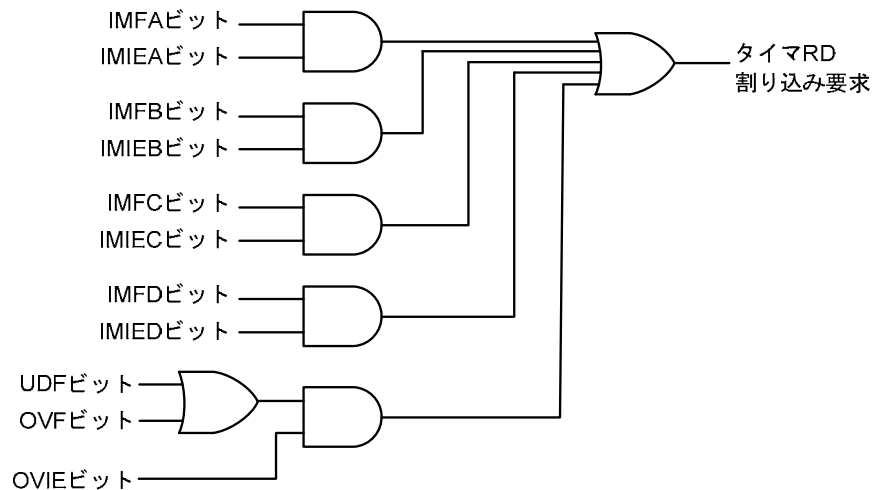
割り込みからの復帰

“REIT”命令を実行することで、スタックに退避していた戻り先番地とFLGレジスタの内容が、それぞれPC、FLGレジスタに復帰し、割り込み要求受け付け前に実行していたプログラムに戻る。

タイマRD、チップセレクト付クロック同期形シリアルI/O、 I²Cバスインタフェース割り込みの割り込み要求

タイマRD(チャンネル0, チャンネル1)、チップセレクト付クロック同期形シリアルI/O、およびI²Cバスインタフェースの割り込み要求については、それぞれ複数の割り込み要求要因を持っており、それらの論理和が割り込み要求となる。

【タイマRD チャンネル*i*(*i* = 0~1)の例】



IMFA : インพุットキャプチャ/コンペアー致フラグA
IMFB : インพุットキャプチャ/コンペアー致フラグB
IMFC : インพุットキャプチャ/コンペアー致フラグC
IMFD : インพุットキャプチャ/コンペアー致フラグD
OVF : オーバフローフラグ
UDF : アンダフローフラグ

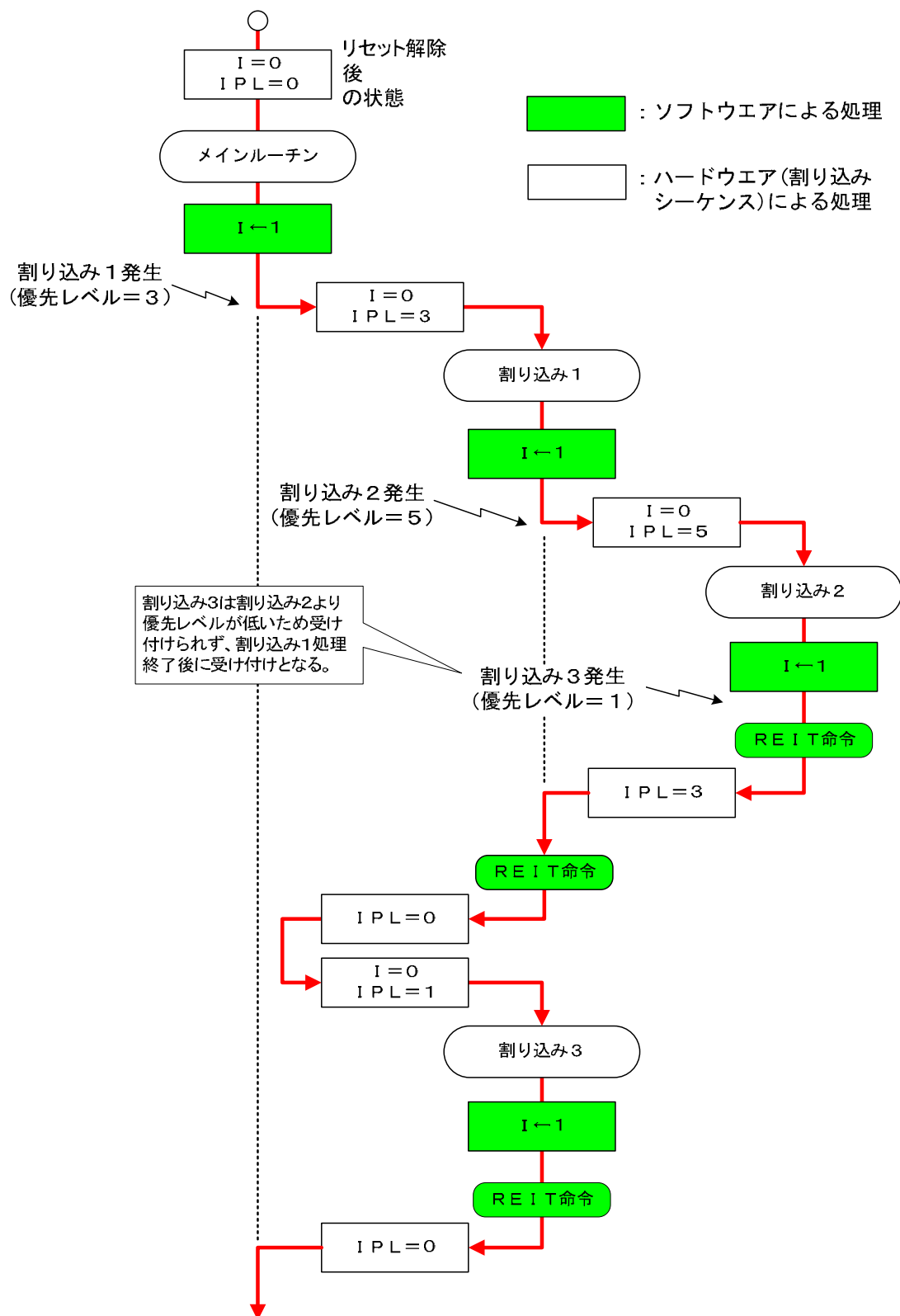
IMIEA : インพุットキャプチャ/コンペアー致割り込み許可ビットA
IMIEB : インพุットキャプチャ/コンペアー致割り込み許可ビットB
IMIEC : インพุットキャプチャ/コンペアー致割り込み許可ビットC
IMIED : インพุットキャプチャ/コンペアー致割り込み許可ビットD
OVIE : オーバフロー/アンダフロー割り込み許可ビット

Code : R8Cコース

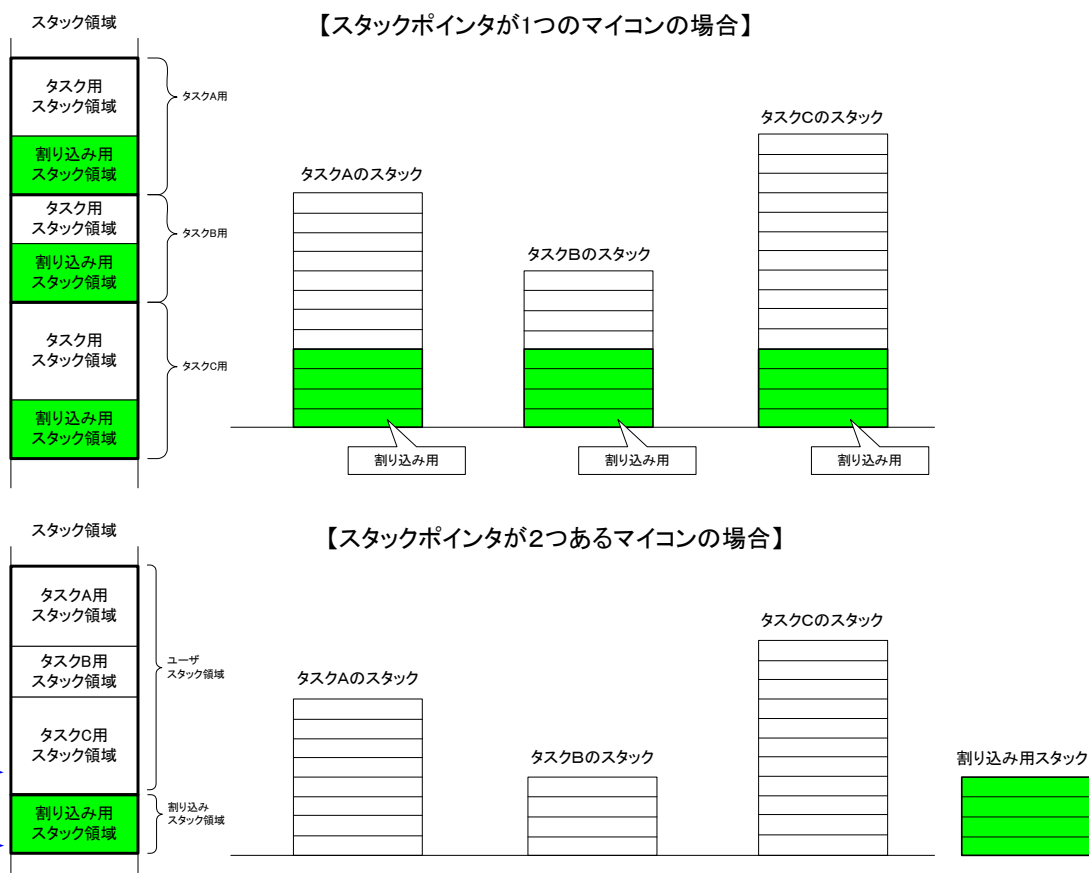
Date : Rev. 2. 20

Page:12 of 17

多重割り込み受け付け動作



スタックポインタ2本の有効性



Code : R8Cコース

Date : Rev. 2.20

Page: 14 of 17

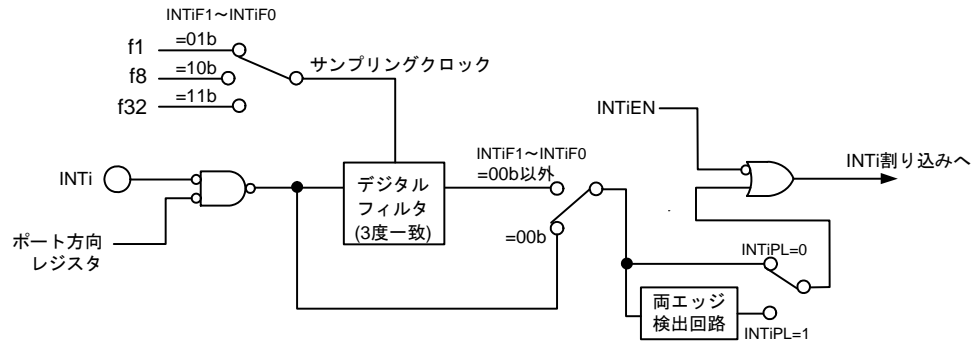
スタックポインタが1つしかないマイコンの場合、割り込みハンドラで消費するスタックサイズを各タスクごとに確保しておく必要がある。

これに対しスタックポインタが2本あるR8C/Tinyの場合は、割り込みが発生するとスタックポインタが自動的に割り込みスタックに切り替わるため、各タスクで使用するスタックをユーザスタック領域としておくことで、タスク毎にハンドラが消費する分を確保しておく必要がなくなり、システム全体のRAM使用量を削減できる。

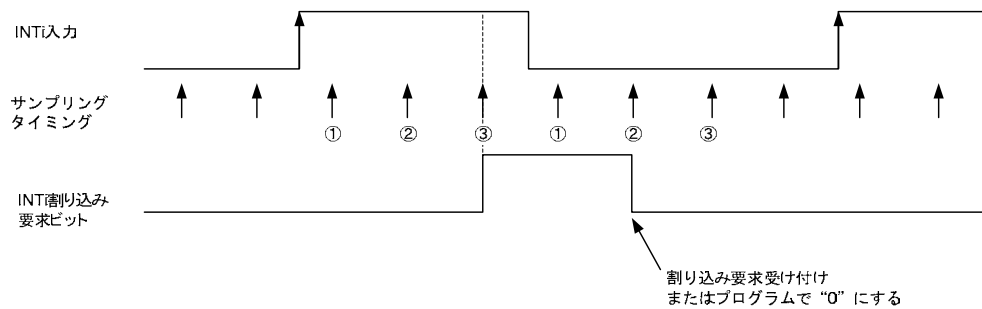
6.2 INT割り込み

INT_i割り込み

INT_i端子から入力されるエッジにより発生する割り込み。INT_i端子からの信号を直接入力するか、デジタルフィルタを介して入力するかを選択できる



INTiF0、INTiF1 : TMT入力フィルタ選択レジスタ (INTFレジスタ) のビット
INTiEN、INTiPL : 外部入力許可レジスタ (INTENレジスタ) のビット



Code : R8Cコース

Date : Rev. 2. 20

Page: 15 of 17

INT_i割り込みはINT_i端子に入力される信号のエッジにより発生する割り込み。R8C/25では、INT0～INT3割り込みの4つがある。

INTi割り込み 関連レジスタ(1)

外部入力許可レジスタ

b7 b6 b5 b4 b3 b2 b1 b0							
シンボル INTEN		アドレス 00F9h番地		リセット後の値 0000 0000b			
ビット シンボル	ビット名			機能		RW	
INT0EN	INT0入力許可ビット			0 : 禁止 1 : 許可		RW	
INT0PL	INT0入力極性選択ビット (注1、2)			0 : 片エッジ 1 : 両エッジ		RW	
INT1EN	INT1入力許可ビット			0 : 禁止 1 : 許可		RW	
INT1PL	INT1入力極性選択ビット (注1、2)			0 : 片エッジ 1 : 両エッジ		RW	
INT2EN	INT2入力許可ビット			0 : 禁止 1 : 許可		RW	
INT2PL	INT2入力極性選択ビット (注1、2)			0 : 片エッジ 1 : 両エッジ		RW	
INT3EN	INT3入力許可ビット			0 : 禁止 1 : 許可		RW	
INT3PL	INT3入力極性選択ビット (注1、2)			0 : 片エッジ 1 : 両エッジ		RW	

注1. INTiPLビットを“1”（両エッジ）にする場合、INTiICレジスタのPOLビットを“0”（立ち下がり

注2. INTiPLビットを変更すると、INTiICレジスタのIRビットが“1”（割り込み要求あり）になることがあります。

Code : R8Cコース

Date : Rev. 2. 20

Page: 16 of 17

INTi割り込み

INTi端子を使用する場合は、該当するポートの方向を必ず入力方向にすること。

INTi割り込み 関連レジスタ(2)

INT入力フィルタレジスタ

b7 b6 b5 b4 b3 b2 b1 b0							
シンボル INTF		アドレス 00FAh番地		リセット後の値 00000000b			
ビット シンボル	ビット名		機能		RW		
INT0F0	INT0入力フィルタ選択ビット		b1 b0 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング		RW		
INT0F1					RW		
INT1F0	INT1入力フィルタ選択ビット		b3 b2 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング		RW		
INT1F1					RW		
INT2F0	INT2入力フィルタ選択ビット		b5 b4 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング		RW		
INT2F1					RW		
INT3F0	INT3入力フィルタ選択ビット		b7 b6 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング		RW		
INT3F1					RW		

第7章 割り込みプログラムの記述

- 7.1 割り込みを使うには
 - 7.2 スタートアッププログラム
 - 7.3 割り込みの初期化
 - 7.4 割り込み関数
-

7.1 割り込みを使うには

割り込みを使うには

1. 割り込みベクタの設定

- ◆ 可変割り込みベクタテーブルの配置アドレスを設定
→ ルネサス統合開発環境上で行う
- ◆ 割り込みベクタに関数の先頭アドレスを設定

2. 割り込みの許可

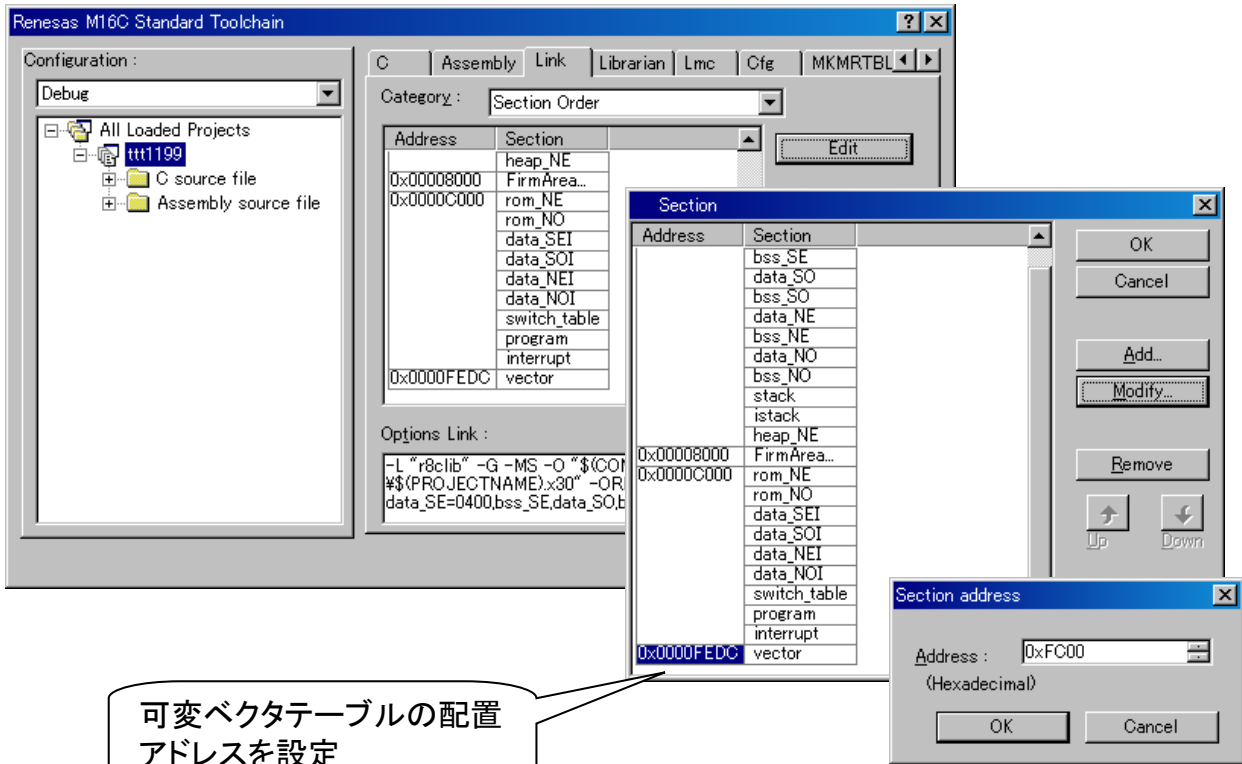
- ◆ 使用する割り込みの割り込み優先レベル設定
- ◆ 割り込み許可フラグ（I）の許可
→ スタートアッププログラム(resetprg.c)内、
あるいはインラインアセンブルにより許可

3. 割り込み関数の作成と割り込み関数指定

- ◆ プリプロセスコマンドによる割り込み関数の指定

7.2 スタートアッププログラム

可変ベクタのアドレス設定



可変ベクタテーブルの配置
アドレスを設定

Code : R8Cコース

Date : Rev. 2. 20

Page: 2 of 9

可変ベクタの先頭アドレス設定は、ルネサス統合開発環境上で行う。

固定ベクタへの関数登録

```

1.  #include "vector.h"
2.  #pragma sectaddress fvector,ROMDATA Fvectaddr
3.
4.  #pragma interrupt/v _dummy_int //udi
5.  #pragma interrupt/v _dummy_int //over_flow
6.  #pragma interrupt/v _dummy_int //brki
7.  #pragma interrupt/v _dummy_int //address_match
8.  #pragma interrupt/v wdt_int //wdt
9.  #pragma interrupt/v _dummy_int //reserved
10. #pragma interrupt/v _dummy_int //reserved
11. #pragma interrupt/v start
12.
13. _asm("        .ofsreg  0FFH");
14.
15. #pragma interrupt _dummy_int()
16. void _dummy_int(void){ }

```

割り込み関数の
先頭アドレス登録

.glb _wdt_int
.lword _wdt_int
に展開される

ダミー割り込み関数

Code : R8Cコース

Date : Rev. 2. 20

Page:3 of 9

※ 固定ベクタに設定した割り込み関数の実体定義で、必ず“#pragma interrupt”指定を行うこと。

記述例)

```
#pragma INTERRUPT wdt_int
```

```
void wdt_int( void )
```

```
{
```

```
    割り込み処理
```

```
    :
```

```
}
```

可変ベクタへの関数登録

intprg.c

```
void int_func(void);  
:  
:  
#pragma interrupt _int0(vect=29)  
void _int0 (void)  
{  
  
    int_func();  
  
}  
:  
:
```

エントリ関数から呼び出す関数のプロトタイプ宣言

割り込みエントリ関数

実処理

Code : R8Cコース

Date : Rev. 2.20

Page: 4 of 9

※ 可変ベクタへの割り込み関数の先頭アドレス設定については、スタートアッププログラムの intprg.c 内で、使用するグループの周辺機能割り込みが全て「エントリ関数」として登録されている。
したがって、ユーザはエントリ関数内で実処理を呼び出すだけでよい。

“#pragma interrupt” 指定の詳細については、後述。

7.3 割り込みの初期化

割り込み優先レベルの設定

```
void main(void)
{
    initial();
    asm( "FSET 1"); /* 割り込み許可 */
    :
}

void initial(void)
{
    INT1IC = 0x03; /* INT1割り込みの優先レベルを */
                  /* "3" に設定 */
    :
    :
}
```

初期化関数コール

7.4 割り込み関数

割り込み関数の宣言 (固定ベクタの割り込み指定)

割り込み関数の宣言

#pragma INTERRUPT 割り込み関数名

記述例

void intr(void) ;
#pragma INTERRUPT intr

void intr(void)
{
:
}

割り込み処理

効果

- ◆ 全レジスタのスタックへの退避・復帰
- ◆ reit命令によるリターン

引数、戻り値は
必ずvoid型

展開イメージ

```
.section      program
.glb  _intr

_intr:
pushm  R0, R1, R2, R3, A0, A1, FB
:
割り込み処理
:
popm    R0, R1, R2, R3, A0, A1, FB
reit
```

全レジスタの退避

割り込み処理

退避したレジスタの復帰

reit命令による関数からの復帰

Code : R8Cコース

Date : Rev. 2. 20

Page: 6 of 9

NC30では、#pragma INTERRUPTを使用することにより、割り込み処理を通常の関数と同様に記述できる。

〔書式〕

- ① #pragma INTERRUPT [/B : /E : /V] 割り込み処理関数名
 - ② #pragma INTERRUPT [/B : /E] 割り込みベクタ番号 割り込み処理関数名
 - ③ #pragma INTERRUPT [/B : /E] 割り込み処理関数名 (vect=割り込みベクタ番号)
- []内は省略可能

※ /Vの指定は、固定割り込みベクタに関数の先頭アドレスを設定するときのみ指定が可能。

割り込み関数の宣言 (可変ベクタの割り込み指定)

割り込み処理関数の宣言

```
#pragma INTERRUPT ベクタ番号 割り込み関数名
#pragma INTERRUPT 割り込み関数名 (vect=ベクタ番号)
```

記述例

```
void intr( void ) ;
#pragma INTERRUPT 31 intr

void intr( void )
{
    割り込み処理
    :
}
```

使用する割り込みの
ベクタ番号(10進数で表記)

効果

- ◆ 可変ベクタテーブルを自動的に生成
- ◆ 指定したベクタ番号の領域に関数の先頭アドレスを設定

Code : R8Cコース

Date : Rev. 2. 20

Page: 7 of 9

```
void intr(void);
#pragma interrupt 31 intr
void intr(void)
{
    int_func();
}
```

展開イメージ

```
.section program
.glb _intr
.glb _int_func
.rvector 31, _intr

_intr:
pushm    R0,R1,R2,R3,A0,A1,FB
jsr      _int_func
popm     R0,R1,R2,R3,A0,A1,FB
reit
```


レジスタバンク切り替えによるレジスタの退避/復帰指定

レジスタバンク切り替えを使用する割り込み関数の指定

#pragma INTERRUPT /B 割り込み関数名

記述例

```
void intr( void ) ;
#pragma INTERRUPT /B intr

void intr( void )
{
    割り込み処理
    :
}
```

効果

レジスタバンク切り替えによる
割り込み応答の高速化

展開イメージ

```
.section      program
.glb  _intr

_intr:
    fset      B
    :
    割り込み処理
    :
    reit
```

レジスタバンクを
0 → 1 に切り替え

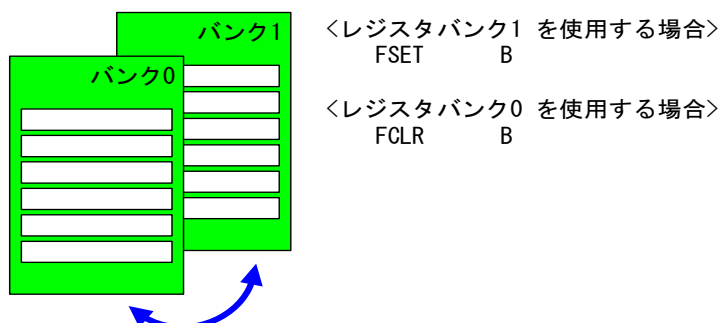
reit 命令により、
レジスタバンクが
1 → 0 に復帰

Code : R8Cコース

Date : Rev. 2. 20

Page: 8 of 9

※ メインプログラム内でレジスタバンク0, 1を共に使用している場合、
および多重割り込みを許可している場合は、レジスタバンク切り
替えによるレジスタの退避・復帰は不可。



多重割り込みを許可する割り込み関数の指定

多重割り込み許可指定

`#pragma INTERRUPT /E 割り込み関数名`

記述例

```
void intr( void ) ;  
#pragma INTERRUPT /E intr  
  
void intr( void )  
{  
    :  
    :  
    :  
}
```

割り込み処理

効果

関数の入口で割り込み許可フラグ(1フラグ)をセットし、多重割り込みを許可する

展開イメージ

```
.section      program  
.glob  _intr  
  
_intr:  
    fset 1  
    pushm R0, R1, R2, R3, A0, A1, FB  
    :  
    :  
    :  
    popm  R0, R1, R2, R3, A0, A1, FB  
    reit
```

割り込み許可
フラグのセット

割り込み処理

Code : R8Cコース

Date : Rev. 2. 20

Page: 9 of 9

#pragma INTERRUPT 指定において、'／B' と '／E' の同時指定はできない。

第8章 *R8C/25グループの内蔵周辺機能*

- 8.1 プログラマブル入出力ポート
 - 8.2 タイマ
 - 8.3 ウォッチドッグタイマ
 - 8.4 シリアルインタフェース
 - 8.5 A/Dコンバータ
 - 8.6 フラッシュメモリ
-

R8C/25グループの内蔵周辺機能

- 入出力ポート：44本（うち3本は入力専用ポート、LED駆動用ポート8本あり）
- 8ビットタイマ：2ch（タイマRA、タイマRB）
 - ◆ タイマモード（タイマRA、タイマRB）
 - ◆ パルス出力モード（タイマRAのみ）
 - ◆ パルス幅／周期測定モード（タイマRAのみ）
 - ◆ イベントカウンタモード（タイマRAのみ）
 - ◆ プログラマブル波形発生モード（タイマRBのみ）
 - ◆ プログラマブルワンショット／ウエイトワンショット発生モード（タイマRBのみ）
- 16ビットタイマ：2ch（タイマRD）
 - ◆ タイマモード（インプットキャプチャ、アウトプットコンペア機能付き）
 - ◆ PWMモード
 - ◆ リセット同期PWMモード
 - ◆ 相補PWMモード
 - ◆ PWM3モード
- リアルタイムクロック：1ch（タイマRE）
 - ◆ コンペアマッチ機能付き
- ウォッチドッグタイマ：1ch（リセットスタート機能あり）
- シリアルインタフェース：2ch
 - ◆ クロック同期形／非同期形選択可
- A/Dコンバータ：1回路（アナログ入力端子：12本、分解能：10ビット）
- クロック同期形シリアルインタフェース：1ch
 - ◆ I²Cバスインタフェース／チップセレクト付クロック同期形シリアルI/Oを選択
- LINモジュール：ハードウェアLIN 1ch（タイマRA、UART0を使用）

8.1 プログラマブル入出力ポート

入出力ポートの制御

ポートPi方向レジスタ (i=0~4、6) (注1、2)

シンボル	アドレス	リセット後の値					
PD0（注3）	00E2h番地	00h					
PD1	00E3h番地	00h					
PD2	00E6h番地	00h					
PD3	00E7h番地	00h					
PD4	00EAh番地	00h					
PD6	00EEh番地	00h					
ビット シンボル	ビット名	機能				RW	
PDi_0	ポートPi_0方向ビット	0：入力モード （入力ポートとして機能） 1：出力モード （出力ポートとして機能）				RW	
PDi_1	ポートPi_1方向ビット					RW	
PDi_2	ポートPi_2方向ビット					RW	
PDi_3	ポートPi_3方向ビット					RW	
PDi_4	ポートPi_4方向ビット					RW	
PDi_5	ポートPi_5方向ビット					RW	
PDi_6	ポートPi_6方向ビット					RW	
PDi_7	ポートPi_7方向ビット					RW	

注1. PD0レジスタは、PRCRレジスタのPRC2ビットを“1”（書き込み許可）にした次の命令で書いてください。

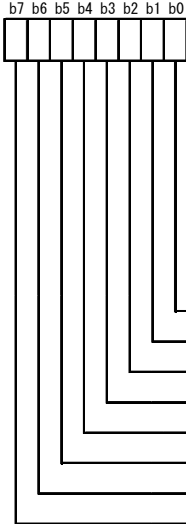
注2. PD3レジスタのPD3_4~PD3_6ビットは何も配置されていません。

PD3_4~PD3_6ビットに書く場合、“0”（入力モード）を書いてください。読んだ場合、その値は不定です。

注3. PD4レジスタのPD4_0~PD4_4ビット、PD4_6ビットとPD4_7ビットは何も配置されていません。

PD4レジスタのPD4_0~PD4_4ビット、PD4_6ビットとPD4_7ビットに書く場合、“0”（入力モード）を書いてください。読んだ場合、その値は不定です。

ポートPiレジスタ (i=0~4、6) (注1、2)

								シンボル	アドレス	リセット後の値
	P0	00E0h番地	不定							
	P1	00E1h番地	不定							
	P2	00E4h番地	不定							
	P3	00E5h番地	不定							
	P4	00E8h番地	不定							
	P6	00ECh番地	不定							
ビット シンボル	ビット名	機能	RW							
Pi_0	ポートPi_0ビット	入力モードに設定した入出力ポートに 対応するビットを読むと、端子のレ ベルが読める。 出力モードに設定した入出力ポートに 対応するビットに書くと、端子のレ ベルを制御できる 0：“L”レベル 1：“H”レベル(注1)	RW							
Pi_1	ポートPi_1ビット		RW							
Pi_2	ポートPi_2ビット		RW							
Pi_3	ポートPi_3ビット		RW							
Pi_4	ポートPi_4ビット		RW							
Pi_5	ポートPi_5ビット		RW							
Pi_6	ポートPi_6ビット		RW							
Pi_7	ポートPi_7ビット		RW							

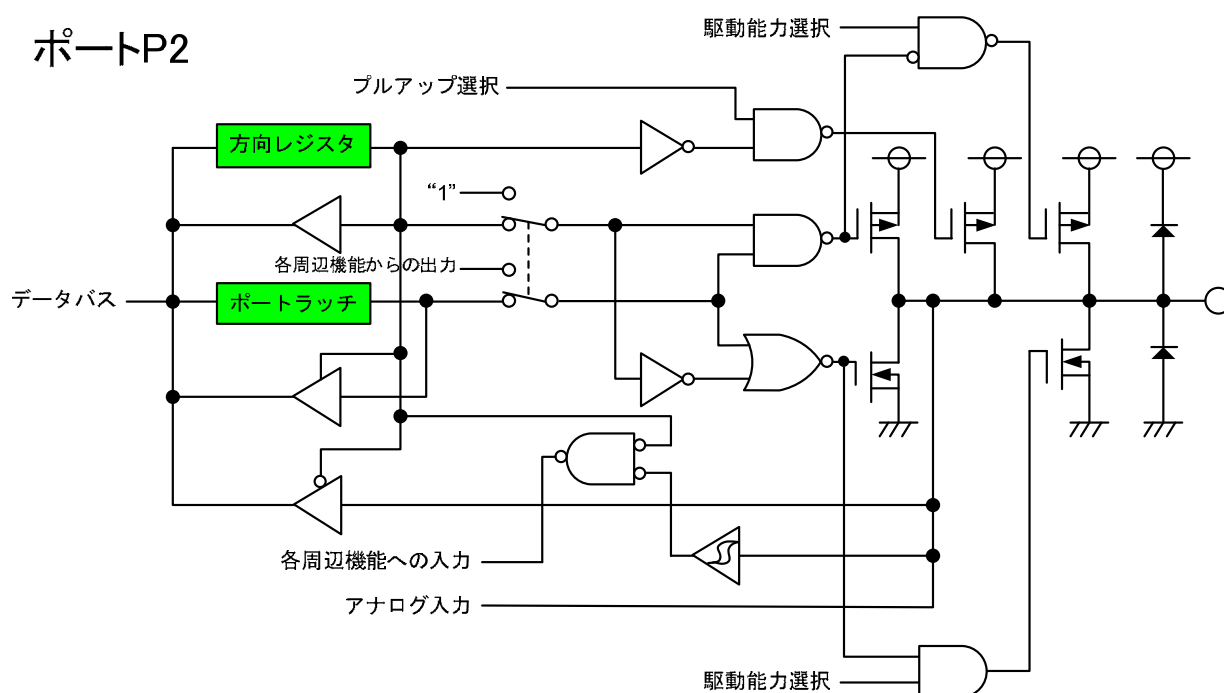
注1. P3レジスタのP3_4~P3_6ビットは何も配置されていません。

P3_4~P3_6ビットに書く場合、“0”（“L”レベル）を書いてください。読んだ場合、その内容は不定です。

注2. P4レジスタのP4_0~P4_4ビットは何も配置されていません。

P4_0~P4_4ビットに書く場合、“0”（“L”レベル）を書いてください。読んだ場合、その内容は不定です。

入出力ポートの構成



Code : R8Cコース

Date : Rev. 2.20

Page:3 of 63

■ポートの入出力

ポートは方向レジスタにより1ビット単位で入出力を設定できる。

■ポートの機能

ポート端子に複数の機能が割り付けられている場合は、使用する周辺機能により端子機能が切り替わる。

■周辺機能の入力端子として使用する場合

周辺機能の入力端子として使用する場合、対応するポートの方向レジスタを「入力」に設定すること。







プリアップ制御レジスタ

プリアップ制御レジスタ0

b7	b6	b5	b4	b3	b2	b1	b0	シンボル	アドレス	リセット後の値	
								PUR0	00FCh番地	00000000b	
								ビット シンボル	ビット名	機能	
								PU00	P0_0～P0_3のブルアップ(注1)	0：ブルアップなし 1：ブルアップあり	
								PU01	P0_4～P0_7のブルアップ(注1)		RW
								PU02	P1_0～P1_3のブルアップ(注1)		RW
								PU03	P1_4～P1_7のブルアップ(注1)		RW
								PU04	P2_0～P2_3のブルアップ(注1)		RW
								PU05	P2_4～P2_7のブルアップ(注1)		RW
								PU06	P3_0、P3_1、P3_3のブルアップ(注1)		RW
								PU07	P3_4～P3_5、P3_7のブルアップ(注1)		RW

注1. このビットが“1”（プリアップあり）で、かつ方向ビットが“0”（入力モード）のとき端子がプリアップされます。

プリアップ制御レジスタ1

b7	b6	b5	b4	b3	b2	b1	b0
			0	0			
シンボル		アドレス		リセット後の値			
PUR1		00FDh番地		XX00XX00b			
ビット シンボル	ビット名			機能		RW	
PU10	P4_3のブルアップ（注1）			0：ブルアップなし 1：ブルアップあり		RW	
PU11	P4_4、P4_5のブルアップ（注1）			0：ブルアップなし 1：ブルアップあり		RW	
－ (b3-b2)	予約ビット			“0”にしてください。		RW	
PU14	P6_0～P6_3のブルアップ（注1）			0：ブルアップなし 1：ブルアップあり		RW	
PU15	P6_4～P6_7のブルアップ（注1）			1：ブルアップあり		RW	
－ (b7-b2)	何も配置されていない。書く場合、 読んだ場合、その値は不定。			“0”を書いてください。		－	

注1. このビットが“1”（プリアップあり）で、かつ方向ビットが“0”（入力モード）のとき端子がプリアップされます。

Code : R8Cコース

Date : Rev. 2. 20

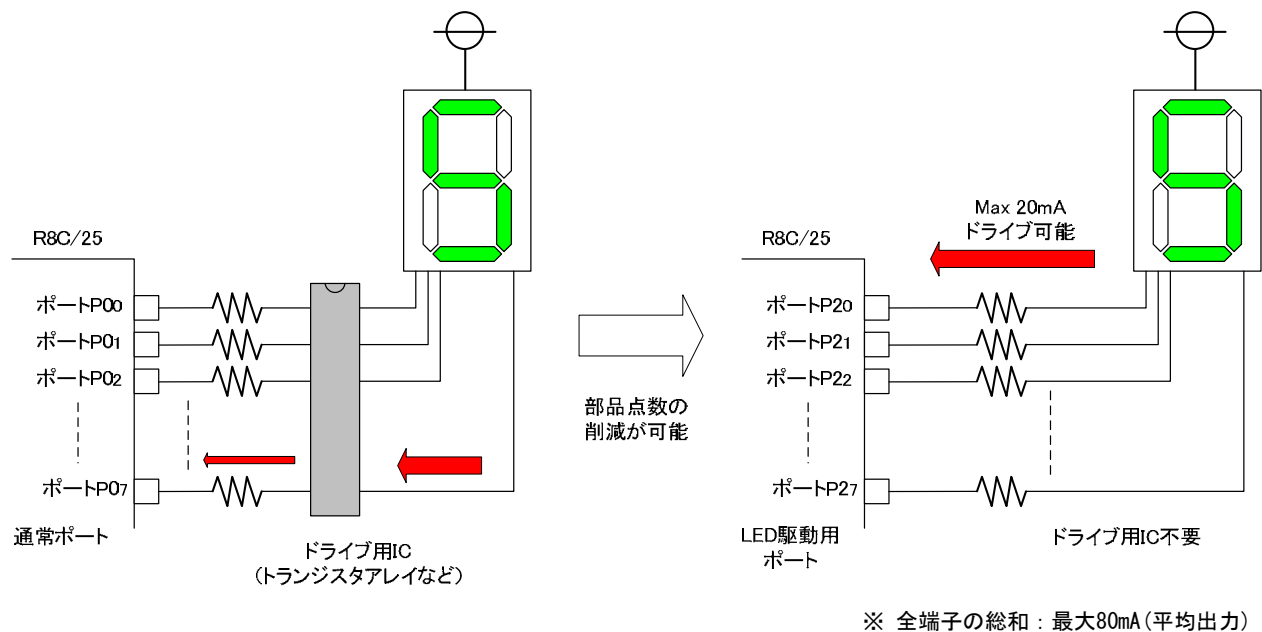
Page: 4 of 63

入力ポートに設定されているポートに対してのみプリアップ抵抗が有効になる。

プリアップ抵抗値 : 平均 50k Ω (Vcc=5V時)

※ Vcc=5V時 : 30k Ω \leq Rpullup \leq 167k Ω

ポートP2駆動能力制御



Code : R8Cコース

Date : Rev. 2.20

Page:5 of 63

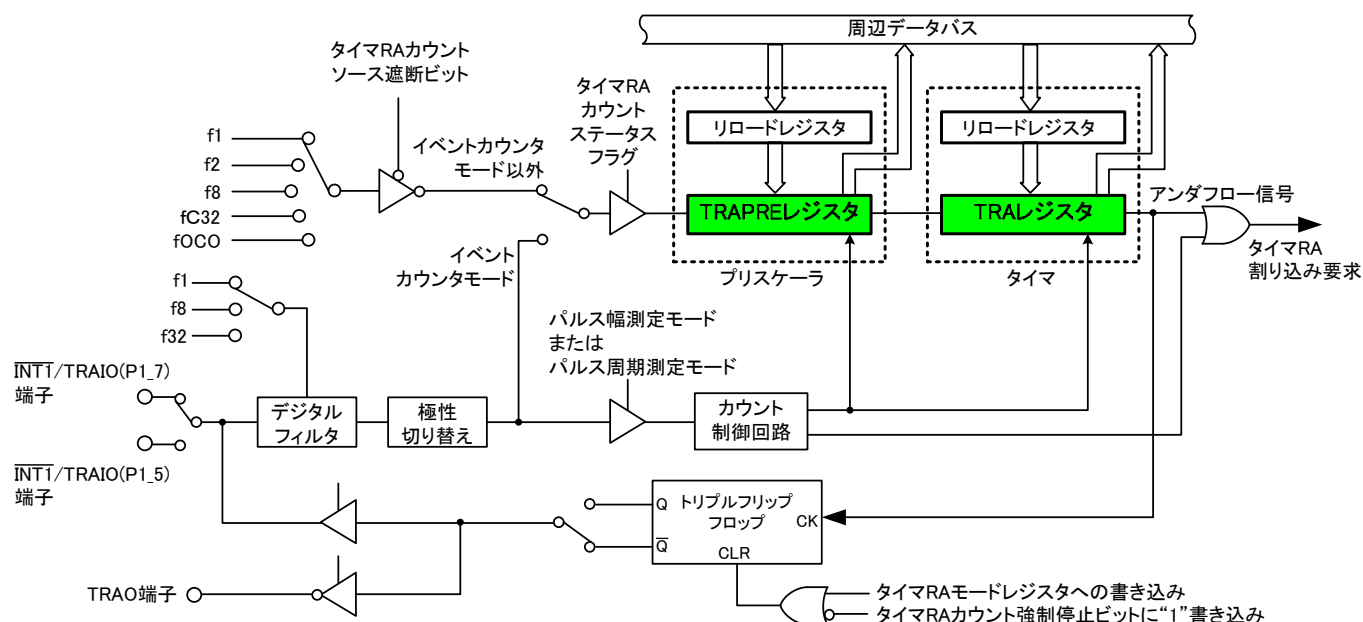
ポートP2駆動能力制御レジスタ

シンボル P2DRR	アドレス 00F4h番地	リセット後の値 00h	
ビット シンボル	ビット名	機能	RW
P2DRR0	P2_0の駆動能力	P2の出力トランジスタの駆動能力 設定を行う 0 : Low 1 : High (注1)	RW
P2DRR1	P2_1の駆動能力		RW
P2DRR2	P2_2の駆動能力		RW
P2DRR3	P2_3の駆動能力		RW
P2DRR4	P2_4の駆動能力		RW
P2DRR5	P2_5の駆動能力		RW
P2DRR6	P2_6の駆動能力		RW
P2DRR7	P2_7の駆動能力		RW

注1. Highレベル、Lowレベルともに駆動能力を高くする。

8.2.1 タイマRA

タイマRAの構成



Code : R8Cヨース

Date : Rev. 2. 20

Page:6 of 63

タイマRAは、8ビットのプリスケアラとタイマで構成され、それぞれリロードレジスタを1つもつ。

タイマカウント値(プリスケアラ×カウンタ)算出方法

$$(n+1)(m+1) = \text{割り込み発生間隔(s)} \times \text{カウントソースの周波数(Hz)}$$

n: プリスケール値 (TRAPREレジスタ設定値)

m: タイマ値 (TRAレジスタ設定値)

例) 2ms周期で割り込みを発生させる (XINクロック20MHz時で、
カウントソースとしてXINクロックの2分周 (f2) を選択した場合)

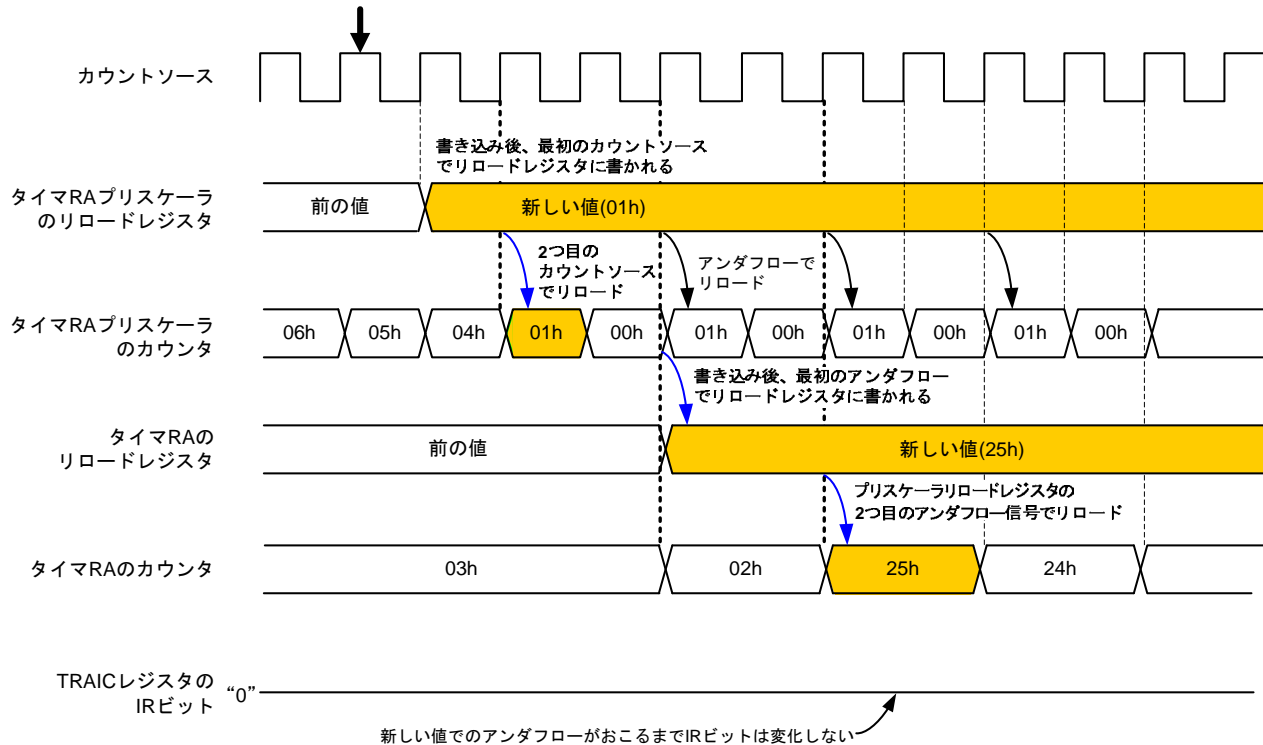
$$\begin{aligned}(n+1)(m+1) &= 2 \times 10^{-3} \times 10 \times 10^6 \\ &= 20 \times 10^3 \\ &= 200 \times 100\end{aligned}$$

$$\therefore n = 200 - 1$$

$$m = 100-1$$

カウント中のタイマへの書き込み動作

プログラムでTRAPREレジスタに“01h”、TRAレジスタに“25h”を書く



上図は、TRACRレジスタのTSTARTビット、TCSTFビットがともに“1” (カウント中) の場合です。

Code : R8Cコース

Date : Rev. 2. 20

Page: 7 of 63

タイマRAはプリスケアラと、タイマ（プリスケアラのアンダフローをカウントする狭義のタイマ）を持ち、それぞれにリロードレジスタとカウンタがあります。

プリスケアラやタイマに書き込む場合、リロードレジスタとカウンタの両方に値が書き込まれます。この際、プリスケアラのリロードレジスタからカウンタへはカウントソースに同期して値を転送しますが、タイマのリロードレジスタからカウンタへはプリスケアラのアンダフローに同期して値を転送します。

このため、カウント中にプリスケアラやタイマに書き込みを行っても、実行後すぐにはタイマRAのリロードレジスタ、およびカウンタの値が更新されませんので、注意してください。

タイマRA関連レジスタ(1)

タイマRAモードレジスタ (注1)

シンボル TRAMR	アドレス 0102h番地	リセット後の値 00h	
ビット シンボル	ビット名	機能	RW
TMOD0	タイマRA動作モード選択 ビット	b2 b1 b0 0 0 0 : タイマモード 0 0 1 : パルス出力モード 0 1 0 : イベントカウンタモード 0 1 1 : パルス幅測定モード 1 0 0 : パルス周期測定モード 1 0 1 : } 1 1 0 : } 設定しない 1 1 1 : }	RW
TMOD1			RW
TMOD2			RW
— (b3)		何も配置されていない。 書く場合、“0”を書く。読んだ場合、その値は“0”。	—
TCK0			RW
TCK1	タイマRAカウントソース選択 ビット	b6 b5 b4 0 0 0 : f1 0 0 1 : f8 0 1 0 : f0C0 0 1 1 : f2 1 0 0 : f032 1 0 1 : } 1 1 0 : } 設定しない 1 1 1 : }	RW
TCK2			RW
TCKCUT		0 : カウントソース供給 1 : カウントソース遮断	RW

注1. TRACRレジスタのTSTARTビットとTCSTFビットがともに“0”（カウント停止）のときに変更する。

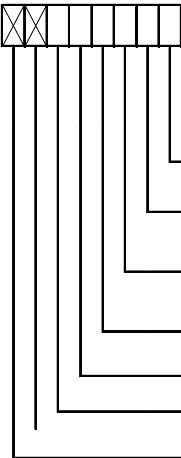
タイマRA制御レジスタ (注4)

シンボル TRACR	アドレス 0100h番地	リセット後の値 00h	
ビット シンボル	ビット名	機能	RW
TSTART	タイマRAカウント 開始ビット(注1)	0 : カウント停止 1 : カウント開始	RW
TCSTF	タイマRAカウント ステータスフラグ(注1)	0 : カウント停止 1 : カウント中	RO
TSTOP	タイマRAカウント 強制停止ビット(注2)	“1”を書くとカウントが強制停止します。 読んだ場合、その値は“0”。	RW
— (B3)	何も配置されていない。 書く場合、“0”を書く。読んだ場合、その値は“0”。		—
TEDGF	有効エッジ判定フラグ (注3、5)	0 : 有効エッジなし 1 : 有効エッジあり（測定期間終了）	RW
TUNDF	タイマRAアンダフロー フラグ(注3、5)	0 : アンダフローなし 1 : アンダフローあり	RW
— (b7-b6)	何も配置されていない。 書く場合、“0”を書く。読んだ場合、その値は“0”。		—

注1. カウント動作中にカウントソースを切り替えしないでください。カウントソースを切り替えるときは、タイマのカウントを停止してください。

タイマRA関連レジスタ(2)

タイマRA I/O制御レジスタ

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRAIOG	アドレス 0101h番地	リセット後の値 00h	
	ビット シンボル	ビット名	機能	RW
	TEDGSEL	TRAIO極性切り替えビット	動作モードによって機能が異なる。	RW
	TOPCR	TRAIO出力制御ビット		RW
	TOENA	TRAIO出力許可ビット		RW
	TIOSEL	INT1/TRAIO入力フィルタ選択ビット		RW
	TIPF0	タイマRAカウントソース選択ビット		RW
	TIPF1	ビット		RW
	— (b7-b6)	何も配置されていない。 書く場合、“0”を書く。読んだ場合、その値は“0”。		—

タイマRAプリスケアラレジスタ

b7

b0

シンボル

TRAPRE

アドレス

0103h番地

リセット後の値

FFh(注1)

モード	機能	設定範囲	RW
タイマモード	内部カウントソースをカウント	00h～FFh	RW
パルス出力モード			RW
イベントカウンタモード	外部カウントソースをカウント		RW
パルス幅測定モード	内部カウントソースをカウント		RW
パルス周期測定モード			RW

タイマRAレジスタ

b7		b0		
<div></div>		シンボル	アドレス	リセット後の値
		TRA	0104h番地	FFh(注1)
		機能	設定範囲	RW
		タイマRAプリスケアラレジスタのアンダフローをカウント	00h~FFh	RW

注1. TRACRレジスタのTSTOPビットに“1”を書くとTRAレジスタは“FFh”になります。

タイマモード

内部で生成されたカウントソースをカウントし、カウンタがアンダフローするごとに割り込み要求を発生させる。

項 目	仕 様
カウントソース ※	f1, f2, f8, fOCO, fC32
分周比	$1/(n+1)(m+1)$ n:TRAPREレジスタ設定値、m:TRAレジスタ設定値
カウント開始条件	タイマRAカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRAカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRAカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	タイマRAのアンダフロー時(タイマRA割り込み)
INT1/TRAIO端子機能	プログラマブル入出力ポート、またはINT1割り込み入力
TRA0端子機能	プログラマブル入出力ポート
タイマの読み出し動作	TRAレジスタ、TRAPREレジスタを読み出すと、それぞれカウント中の値が読み出される
タイマの書き込み動作	タイマの停止中、動作中に関わらず TRAPREレジスタ、およびTRAレジスタに書き込むと、それぞれリロードレジスタとカウンタの両方に書き込まれる

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

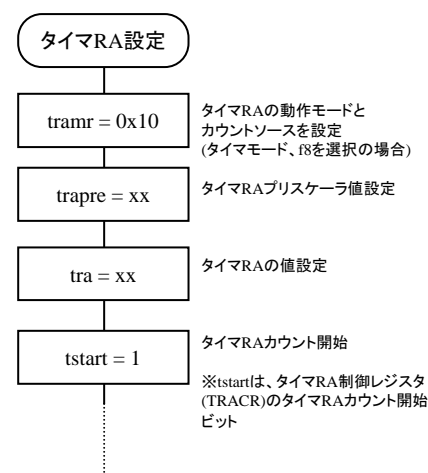
Code : R8Cコース

Date : Rev. 2.20

Page:10 of 63

タイマRA I/O制御レジスタ (タイマモード時)

シンボル TRAIOC	アドレス 0101h番地	リセット後の値 00h	
ビット シンボル	ビット名	機能	RW
TEDGSEL	TRAIO極性切り替えビット	タイマモードでは“0”にしてください。	RW
TOPCR	TRAIO出力制御ビット		RW
TOENA	TRA0出力許可ビット		RW
TIOSEL	INT1/TRAIO選択ビット	0 : INT1/TRAIO端子 (P1_7) 1 : INT1/TRAIO端子 (P1_5)	RW
TIPF0	TRAIO入力フィルタ選択ビット	タイマモードでは“0”にしてください。	RW
TIPF1		ください。	RW
— (b7-b6)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”		—



パルス出力モード

内部で生成されたカウントソースをカウントし、タイマがオーバフローするごとに極性を反転したパルスをTRAIO端子から出力する。

パルス初期出力レベルの選択が可能。またTRA0端子からTRAIO出力の極性を反転したパルスを出力できる。

項 目	仕 様
カウントソース ※	f1, f2, f8, fOCO, fC32
分周比	$1/(n+1)(m+1)$ n: TRAPREレジスタ設定値、m: TRAレジスタ設定値
カウント開始条件	タイマRAカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRAカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRAカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	タイマRAのアンダフロー時(タイマRA割り込み)
INT1/TRAIO端子機能	パルス出力、またはプログラマブル出力ポート、INT1割り込み入力
TRA0端子機能	プログラマブル入出力ポート、またはTRAIO出力の反転出力
タイマの読み出し動作	TRAレジスタ、TRAPREレジスタを読み出すと、それぞれカウント中の値が読み出される
タイマの書き込み動作	タイマの停止中、動作中に問わず TRAPREレジスタ、およびTRAレジスタに書き込むと、それぞれリロードレジスタとカウンタの両方に書き込まれる

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

Code : R8Cコース

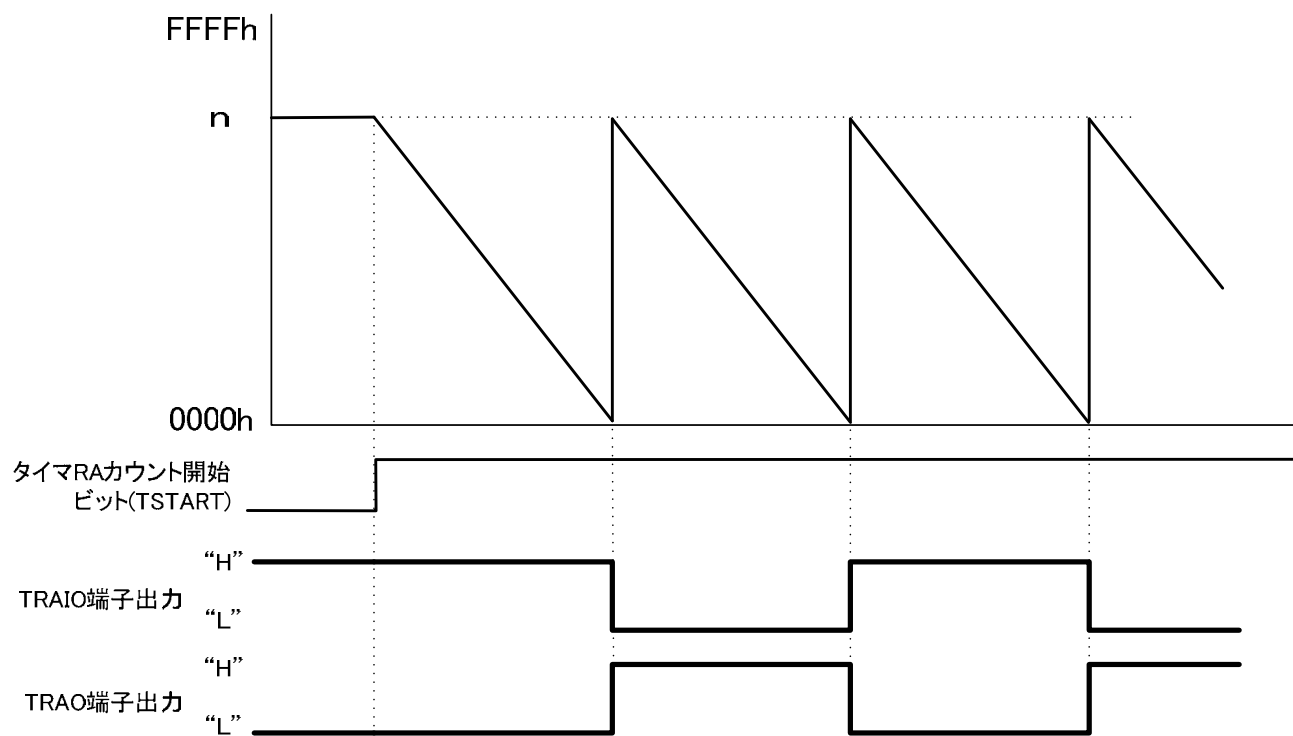
Date : Rev. 2.20

Page: 11 of 63

タイマRA I/O制御レジスタ (パルス出力モード時)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRAIOC	アドレス 0101h番地	リセット後の値 00h	
0 0	ビット シンボル	ビット名	機能	RW
	TEDGSEL	TRAIO極性切り替えビット	0 : “H” からTRAIO出力開始 1 : “L” からTRAIO出力開始	RW
	TOPCR	TRAIO出力制御ビット	0 : TRAIO出力 1 : ポートP1_7またはポートP1_5	RW
	TOENA	TRA0出力許可ビット	0 : ポートP3_0 1 : TRA0出力(P3_0からTRAIOの反転出力)	RW
	TIOSEL	INT1/TRAIO選択ビット	0 : INT1/TRAIO端子 (P1_7) 1 : INT1/TRAIO端子 (P1_5)	RW
	TIPF0	TRAIO入力フィルタ選択ビット	パルス出力モードでは“0”にしてください。	RW
	— (b7-b6)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”		—

パルス出力モードの動作例



タイマRA I/O制御レジスタの“TEDGSELビット”でTRAIO初期出力レベルを“H”で、TRA0出力を許可とした場合

イベントカウンタモード

INT1/TRAI0端子から入力する外部信号をカウントし、カウンタがアンダフローするごとに割り込み要求を発生させる。
INT1/TRAI0端子に入力されるカウントソースの有効エッジを選択可能。

項 目	仕 様
カウントソース	TRAIO端子に入力された外部信号
分周比	$1/(n+1)(m+1)$ n: TRAPREレジスタ設定値、m: TRAレジスタ設定値
カウント開始条件	タイマRAカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRAカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRAカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	タイマRAアンダフロー時(タイマRA割り込み)
INT1/TRAI0端子機能	カウントソース入力(INT1割り込み入力)
TRA0端子機能	プログラマブル入出力ポート、またはパルス出力
タイマの読み出し動作	TRAレジスタ、TRAPREレジスタを読み出すと、それぞれカウント中の値が読み出される
タイマの書き込み動作	タイマの停止中、動作中に関わらず TRAPREレジスタ、およびTRAレジスタに書き込むと、それぞれリロードレジスタとカウンタの両方に書き込まれる

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

Code : R8Cコース

Date : Rev. 2.20

Page:13 of 63

タイマRA I/O制御レジスタ (イベントカウントモード時)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRAIOC	アドレス 0101h 番地	リセット後の値 00h	
	ビット シンボル	ビット名	機能	RW
	TEDGSEL	TRAIO極性切り替えビット	0 : TRAI0入力の立ち上がりエッジでカウント、 またTRA0端子は“L”から出力開始 1 : TRAI0入力の立ち下がりエッジでカウント	RW
	TOPCR	TRAIO出力制御ビット	イベントカウントモードでは“0”にしてください。	RW
	TOENA	TRA0出力許可ビット	0 : ポートP3_0 1 : TRA0出力(P3_0からTRAIOの反転出力)	RW
	TIOSEL	INT1/TRAI0選択ビット	0 : INT1/TRAI0端子 (P1_7) 1 : INT1/TRAI0端子 (P1_5)	RW
	TIF0	TRAIO入力フィルタ選択ビット ください。(注1)	b5b4 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング	RW
	TIF1			RW
	— (b7-b6)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”		—

注1. TRAI0端子から同じ値を3回連続してサンプリングした時点で入力が確定します。

パルス幅測定モード

INT1/TRAO端子から入力する外部信号のパルス幅を測定するモード。入力パルスの測定幅として“H”レベル期間“L”レベル期間を選択可能。

項 目	仕 様
カウントソース ※	f1, f2, f8, fOCO, fC32
カウント動作	測定パルスの“H”レベル期間または“L”レベル期間のみカウント(ダウンカウント)を行う。
カウント開始条件	タイマRAカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRAカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRAカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	●タイマRAのアンダフロー時 → タイマRA割り込み要求 ●測定期間終了時(TRAIO端子への立ち上がり、または立ち下がりがエッジ入力時) → タイマRA割り込み要求
INT1/TRAO端子機能	測定パルス入力(INT1割り込み入力)
TRA0端子機能	プログラマブル入出力ポート
タイマの読み出し動作	TRAREGレジスタ、TRAPREレジスタを読み出すと、それぞれカウント中の値が読み出される
タイマの書き込み動作	タイマの停止中、動作中に問わず TRAPREレジスタ、およびTRAREGレジスタに書き込むと、それぞれリロードレジスタとカウンタの両方に書き込まれる

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

Code : R8Cコース

Date : Rev. 2.20

Page:14 of 63

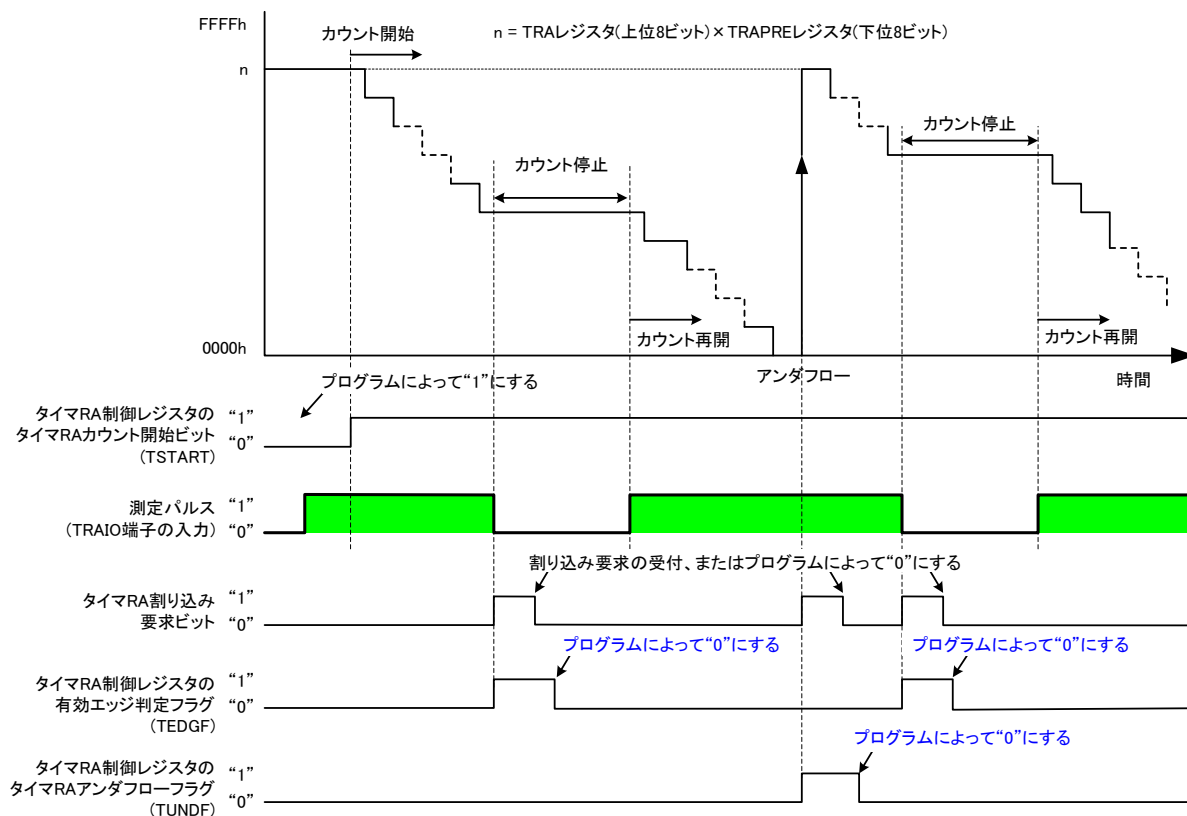
タイマRA I/O制御レジスタ (パルス幅測定モード時)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRA10C	アドレス 0101h番地	リセット後の値 00h	
	ビット シンボル	ビット名	機能	RW
	TEDGSEL	TRA10極性切り替えビット	0 : TRA10入力の“L”レベル幅を測定 1 : TRA10入力の“H”レベル幅を測定	RW
	TOPCR	TRA10出力制御ビット	パルス幅測定モードでは“0”にしてください。	RW
	TOENA	TRA0出力許可ビット		RW
	TIOSEL	INT1/TRAO選択ビット	0 : INT1/TRAO端子 (P1_7) 1 : INT1/TRAO端子 (P1_5)	RW
	TIPF0	TRA10入力フィルタ選択ビット ください。(注1)	b5b4 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング	RW
	TIPF1			RW
	— (b7-b6)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”		—

注1. TRA10端子から同じ値を3回連続してサンプリングした時点で入力が確定します。

パルス幅測定モードの動作例

【測定パルスの“H”レベル幅を測定した場合の動作例】

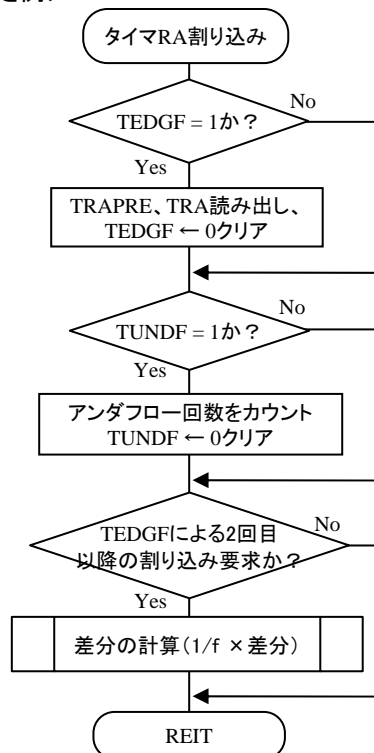


Code : R8Cコース

Date : Rev. 2. 20

Page: 15 of 63

<パルス幅測定例>



※ 連続して入力されるパルスの幅測定をしている場合、タイマRAがアンダフローし、割り込み要求が発生します。この場合リロードレジスタから値がリロードされるため、正しいカウント値を得られなくなります。

よって、タイマRAアンダフローフラグがセットされた回数をカウントしておき、有効エッジ判定フラグがセットされた時点で、パルス幅の計算を行うようにします。

※差分 = 前回読み出したカウント値 + (リロードレジスタに設定した初期値 × アンダフロー回数) - 今回読み出したカウント値

f = カウントソースの周波数

パルス周期測定モード

INT1/TRAO端子から入力する外部信号のパルスの周期を測定するモード。入力パルス周期の測定期間(立ち上がり～立ち上がり、または立ち下がり～立ち下がり)を選択可能。

項 目	仕 様
カウントソース ※	f1, f2, f8, fOCO, fC32
カウント動作	測定パルスの有効エッジ入力後、1回目のプリスケアラのアンダフロー時に読み出し用バッファの内容を保持し、2回目のプリスケアラのアンダフロー時にタイマRAのリロードレジスタの内容をリロードしてカウント(ダウンカウント)を継続する。
カウント開始条件	タイマRAカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRAカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRAカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	●タイマRAのアンダフロー時 → タイマRA割り込み要求 ●測定期間終了時(TRAIO端子への立ち上がり、または立ち下がりエッジ入力時) → タイマRA割り込み要求
INT1/TRAO端子機能	測定パルス入力(INT1割り込み入力)
TRA0端子機能	プログラマブル入出力ポート
タイマの読み出し動作	TRAレジスタ、TRAPREレジスタを読み出すと、それぞれカウント中の値が読み出される
タイマの書き込み動作	タイマの停止中、動作中に問わず TRAPREレジスタ、およびTRAレジスタに書き込むと、それぞれリロードレジスタとカウンタの両方に書き込まれる

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

Code : R8Cコース

Date : Rev. 2.20

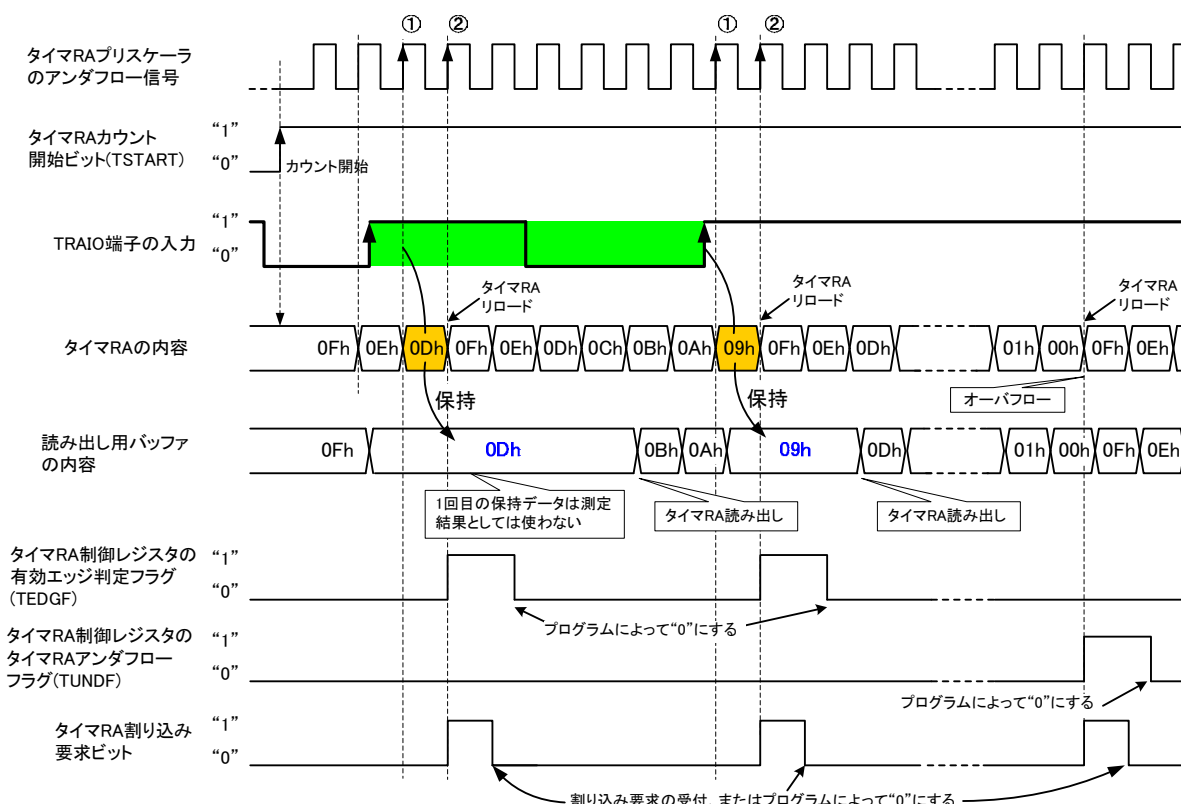
Page:16 of 63

タイマRA I/O制御レジスタ (パルス周期測定モード時)

シンボル	アドレス	リセット後の値	
TRAIOC	0101h 番地	00h	
ビットシンボル	ビット名	機能	RW
TEDGSEL	TRAIO極性切り替えビット	0 : 測定パルスの立ち上がりから立ち上がり間を測定 1 : 測定パルスの立ち下がりから立ち下がり間を測定	RW
TOPCR	TRAIO出力制御ビット	パルス周期測定モードでは“0”にしてください。	RW
TOENA	TRA0出力許可ビット		RW
TIOSEL	INT1/TRAO選択ビット	0 : INT1/TRAO端子 (P1_7) 1 : INT1/TRAO端子 (P1_5)	RW
TIPF0	TRAIO入力フィルタ選択ビット ください。(注1)	b5b4 0 0 : フィルタなし 0 1 : フィルタあり、f1でサンプリング 1 0 : フィルタあり、f8でサンプリング 1 1 : フィルタあり、f32でサンプリング	RW
TIPF1			RW
— (b7-b6)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”		—

注1. TRAIO端子から同じ値を3回連続してサンプリングした時点で入力が増大します。

パルス周期測定モード動作例



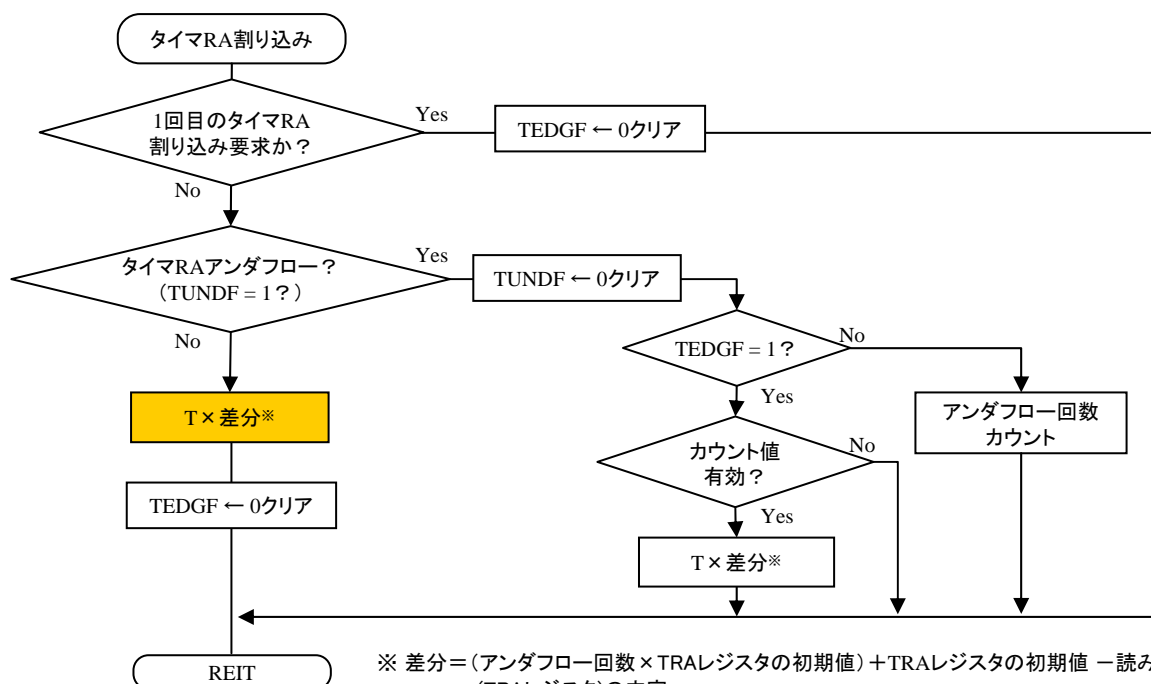
タイマRAレジスタの初期値を0Fhとし、測定パルスの立ち上がりから立ち上がりまでを測定した場合

Code : R8Cコース

Date : Rev. 2.20

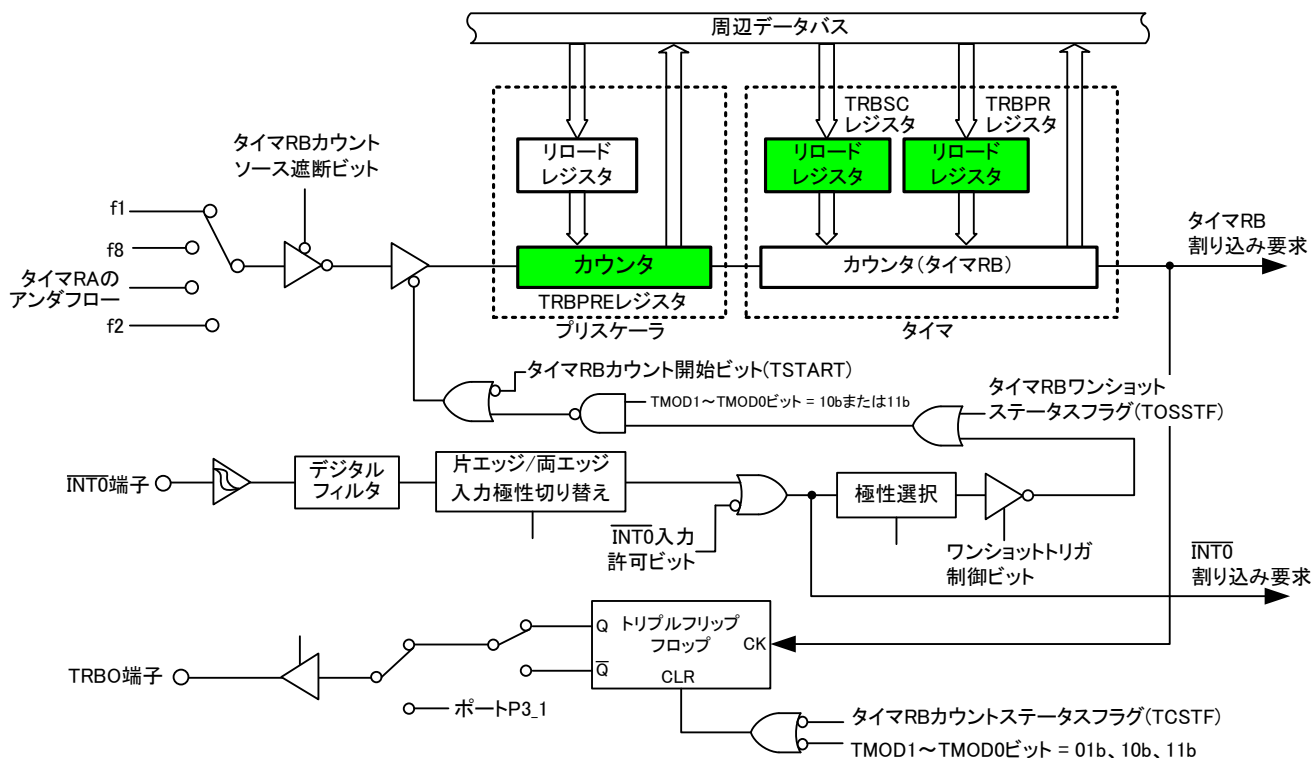
Page:17 of 63

<パルス周期測定例>



8.2.2タイマRB

タイマRBの構成



Code : R8Cコース

Date : Rev. 2.20

Page:18 of 63

タイマRBは、8ビットのプリスケアラとタイマで構成され、タイマ側はプライマリレジスタとセカンダリレジスタ(書き込み専用)の2つのリロードレジスタをもつ。

タイマRBモードレジスタ

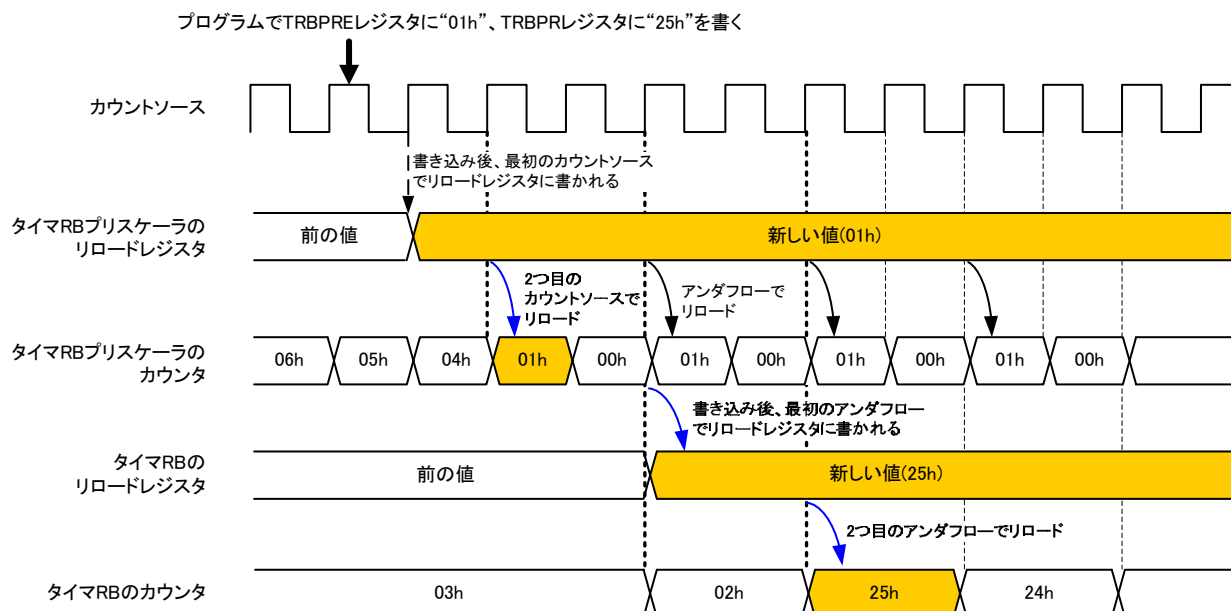
ビット	シンボル	アドレス	リセット後の値
b7 b6 b5 b4 b3 b2 b1 b0	TRBMR	010Bh番地	00h
ビットシンボル	ビット名	機能	RW
TMOD0	タイマRB動作モード選択ビット (注1)	b1 b0 0 0 : タイマモード 0 1 : プログラマブル波形発生モード 1 0 : プログラマブルワンショット発生モード 1 1 : プログラマブルウェイトワンショット発生モー	RW
TMOD1			RW
— (b2)	何も配置されていない。書く場合、“0”を書く。読んだ場合、その値は“0”。		RW
TWRC	タイマRB書き込み制御ビット (注2)	0 : リロードレジスタとカウンタへの書き込み 1 : リロードレジスタのみ書き込み	—
TCK0	タイマRBカウントソース選択ビット (注1)	b5 b4 0 0 : f1 0 1 : f8 1 0 : タイマRAのアンダフロー 1 1 : f2	RW
TCK1			RW
— (b6)	何も配置されていない。書く場合、“0”を書く。読んだ場合、その値は“0”。		—
TCKCUT	タイマRBカウントソース遮断ビット (注1)	0 : カウントソース供給 1 : カウントソース遮断	RW

注1. TMOD1～TMOD0ビット、TCK1～TCK0ビット、TCKCUTビットは、TRBCRレジスタのTSTARTビットとTCSTFビットと共に“0”(カウント停止)のときに変更してください。

注2. TWRCビットは、タイマモードのとき“0”または“1”が選択できます。プログラマブル波形発生モード、プログラマブルワンショット発生モード、プログラマブルウェイトワンショット発生モードでは、“1”(リロードレジスタのみ書き込み)にしてください。

カウント中のタイマへの書き込み動作(1)

TRBMRレジスタのTWRCビットが“0”（リロードレジスタとカウンタへの書き込み）場合



Code : R8Cコース

Date : Rev. 2.20

Page:19 of 63

<タイマRBのタイマ書き込み制御>

タイマRBは、タイマRA同様プリスケアラとタイマ（プリスケアラのアンダフローをカウントする狭義のタイマ）をもち、それぞれにリロードレジスタとカウンタがあります。プリスケアラのリロードレジスタからカウンタへの転送は、タイマRA同様カウントソースに同期して値を転送しますが、タイマのリロードレジスタからカウンタへはプリスケアラのアンダフローに同期して値を転送します。

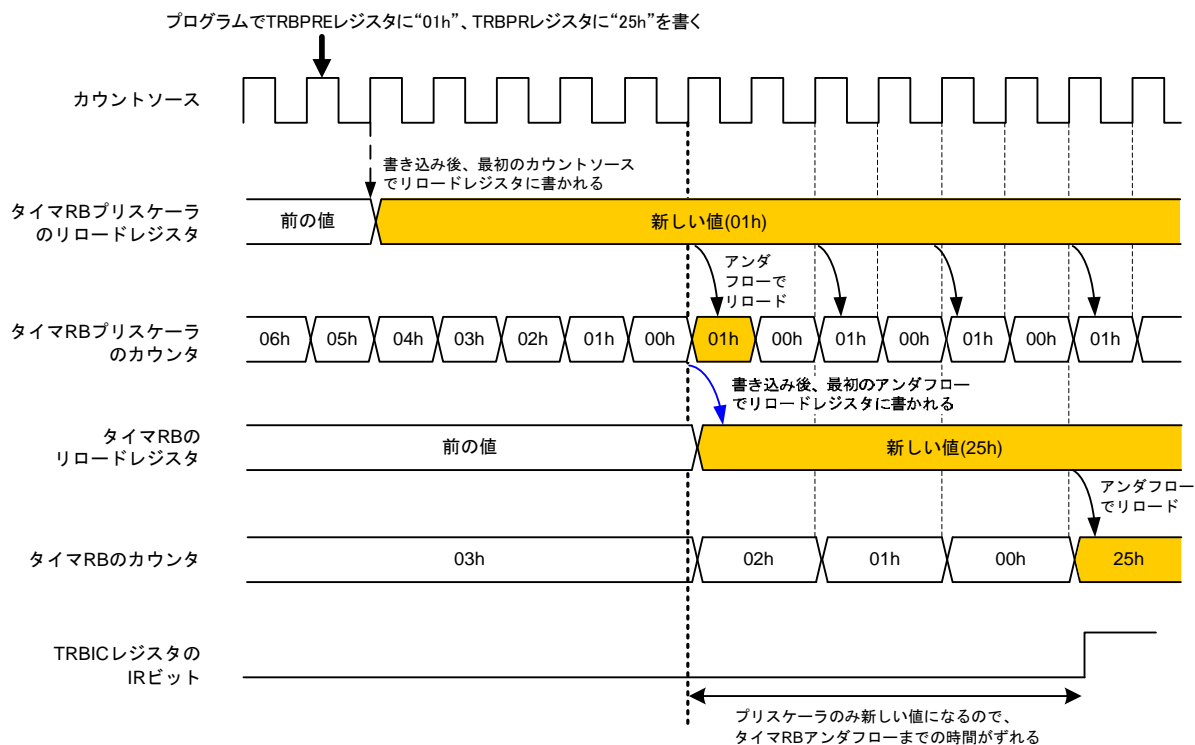
タイマRBのタイマモードでは、カウント中のプリスケアラやタイマへ書き込む場合、TRBMRレジスタのTWRCビットで、リロードレジスタとカウンタの両方へ書き込むか、リロードレジスタだけに書き込むかを選択できます。

TWRCビットで“リロードレジスタとカウンタへ書き込む”を選択している場合は、タイマRAと同様に書き込み命令実行後すぐにはカウンタの値が更新されませんので注意が必要です。

また、“リロードレジスタだけに書き込む”を選択している場合は、プリスケアラの値のみがカウントソースに同期して変更されるため、値を書き込んだときのタイマ周期がずれますので、注意してください。

カウント中のタイマへの書き込み動作(2)

TRBMRLレジスタのTWRCビットが“1” (リロードレジスタのみ書き込み)の場合



上図は次の条件の場合です。
TRBCRレジスタのTSTARTビット、TCSTFビットがともに“1” (カウント中)

Code : R8Cコース

Date : Rev. 2. 20

Page: 20 of 63

タイマモード

内部で生成されたカウントソース、またはタイマRAのアンダフロー信号をカウントし、カウンタがアンダフローするごとに割り込み要求を発生させる。

項 目	仕 様
カウントソース※	f1, f2, f8, タイマRAのアンダフロー
カウント動作	アンダフロー時リロードレジスタの内容をリロードしてカウントを継続 (タイマRBアンダフロー時は、“タイマRBプライマリリロードレジスタ”の内容をリロード)
分周比	$1/(n+1)(m+1)$ n: TRBPRESレジスタの設定値、m: TRBPRレジスタの設定値
カウント開始条件	タイマRBカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRBカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRBカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	タイマRBのアンダフロー時(タイマRB割り込み)
TRBO端子機能	プログラマブル入出力ポート
INT0端子機能	プログラマブル入出力ポート、またはINT0割り込み入力端子
タイマの読み出し動作	TRBプリスケアラレジスタ、タイマRBプライマリレジスタを読み出すと、それぞれカウンタの値が読み出される
タイマの書き込み動作	タイマ動作中にTRBPRESレジスタ、TRBRPレジスタに書き込んだ場合、リロードレジスタのみの書き込みか、リロードレジスタとカウンタの両方に書き込むかを選択できる (タイマ停止中は、リロードレジスタとカウンタの両方に書き込まれる)

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

Code : R8Cコース

Date : Rev. 2.20

Page: 21 of 63

タイマRB I/O制御レジスタ (タイマモード時)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRBIOC	アドレス 010Ah番地	リセット後の値 00h	
	ビット シンボル	ビット名	機能	RW
	TOPL	タイマRBアウトプットレベル 選択ビット	タイマモードでは“0”にしてください。	RW
	TOCNT	タイマRB出力切り替えビット		RW
	INOSTG	ワンショットトリガ制御ビット		RW
	INOSEG	ワンショットトリガ極性選択 ビット		RW
	— (b7-b4)	何も配置されていない。書く場合“0”を書いてください。読んだ場合、 その値は“0”		—

プログラマブル波形発生モード

タイマRBプライマリレジスタ(TRBPR)とタイマYセカンダリレジスタ(TRBSC)の値を交互にリロードしてカウントし、カウンタがアンダフローするごとにTRBO端子から出力する信号を反転する。カウント開始時はタイマRBプライマリレジスタに設定した値からカウント。プライマリ期間、セカンダリ期間の出力レベルを選択可能。

項 目	仕 様
カウントソース※	f1, f2, f8, タイマRAのアンダフロー
カウント動作	カウンタアンダフロー時、“プライマリリロードレジスタ”と“セカンダリリロードレジスタ”の内容を交互にリロードしてカウント(ダウンカウント)を継続
出力波形の幅と周期	幅 : プライマリ期間 : $(n+1)(m+1)/f_i$ セカンダリ期間 : $(n+1)(p+1)/f_i$ 周期 : $(n+1)\{(m+1) + (p+1)\}/f_i$ n: TRBPREレジスタ設定値、m: TRBPRレジスタ設定値、 p: TRBSCレジスタ設定値、f _i : カウントソースの周波数
カウント開始条件	タイマRBカウント開始ビットへの“1”(カウント開始)書き込み
カウント停止条件	・タイマRBカウント開始ビットへの“0”(カウント停止)書き込み ・タイマRBカウント強制停止ビットへの“1”書き込み
割り込み要求発生タイミング	セカンダリ期間のタイマRBのアンダフローからカウントソースの1/2サイクル後(タイマRB割り込み)
TRBO端子機能	プログラマブル入出力ポート、またはパルス出力
INT0端子機能	プログラマブル入出力ポート、またはINT0割り込み入力
タイマの読み出し動作	TRBPREレジスタ、TRBPRレジスタを読み出すと、それぞれカウンタの値が読み出される(注1)
タイマの書き込み動作	タイマ動作中にTRBPREレジスタ、TRBPRレジスタ、TRBSCレジスタに書き込んだ場合、それぞれリロードレジスタのみに書き込み(注2)(タイマ停止中は、リロードレジスタとカウンタの両方に書き込まれる)

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

注1. セカンダリ期間でもTRBPRレジスタを読み出してください。セカンダリ期間のカウント値が読み出せます。

注2. TRBPRレジスタへの書き込み動作により、TRBSCレジスタに設定した値が有効になります。波形の出力はTRBPRレジスタへの書き込み後、次のプライマリ期間から設定値が反映されます。

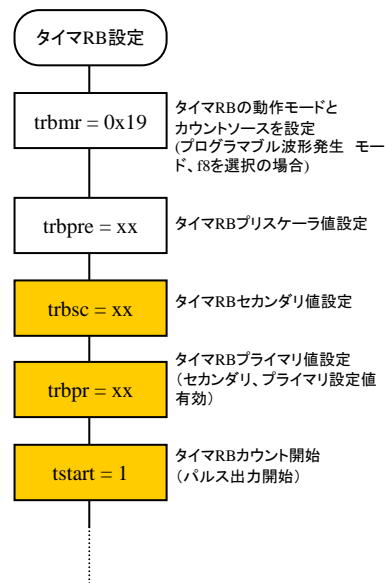
Code : R8Cコース

Date : Rev. 2.20

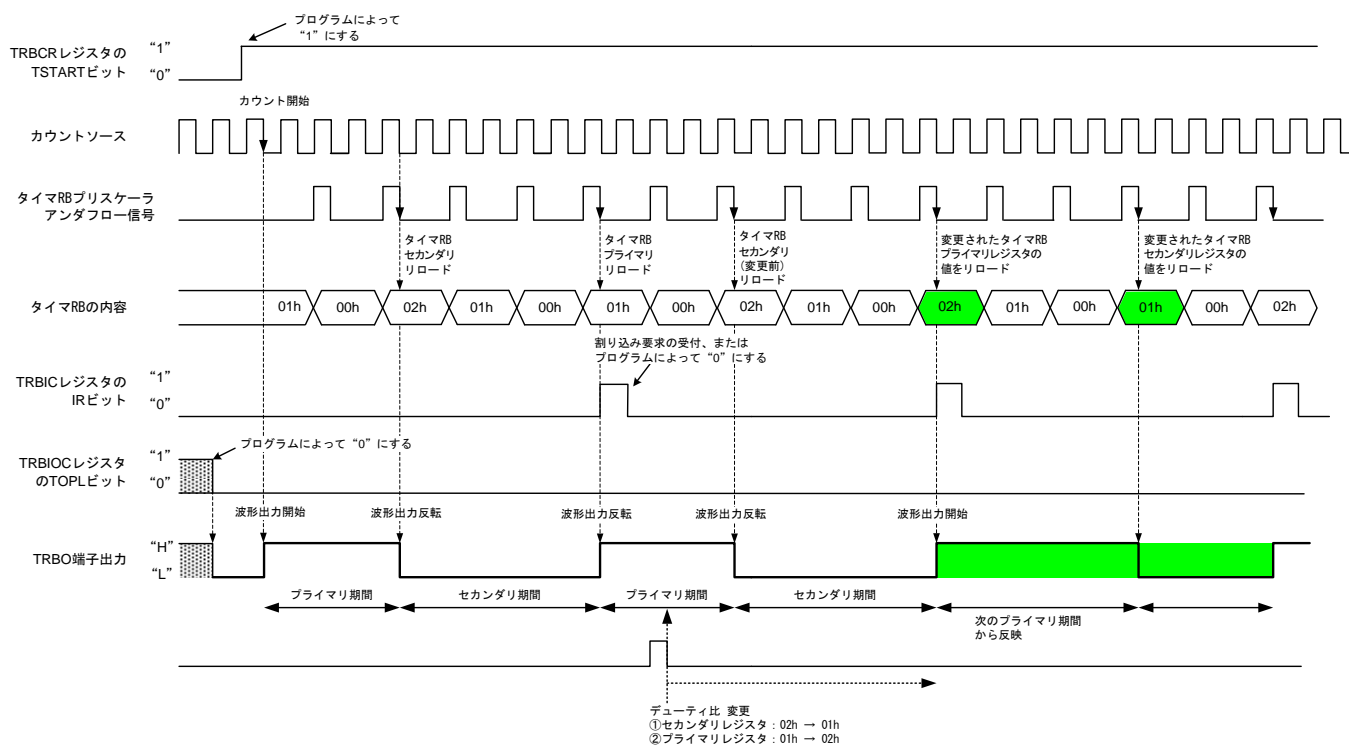
Page: 22 of 63

タイマRB I/O制御レジスタ (プログラマブル波形発生モード時)

シンボル	アドレス	リセット後の値
TRBIOC	010Ah番地	00h
ビット	ビット名	機能
TOPL	タイマRBアウトプットレベル選択ビット	0: プライマリ期間 “H” 出力、セカンダリ期間 “L” 出力 タイマ停止時 “L” 出力 1: プライマリ期間 “L” 出力、セカンダリ期間 “H” 出力 タイマ停止時 “H” 出力
TOCNT	タイマRB出力切り替え	0: タイマRB波形出力 1: P3.1ポートラッチの値を出力
INOSTG	ワンショットトリガ制御ビット	プログラマブル波形発生モードでは “0” にしてください。
INOSEG	ワンショットトリガ極性選択ビット	
— (b7-b4)	何も配置されていない。書く場合 “0” を書いてください。読んだ場合、その値は “0”	



プログラマブル波形発生モード動作例



TRBPRES = 01h、TRBPR = 01h(初期値)、TRBSC = 02h(初期値)、TRBIOCレジスタのTOCNTビット = 0(プログラマブル波形出力)の場合

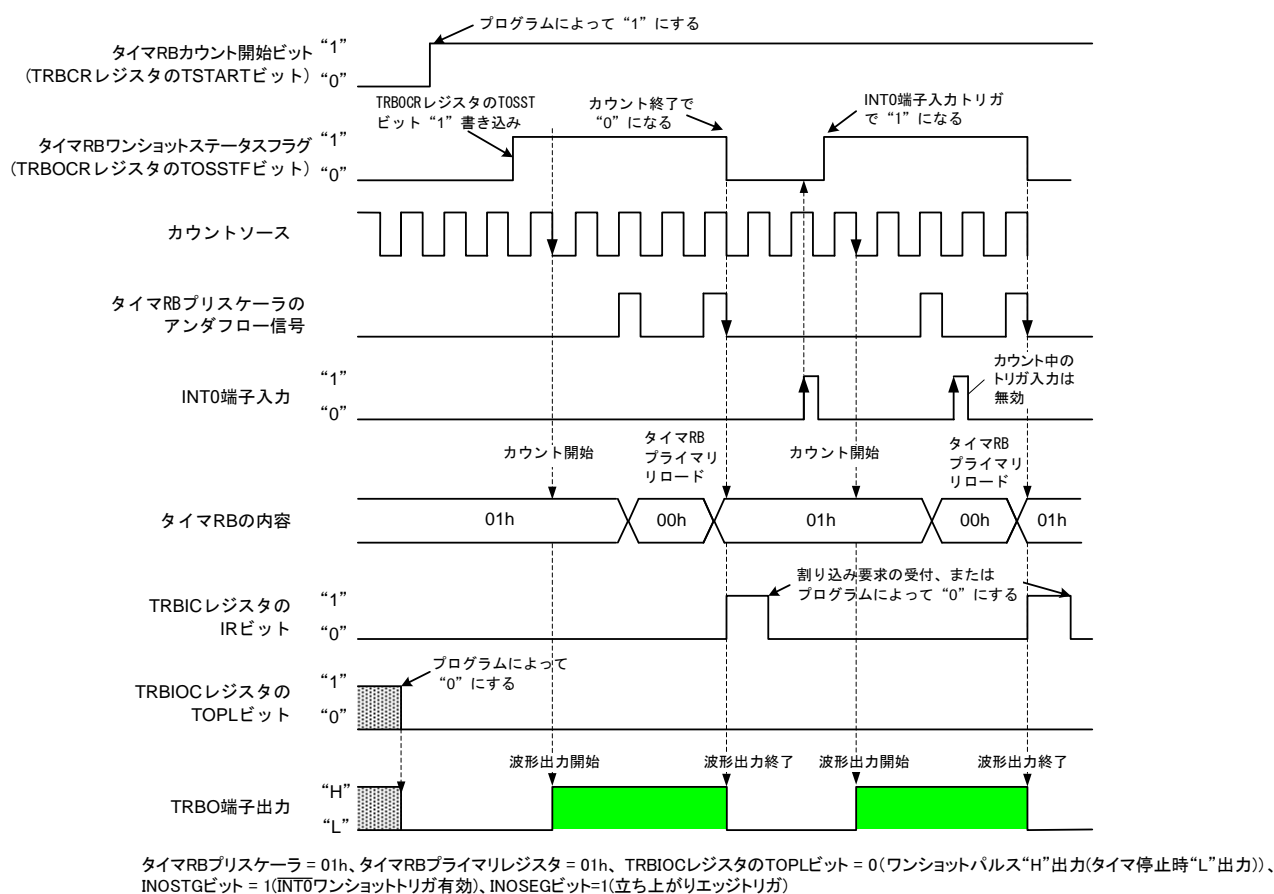
プログラマブルワンショット発生モード

トリガ(内部フラグ／外部トリガ(INT0端子入力))が発生すると、その時点から任意の時間(タイマRBプライマリレジスタ設定値)1度だけタイマが動作し、動作中パルスをTRBO端子から出力する。出力パルス波形のレベルは選択可能。

項 目	仕 様
カウントソース※	f1, f2, f8, タイマRAのアンダフロー
カウント動作	<ul style="list-style-type: none"> ・カウンタのアンダフロー時は“プライマリリロードレジスタ”の内容をリロードしてカウントを終了する(ワンショット停止) ・タイマRBカウント開始ビット(TSTARTビット)、およびタイマRBワンショット停止ビット(TOSSPビット)でカウントを停止した場合、“プライマリリロードレジスタ”の内容をリロードしてから停止する
ワンショットパルス時間	$(n+1)(m+1)/f_i$ n: TRBPRESレジスタ設定値、m: TRBPREレジスタ設定値、 f_i : カウントソース周波数
カウント開始条件	<ul style="list-style-type: none"> ●タイマRBワンショット制御レジスタの“タイマRBワンショット開始ビット”への“1”(ワンショット開始)書き込み(タイマRB制御レジスタの“タイマRBカウント開始ビット”=“1”であること) ●INT0端子への有効トリガ入力
カウント停止条件	<ul style="list-style-type: none"> ●カウンタがアンダフローし、タイマRBプライマリレジスタから値をリロードした後 ●タイマRBワンショット制御レジスタの“タイマRBワンショット停止ビット”への“1”(ワンショット停止)書き込み ●タイマRB制御レジスタの“タイマRBカウント開始ビット”への“0”(カウント停止)書き込み、あるいはタイマRBカウント強制停止ビットへの“1”(カウント停止)書き込み
割り込み要求発生タイミング	タイマRBのアンダフローからカウントソースの1/2サイクル後(タイマRB割り込み)
TRBO端子機能	パルス出力
INT0端子機能	<ul style="list-style-type: none"> ●TRBIOCレジスタの“INOSTGビット”が“0”(INT0ワンショットトリガ無効)の場合は、プログラマブル入出力ポート、またはINT0割り込み入力 ●TRBIOCレジスタの“INOSTGビット”が“1”(INT0ワンショットトリガ有効)の場合は、外部トリガ入力(INT0割り込み入力)
タイマの読み出し動作	TRBPRESレジスタ、TRBPREレジスタを読み出すと、それぞれカウンタの値が読み出される
タイマの書き込み動作	タイマ動作中にTRBPRESレジスタ、TRBPREレジスタに書き込んだ場合、それぞれリロードレジスタのみに書き込み (タイマ停止中は、リロードレジスタとカウンタの両方に書き込まれる)

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

プログラマブルワンショット動作例





Code : R8Cコース

Date : Rev. 2.20

Page: 25 of 63

タイマRB I/O制御レジスタ (プログラマブルワンショット発生モード時)

b7	b6	b5	b4	b3	b2	b1	b0
							
シンボル TRBIOC				アドレス 010Ah番地		リセット後の値 00h	
ビット シンボル		ビット名		機能		RW	
TOPL		タイマRBアウトプットレベル選択ビット		0：ワンショットパルス“H”出力、 タイマ停止時“L”出力 1：ワンショットパルス“L”出力、 タイマ停止時“H”出力		RW	
TOCNT		タイマRB出力切り替え		プログラマブルワンショット発生モードでは “0”にしてください。		RW	
INOSTG		ワンショットトリガ制御ビット		0：INT0端子ワンショットトリガ無効 1：INT0端子ワンショットトリガ有効		RW	
INOSEG		ワンショットトリガ極性選択ビット		0：立ち下がりエッジトリガ 1：立ち上がりエッジトリガ		RW	
— (b7-b4)		何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“0”				—	

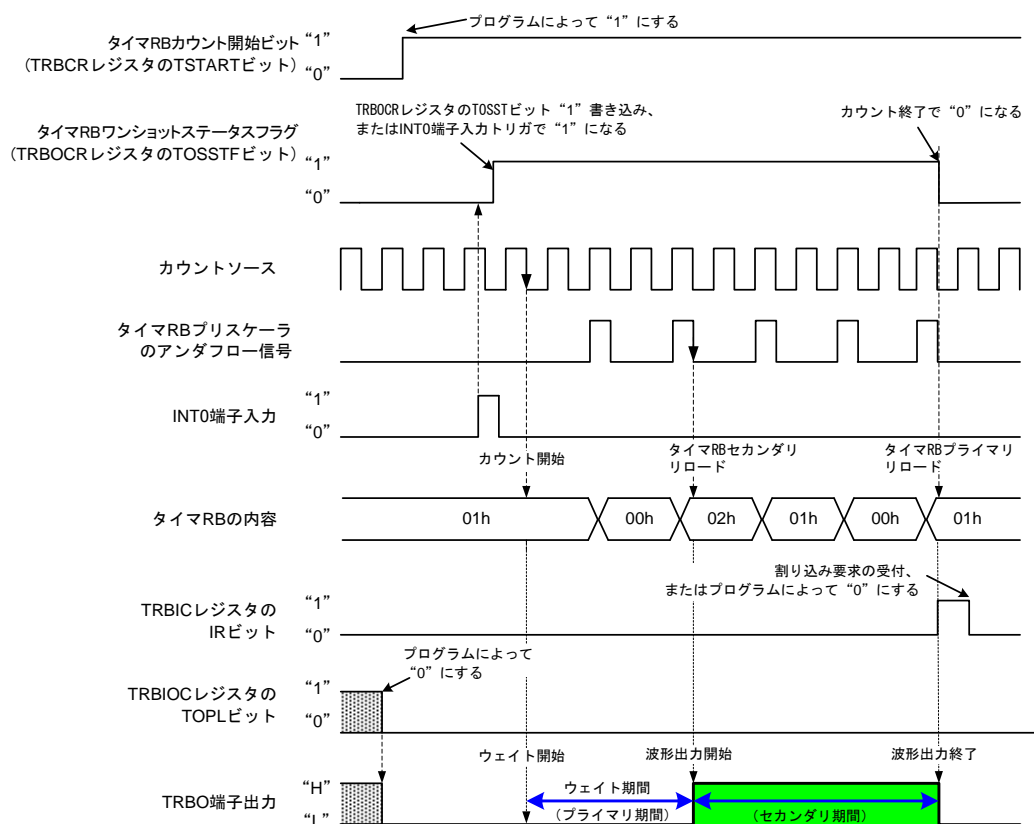
プログラマブルウェイトワンショット発生モード

トリガ(内部フラグ／外部トリガ(INT0端子入力))が発生すると、その時点から任意の時間(タイマRBプライマリレジスタ設定値)経過後、任意の時間(タイマRBセカンダリレジスタ設定値)だけパルスをTRBO端子から出力する。出力パルス波形のレベルは選択可能。

項 目	仕 様
カウントソース※	f1, f2, f8, タイマRAのアンダフロー
カウント動作	(1) タイマRBプライマリをダウンカウントし、アンダフロー後(ウェイト終了)にタイマRBセカンダリの内容をリロードしてカウントを継続(ワンショット開始) (2) タイマRBセカンダリのカウントがアンダフローすると、タイマRBプライマリの内容をリロードしてカウントを終了(ワンショット停止)、カウント停止時、リロードレジスタの内容をリロードして停止。
ウェイト時間	$(n+1)(m+1)/f_i$ n: TRBPRESレジスタ設定値、m: TRBPREレジスタ設定値、 f_i : カウントソース周波数
ワンショットパルス時間	$(n+1)(p+1)/f_i$ n: TRBPRESレジスタ設定値、p: TRBSCレジスタ設定値、 f_i : カウントソース周波数
カウント開始条件	●タイマRBワンショット制御レジスタの“タイマRBワンショット開始ビット”への“1”(ワンショット開始)書き込み(タイマRB制御レジスタの“タイマRBカウント開始ビット”=“1”であること) ●INT0端子への有効トリガ入力
カウント停止条件	●カウンタがアンダフローし、タイマRBプライマリレジスタから値をリロードした後 ●タイマRBワンショット制御レジスタの“タイマRBワンショット停止ビット”への“1”(ワンショット停止)書き込み ●タイマRB制御レジスタの“タイマRBカウント開始ビット”への“0”(カウント停止)書き込み、あるいはタイマRBカウント強制停止ビットへの“1”(カウント停止)書き込み
割り込み要求発生タイミング	タイマRBのアンダフローからカウントソースの1/2サイクル後(タイマRB割り込み)
TRBO端子機能	パルス出力
INT0端子機能	●TRBIOCレジスタの“INOSTGビット”が“0”(INT0ワンショットトリガ無効)の場合は、プログラマブル入出力ポート、またはINT0割り込み入力 ●TRBIOCレジスタの“INOSTGビット”が“1”(INT0ワンショットトリガ有効)の場合は、外部トリガ入力(INT0割り込み入力)
タイマの読み出し動作	TRBPRESレジスタ、TRBPREレジスタを読み出すと、それぞれカウンタの値が読み出される
タイマの書き込み動作	タイマ動作中にTRBPRESレジスタ、TRBPREレジスタに書き込んだ場合、それぞれリロードレジスタのみに書き込み (タイマ停止中は、リロードレジスタとカウンタの両方に書き込まれる)

※ f1, f2, f8は、XINクロックまたはオンチップオシレータクロックの1、2、8分周

プログラマブルウェイトワンショット動作例



タイマRBプリスケアラ = 01h、タイマRBプライマリレジスタ = 01h、タイマRBセカンダリレジスタ = 02h、TRBIOCレジスタのTOPLビット = 0 (ワンショットパルス "H" 出力(タイマ停止時 "L" 出力))、
INOSTGビット = 1 (INT0端子ワンショットトリガ有効)、INOSEGビット = 1 (立ち上がりエッジトリガ)

Code : R8Cコース

Date : Rev. 2.20

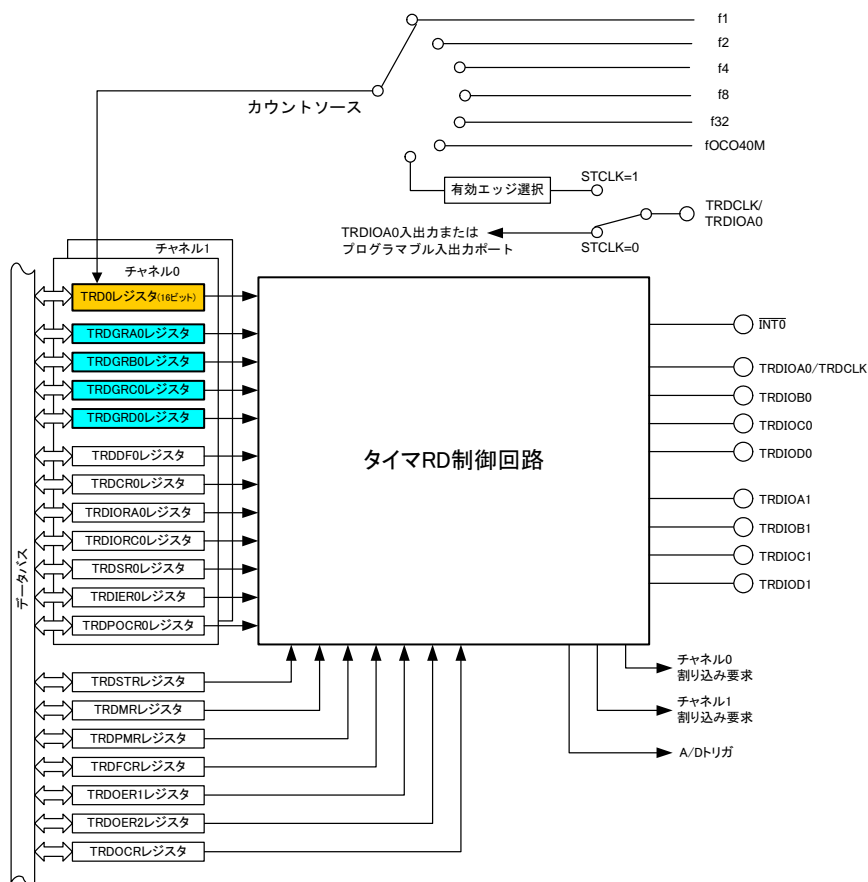
Page: 27 of 63

タイマRB I/O制御レジスタ (プログラマブルウェイトワンショット発生モード時)

b7	b6	b5	b4	b3	b2	b1	b0
X		X		X		X	
X		X		X		0	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X		X	
X		X		X</			

8.2.3 タイマRD

タイマRDの構成



Code : R8Cコース

Date : Rev. 2.20

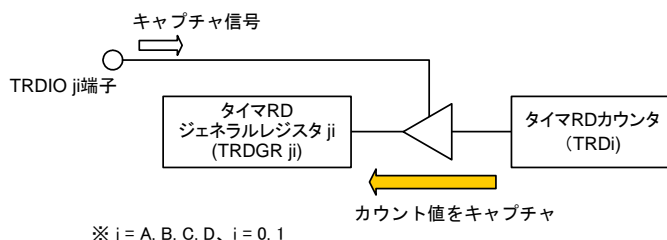
Page: 28 of 63

タイマRDは2チャンネルあり、16ビットのフリーランカウンタとインプットキャプチャ／コンペアマッチ用“ジェネラルレジスタ(各チャンネル4本づつ)”で構成される。

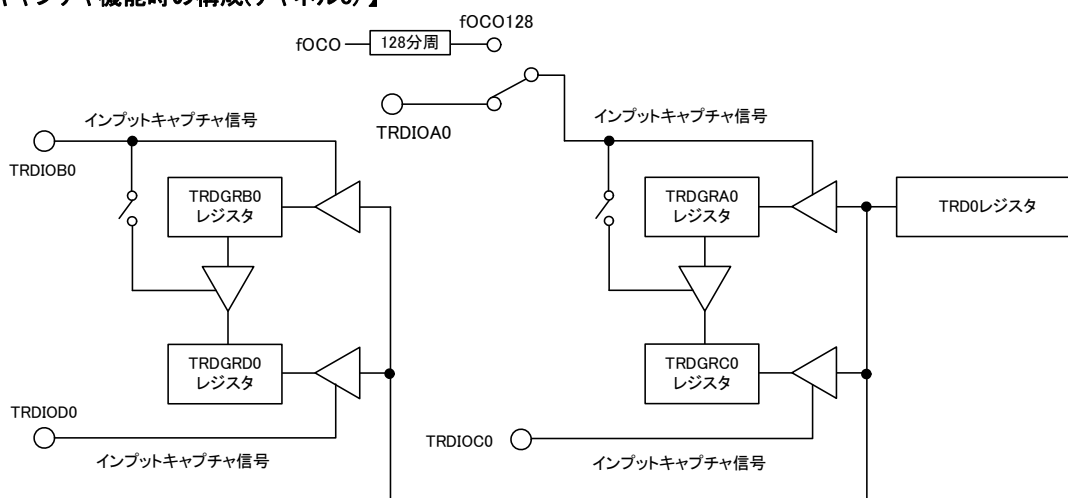
さまざまなPWM波形を出力できる高機能タイマ。

インプットキャプチャ機能

外部信号の幅や周期を測定する機能で、TRDIOj(i=j=A,B,C,D、i=0,1)端子から入力される外部信号をトリガにしてTRDi (i=0,1)レジスタ(カウンタ)の内容をジェネラルレジスタj(i=j=A,B,C,D、i=0,1)に転送する。



【インプットキャプチャ機能時の構成(チャンネル0)】

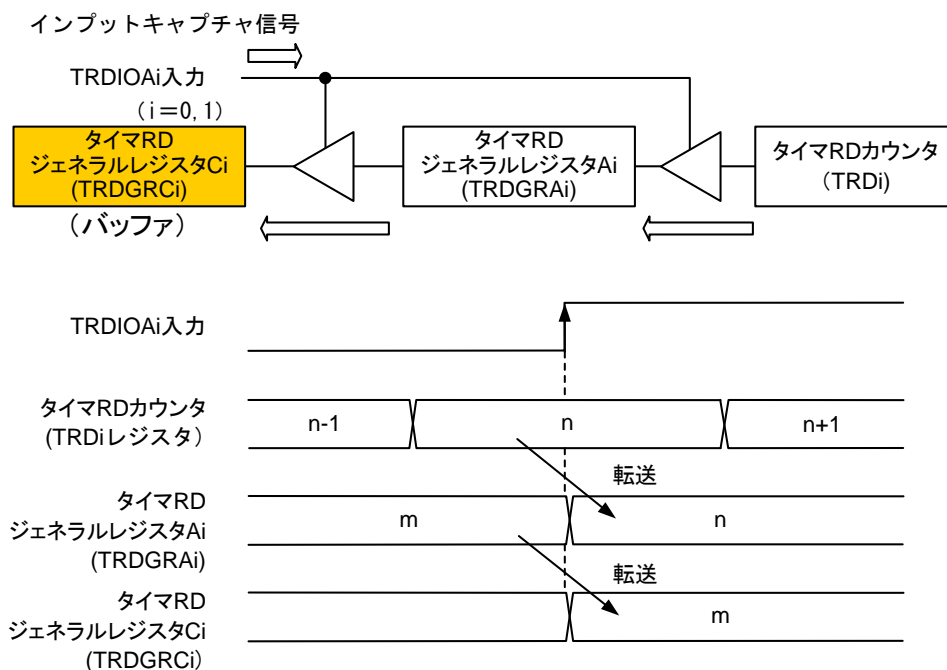


Code : R8Cコース

Date : Rev. 2. 20

Page: 29 of 63

インプットキャプチャ機能時の バッファ動作例



※ キャプチャ用レジスタをTRDGRBiとした場合は、TRDGRDiがバッファレジスタとなります。

Code : R8Cコース

Date : Rev. 2.20

Page:30 of 63

タイマRD I/O制御レジスタAi ($i=0\sim1$)

シンボル	アドレス	リセット後の値
TRDIOA0	0141h番地	10001000b
TRDIOA1	0151h番地	10001000b

ビット シンボル	ビット名	機能	RW
IOA0	TRDGRA制御ビット	b1b0 0 0: 立ち上がりエッジでTRDGRAiへインプットキャプチャ 0 1: 立ち下がりエッジでTRDGRAiへインプットキャプチャ 1 0: 両エッジでTRDGRAiへインプットキャプチャ 1 1: 設定しないでください	RW
IOA1			RW
IOA2			RW
IOA3			RW
IOB0	TRDGRB制御ビット	b5b4 0 0: 立ち上がりエッジでTRDGRBiへインプットキャプチャ 0 1: 立ち下がりエッジでTRDGRBiへインプットキャプチャ 1 0: 両エッジでTRDGRBiへインプットキャプチャ 1 1: 設定しないでください	RW
IOB1			RW
IOB2			RW
— (b7)			—
—	何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“1”。		—

注1. TRDMRレジスタのBFCiビットで“1” (TRDGRAiレジスタのバッファレジスタ) を選択した場合、TRDIOA1レジスタのIOA2ビットとTRDIOA0レジスタのIOA2ビットの設定を同じにしてください。

注2. TRDMRレジスタのBFDiビットで“1” (TRDGRBiレジスタのバッファレジスタ) を選択した場合、TRDIOA1レジスタのIOB2ビットとTRDIOA0レジスタのIOB2ビットの設定を同じにしてください。

注3. TRDIOA0レジスタのみ有効です。TRDIOA1レジスタは、“1”にしてください。

注4. IOA2ビットが“1” (インプットキャプチャ機能) のとき有効です。

インプットキャプチャ機能の仕様

項 目	仕 様
カウントソース	f1, f2, f4, f8, f32, fOCO40M TRDCLK端子に入力された外部信号(有効エッジを選択可能)
カウント動作	アップカウント
カウント開始条件	TRDSTRレジスタのTSTARTiビットへの“1”(カウント開始)書き込み
カウント停止条件	TRDSTRレジスタのTSTARTiビットへの“0”(カウント停止)書き込み
割り込み要求発生タイミング	●インプットキャプチャ信号(TRDIOji入力の有効エッジまたはfOCO128のエッジ)入力時 ●タイマRDiのオーバフロー時
TRDIOji 端子機能 (j=A,B,C,D, i=0,1)	プログラマブル入出力ポート、インプットキャプチャ入力、またはTRDCLK(外部クロック)入力 (TRDIOAi端子のみ)
INT0端子機能	プログラマブル入出力ポート、またはINT0割り込み入力
カウンタ値初期化タイミング	タイマRDオーバフロー時またはインプットキャプチャ時
タイマの読み出し動作※	タイマRDiレジスタを読み出すと、カウント値が読み出される
タイマの書き込み動作	チャンネル0とチャンネル1が独立動作の場合は、書き込んだTRDiレジスタのみに書き込み、 チャンネル0とチャンネル1が同期動作の場合は、TRDiレジスタに書き込むとTRD0レジスタ、 TRD1レジスタの両方に書き込まれる

※ タイマRDiレジスタは、必ず16bit単位でアクセスすること

Code : R8Cコース

Date : Rev. 2.20

Page:31 of 63

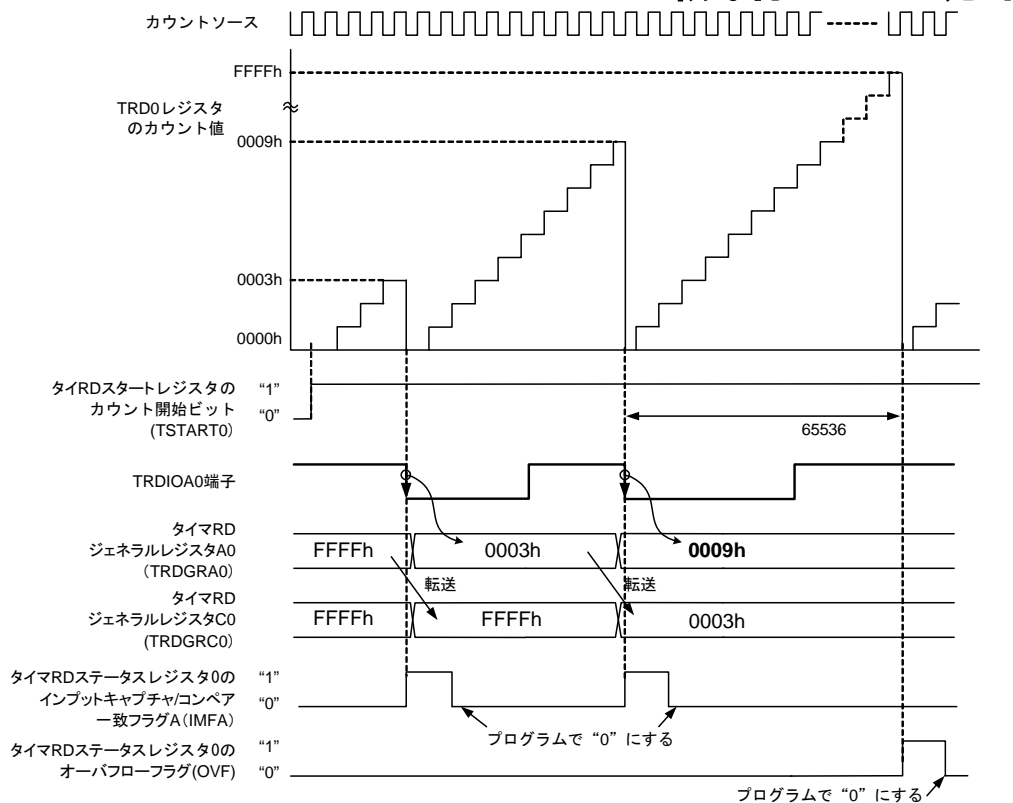
タイマRD I/O制御レジスタCi (i=0~1)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル	アドレス	リセット後の値
1 1 1 1 1 1 1 1	TRD1ORC0	0142h番地	10001000b
	TRD1ORC1	0152h番地	10001000b
ビット シンボル	ビット名	機能	RW
IOC0	TRDGRC制御ビット	b1b0 0 0 : 立ち上がりエッジでTRDGRCiへインプットキャプチャ 0 1 : 立ち下がりエッジでTRDGRCiへインプットキャプチャ 1 0 : 両エッジでTRDGRCiへインプットキャプチャ 1 1 : 設定しないでください	RW
IOC1			RW
IOC2		TRDGRCモード選択ビット(注1)	RW
IOC3		TRDGRCレジスタ機能選択ビット	RW
IOD0	TRDGRD制御ビット	b5b4 0 0 : 立ち上がりエッジでTRDGRDiへインプットキャプチャ 0 1 : 立ち下がりエッジでTRDGRDiへインプットキャプチャ 1 0 : 両エッジでTRDGRDiへインプットキャプチャ 1 1 : 設定しないでください	RW
IOD1			RW
IOD2		TRDGRDモード選択ビット(注2)	RW
IOD3		TRDGRDレジスタ機能選択ビット	RW

注1. TRDMRレジスタのBFCiビットで“1”(TRDGRAiレジスタのバッファレジスタ)を選択した場合、TRD1ORAiレジスタのIOA2ビットとTRD1ORCiレジスタのIOC2ビットの設定を同じにしてください。

注2. TRDMRレジスタのBFDiビットで“1”(TRDGRBiレジスタのバッファレジスタ)を選択した場合、TRD1ORAiレジスタのIOB2ビットとTRD1ORCiレジスタのIOD2ビットの設定を同じにしてください。

インプットキャプチャ機能の動作例



・TRDIOA0入力の立ち下がりでインプットキャプチャし、TRDGRA0のインプットキャプチャでTRD0を"0000"にクリア
 ・TRDGRC0レジスタは、TRDGRA0レジスタのバッファレジスタとして動作

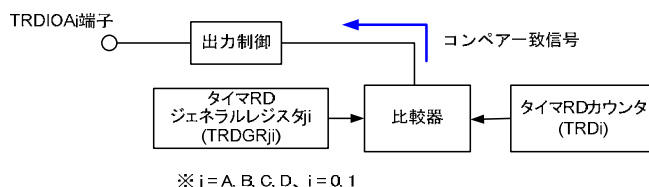
Code : R8Cコース

Date : Rev. 2. 20

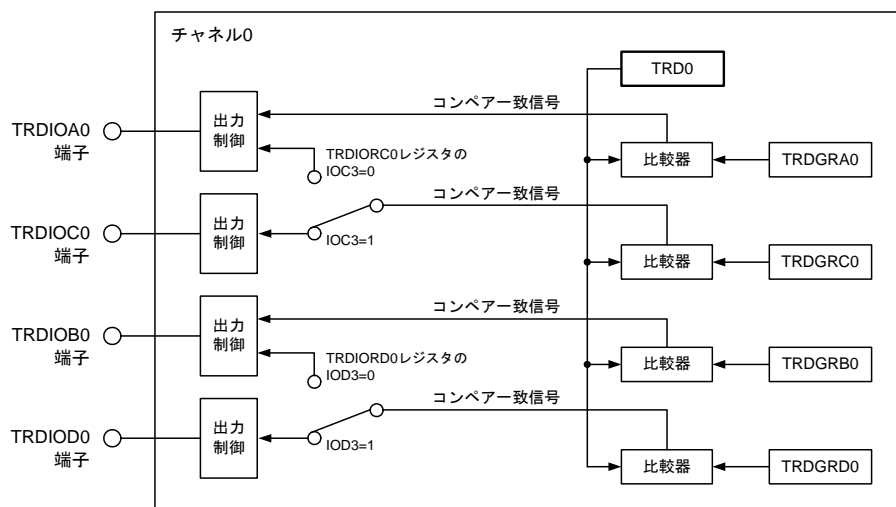
Page: 32 of 63

アウトプットコンペア機能

TRDi (i=0,1)レジスタ(カウンタ)の内容と、ジェネラルレジスタ*ji*(*j*=A,B,C,D, *i*=0,1)の内容の一致(コンペア一致)を検出する機能で、コンペア一致したときに TRDIO*ji*(*j*=A,B,C,D, *i*=0,1)端子から任意のレベルを出力する。
コンペア一致時の出力レベル、初期出力レベル(カウント開始からコンペア一致までの出力レベル)を選択可能。



【アウトプットコンペア機能時の構成(チャネル0)】

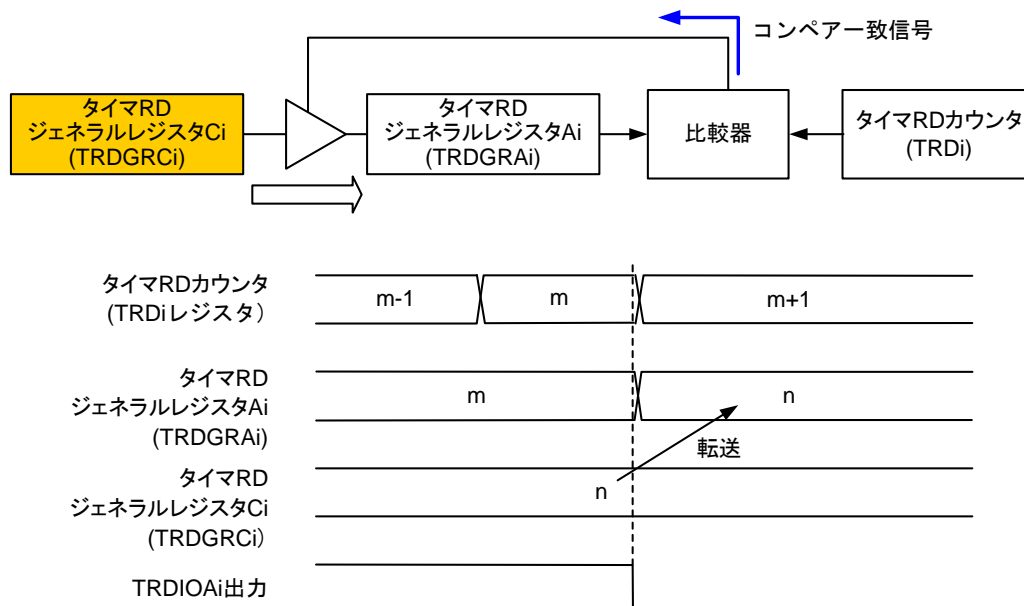


Code : R8Cヨース

Date : Rev. 2. 20

Page:33 of 63

アウトプットコンペア機能時の バッファ動作例



※ キャプチャ用レジスタをTRDGRBiとした場合は、TRDGRDiがバッファレジスタとなります。

Code : R8Cコース

Date : Rev. 2.20

Page:34 of 63

タイマRD I/O制御レジスタAi (i=0~1)

シンボル	アドレス	リセット後の値
TRDIOA0	0141h番地	10001000b
TRDIOA1	0151h番地	10001000b

ビット シンボル	ビット名	機能	RW
IOA0	TRDGRA制御ビット	b1b0 0 0 : コンペアー一致による端子出力禁止 (TRDIOAi端子はプログラマブル入出力ポート)	RW
IOA1		0 1 : TRDGRAiのコンペアー致で“L”出力 1 0 : TRDGRAiのコンペアー致で“H”出力 1 1 : TRDGRAiのコンペアー致でトグル動作	RW
IOA2		TRDGRAモード選択ビット (注1)	RW
IOA3		インプットキャプチャ入力切替ビット (注3、4)	RW
IOB0	TRDGRB制御ビット	b1b0 0 0 : コンペアー一致による端子出力禁止 (TRDIOBi端子はプログラマブル入出力ポート)	RW
IOB1		0 1 : TRDGRBiのコンペアー致で“L”出力 1 0 : TRDGRBiのコンペアー致で“H”出力 1 1 : TRDGRBiのコンペアー致でトグル動作	RW
IOB2		TRDGRBモード選択ビット (注2)	RW
— (b7)		何も配置されていない。書く場合“0”を書いてください。読んだ場合、その値は“1”。	—

注1. TRDMRレジスタのBFCiビットで“1” (TRDGRAiレジスタのバッファレジスタ) を選択した場合、TRDIOA1レジスタのIOA2ビットとTRDIORCiレジスタのIOC2ビットの設定を同じにしてください。

注2. TRDMRレジスタのBFDiビットで“1” (TRDGRBiレジスタのバッファレジスタ) を選択した場合、TRDIOA1レジスタのIOB2ビットとTRDIORCiレジスタのIOD2ビットの設定を同じにしてください。

アウトプットコンペア機能

項 目	仕 様
カウントソース	f1, f2, f4, f8, f32, fOCO40M TRDCLK端子に入力された外部信号(有効エッジを選択可能)
カウント動作	アップカウント
カウント開始条件	TRDSTRレジスタのTSTARTiビットへの“1”(カウント開始)書き込み
カウント停止条件	<ul style="list-style-type: none"> ・“コンペアー一致後TRDiレジスタのカウントを継続”としている場合は、TRDSTRレジスタのTSTARTiビットへの“0”(カウント停止)書き込み。この際、アウトプットコンペア出力端子は、カウント停止前の出力レベルを保持。 ・“コンペアー一致後TRDiレジスタのカウントを停止”としている場合は、コンペアー一致でカウントを停止。この際、アウトプットコンペア出力端子は、コンペアー一致による出力変化後のレベルを保持。
割り込み要求発生 タイミング	<ul style="list-style-type: none"> ●コンペアー一致(TRDiレジスタとTRDGRjレジスタの内容が一致)時 ●タイマRD_iのオーバフロー時
TRDIO _{ji} 端子機能 (j=A,B,C,D, i=0,1)	プログラマブル入出力ポート、アウトプットコンペア出力、またはTRDCLK(外部クロック)入力 (TRDIOAi端子のみ)
INT0端子機能	プログラマブル入出力ポート、パルス出力強制遮断信号入力またはINT0割り込み入力
カウンタ値初期化 タイミング	タイマRDオーバフローまたはTRDGRAiレジスタのコンペアー一致時
タイマの読み出し動作※	タイマRD _i レジスタを読み出すと、カウント値が読み出される
タイマの書き込み動作	チャンネル0とチャンネル1が独立動作の場合は、書き込んだTRDiレジスタのみに書き込み、 チャンネル0とチャンネル1が同期動作の場合は、TRDiレジスタに書き込むとTRD0レジスタ、TRD1レジスタの両方に書き込まれる

※ タイマRD_iレジスタは、必ず16bit単位でアクセスすること

Code : R8Cコース

Date : Rev. 2.20

Page:35 of 63

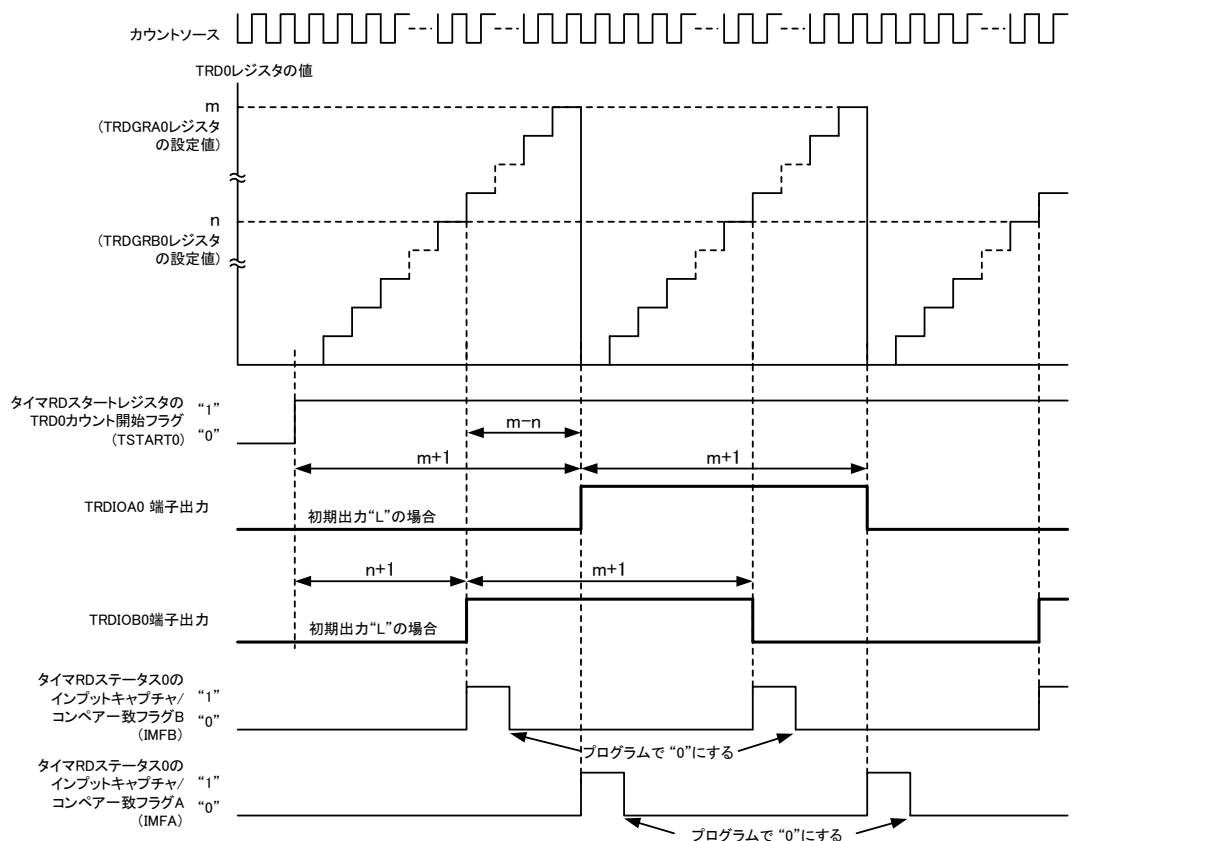
タイマRD I/O制御レジスタCi (i=0~1)

b7 b6 b5 b4 b3 b2 b1 b0				
0			0	
シンボル		アドレス	リセット後の値	
TRD1ORC0		0142h番地	10001000b	
TRD1ORC1		0152h番地	10001000b	
ビット シンボル	ビット名		機能	RW
IOC0	TRDGR <i>Ci</i> 制御ビット	b1b0 0 0 : コンペアー一致による端子出力禁止 0 1 : TRDGR <i>Ci</i> のコンペアー一致で“L”出力 1 0 : TRDGR <i>Ci</i> のコンペアー一致で“H”出力 1 1 : TRDGR <i>Ci</i> のコンペアー一致でトグル動作	RW	
IOC1		RW		
IOC2	TRDGR <i>Ci</i> モード選択ビット (注1)	アウトプットコンペア機能では“0”にしてください。	RW	
IOC3	TRDGR <i>Ci</i> レジスタ機能選択ビット	0 : TRDIOA出力レジスタとして動作 (TRDIOA <i>i</i> 端子に出力) 1 : ジェネラルレジスタまたはバッファレジスタ	RW	
IOD0	TRDGRD <i>i</i> 制御ビット	b1b0 0 0 : コンペアー一致による端子出力禁止 0 1 : TRDGRD <i>i</i> のコンペアー一致で“L”出力 1 0 : TRDGRD <i>i</i> のコンペアー一致で“H”出力 1 1 : TRDGRD <i>i</i> のコンペアー一致でトグル動作	RW	
IOD1		RW		
IOD2	TRDGRD <i>i</i> モード選択ビット (注2)	アウトプットコンペア機能では“0”にしてください。	RW	
IOD3	TRDGRD <i>i</i> レジスタ機能選択ビット	0 : TRDIOB出力レジスタとして動作 (TRDIOB <i>i</i> 端子に出力) 1 : ジェネラルレジスタまたはバッファレジスタ	RW	

注1. TRDMRレジスタのBFCiビットで“1”(TRDGRAiレジスタのバッファレジスタ)を選択した場合、TRDIOAiレジスタのIOA2ビットとTRD1ORCiレジスタのIOC2ビットの設定を同じにしてください。

注2. TRDMRレジスタのBFDiビットで“1”(TRDGRBiレジスタのバッファレジスタ)を選択した場合、TRDIOBiレジスタのIOB2ビットとTRD1ORCiレジスタのIOD2ビットの設定を同じにしてください。

アウトプットコンペア機能の動作例(1)



TRDGRA0、TRDGRB0 ともにコンペアー一致でトグル出力、TRDGRA0とのコンペアー一致でTRD0レジスタを0クリアとした場合の動作

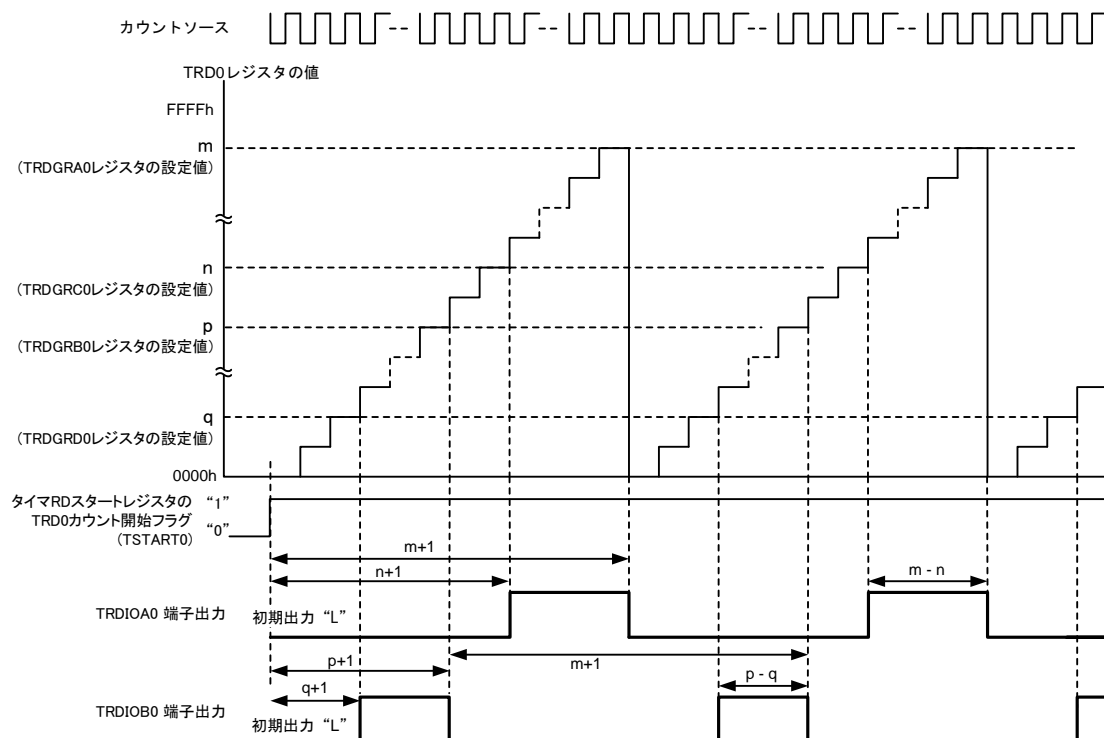
Code : R8Cコース

Date : Rev. 2. 20

Page: 36 of 63

アウトプットコンペア機能の動作例(2)

TRDGRA0とTRDGR0のコンペアー一致時の出力をTRDIOA0端子、TRDGRB0とTRDGRD0のコンペアー一致時の出力をTRDIOB0端子とすると、任意のデューティ比でパルスを出力することが可能

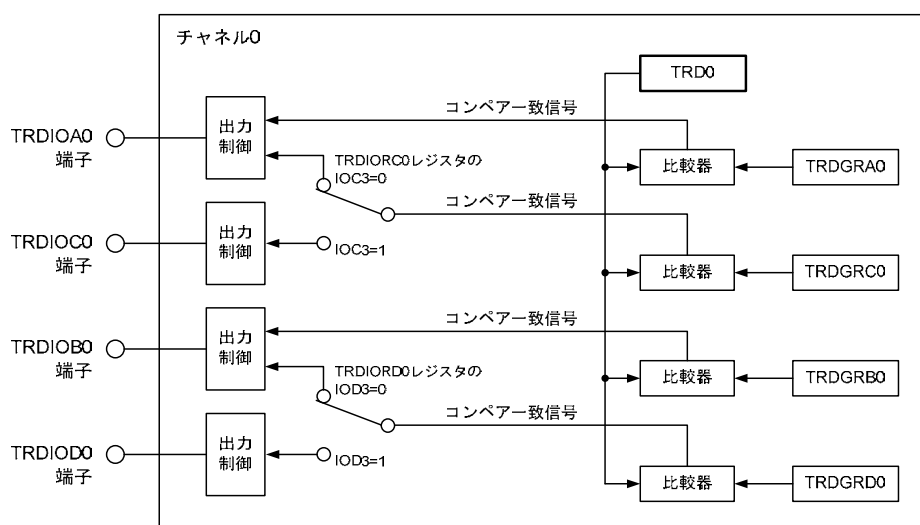


TRDGRA0、TRDGRB0、TRDGR0、TRDGRD0 全てコンペアー一致でトグル出力、TRDGRA0とのコンペアー一致でTRD0を0クリアとした場合の動作

Code : R8Cコース

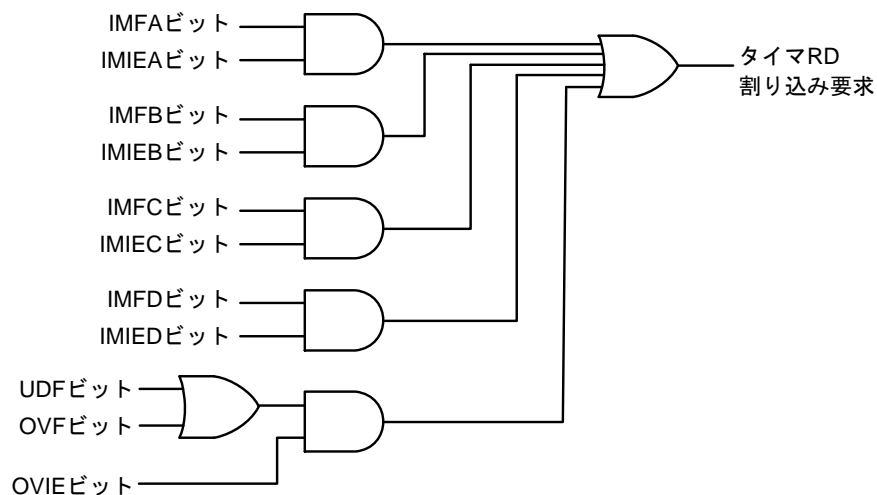
Date : Rev. 2.20

Page: 37 of 63



タイマRDの割り込み要因

チャンネル*i* (*i* = 0~1)



IMFA : インพุットキャプチャ/コンペアー致フラグA
 IMFB : インพุットキャプチャ/コンペアー致フラグB
 IMFC : インพุットキャプチャ/コンペアー致フラグC
 IMFD : インพุットキャプチャ/コンペアー致フラグD
 OVF : オーバフローフラグ
 UDF : アンダフローフラグ

IMIEA : インพุットキャプチャ/コンペアー致割り込み許可ビットA
 IMIEB : インพุットキャプチャ/コンペアー致割り込み許可ビットB
 IMIEC : インพุットキャプチャ/コンペアー致割り込み許可ビットC
 IMIED : インพุットキャプチャ/コンペアー致割り込み許可ビットD
 OVIE : オーバフロー/アンダフロー割り込み許可ビット

Code : R8Cコース

Date : Rev. 2.20

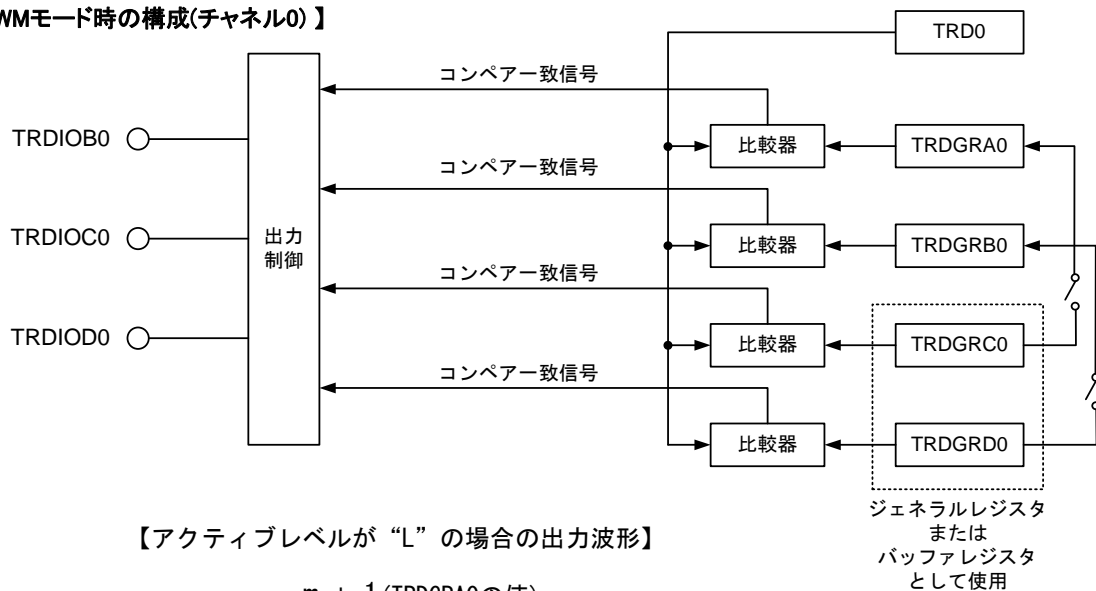
Page:38 of 63

タイマRDは割り込み要求要因が複数ある。よって上図に示すように、IMFA~IMFD、UDF、OVFの何れか一つの割り込み要因の発生で“タイマRD割り込み要求ビット”がセットされる。

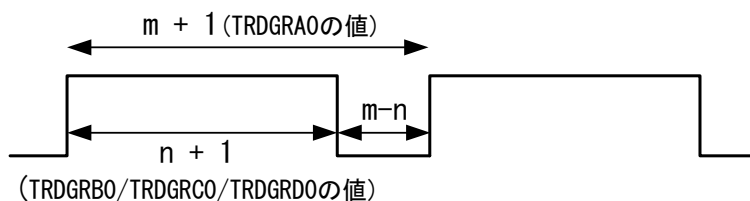
PWMモード

1チャンネルで最大3本までのPWM波形を出力できるモード。TRDGRAiレジスタの値をPWM波形の周期設定用として使用し、TRDGRBi、TRDGRCi、TRDGRDi がそれぞれPWMのデューティ設定用レジスタとなる。

【PWMモード時の構成(チャンネル0)】

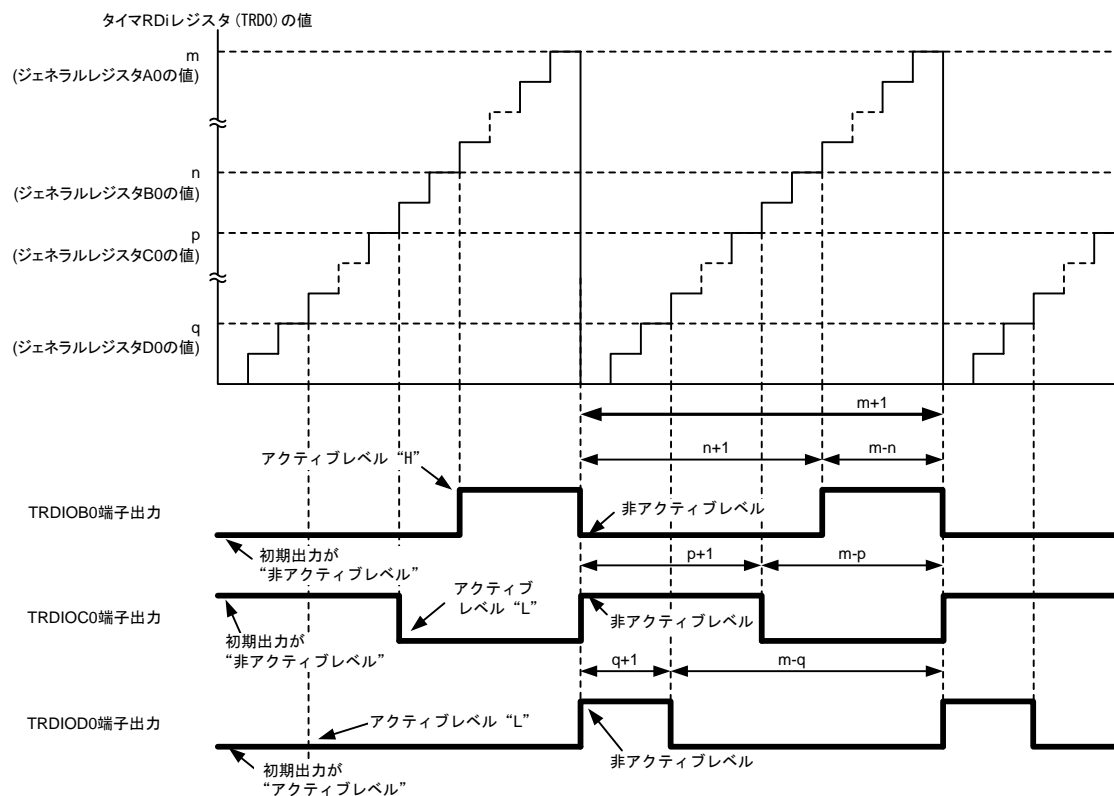


【アクティブレベルが“L”の場合の出力波形】



PWMモードの動作例

TRDGRB0のアクティブ出力レベルを“H”、TRDGRC0のアクティブ出力レベルを“L”、TRDGRD0のアクティブ出力レベルを“L”とした場合の出力波形



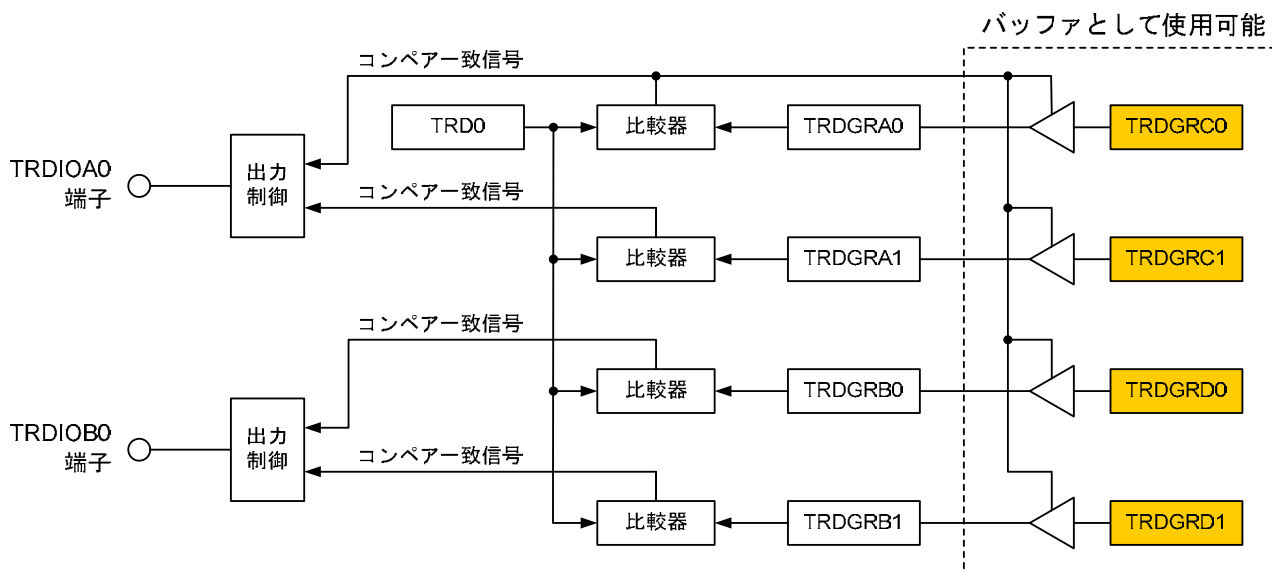
Code : R8Cコース

Date : Rev. 2.20

Page: 40 of 63

PWM3モード

PWM3モードは同一周期のPWM波形を2本出力するモードで、チャンネル0とチャンネル1の両方を使用する。ジェネラルレジスタA0とA1のコンペア値の2点、およびジェネラルレジスタB0とB1のコンペア値の2点で制御するので、バッファレジスタとしてジェネラルレジスタC0とC1、ジェネラルレジスタD0とD1を使用できる。

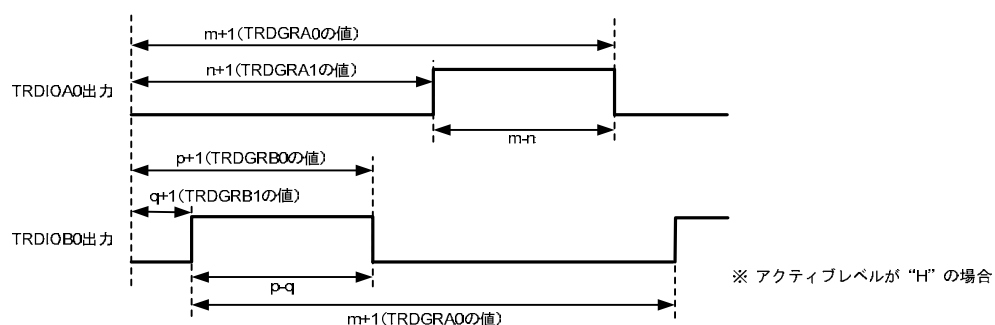


Code : R8Cコース

Date : Rev. 2. 20

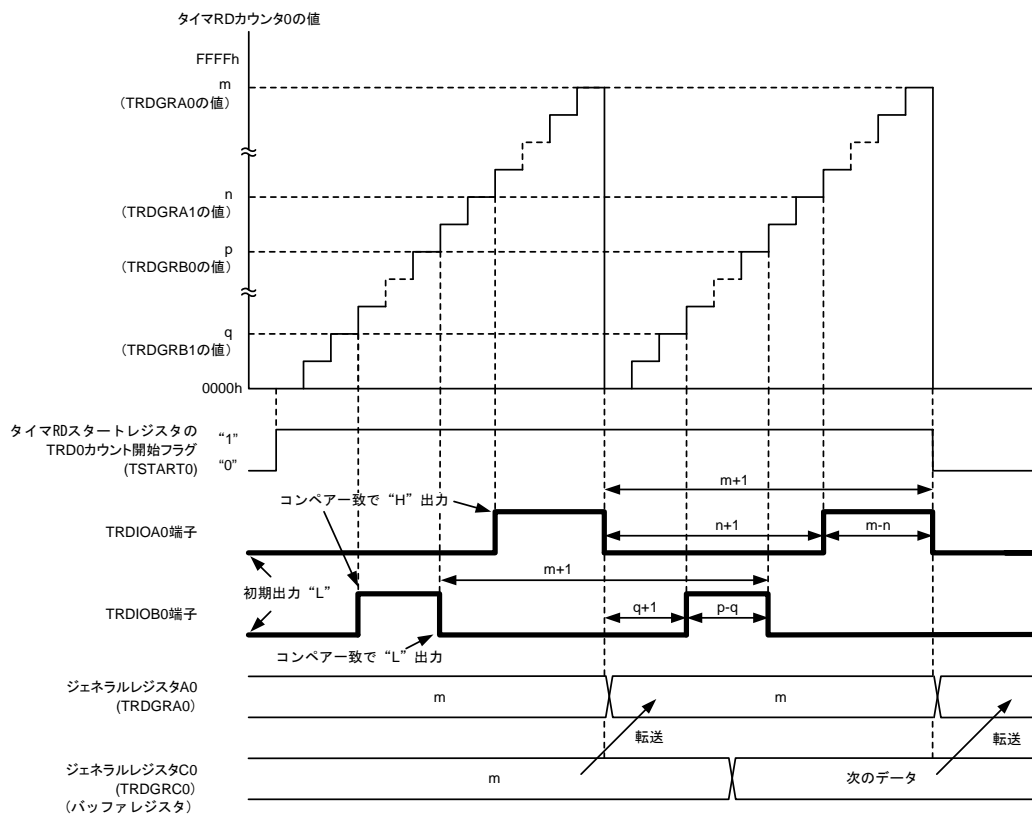
Page: 41 of 63

<出力するPWM波形>



PWM3モードの動作

初期出力“L”、TRDGRj1レジスタのコンペアー致で“H”出力、TRDGRj0レジスタのコンペアー致で“L”出力とした場合
(TRDOCレジスタのTOA0、TOA1がともに“0”)の出力波形



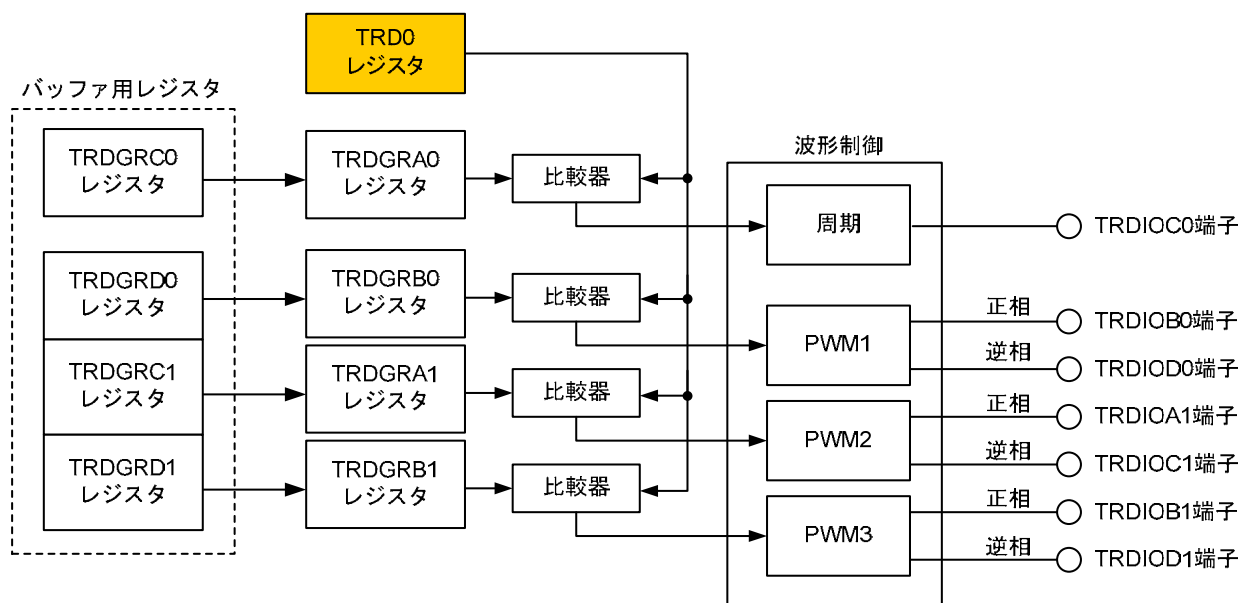
Code : R8Cコース

Date : Rev. 2. 20

Page: 42 of 63

リセット同期PWMモード

インバータや三相波制御で使えるモード。同一周期のPWM波形を正相、逆相それぞれ3本、計6本を出力。鋸波変調で短絡防止時間設定機能はなし。チャンネル0とチャンネル1のジェネラルレジスタを組み合わせ使用し、TRDGRA0レジスタの値をPWM波形の周期設定用として使用。



Code : R8Cコース

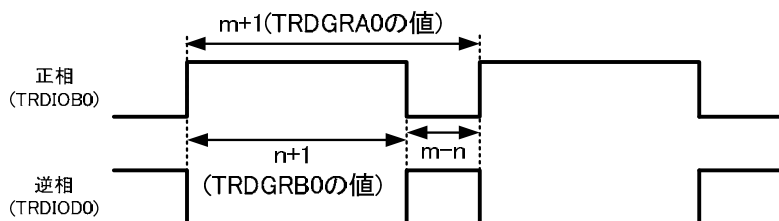
Date : Rev. 2.20

Page: 43 of 63

ジェネラルレジスタB0、A1、B1には各相の正相および逆相のデューティを決める値を設定します。

ジェネラルレジスタD0、C1、D1をバッファレジスタとして使用するかどうかは、ユーザプログラムで任意に設定。

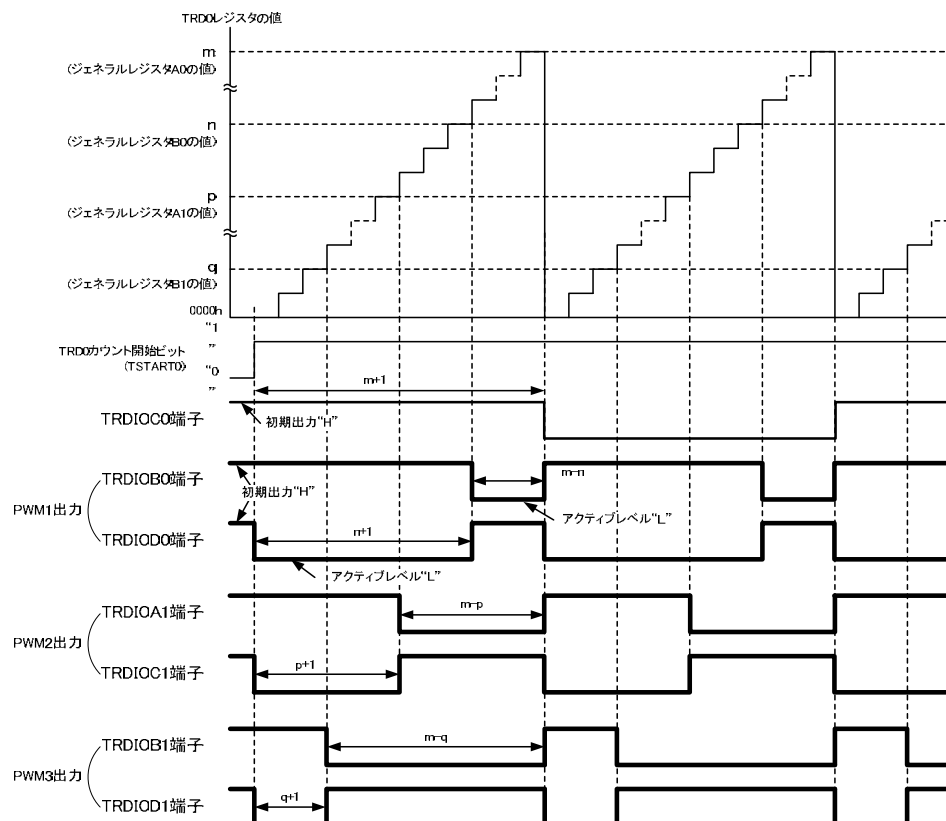
＜出力されるPWM波形＞



※ アクティブレベルが“L”の場合

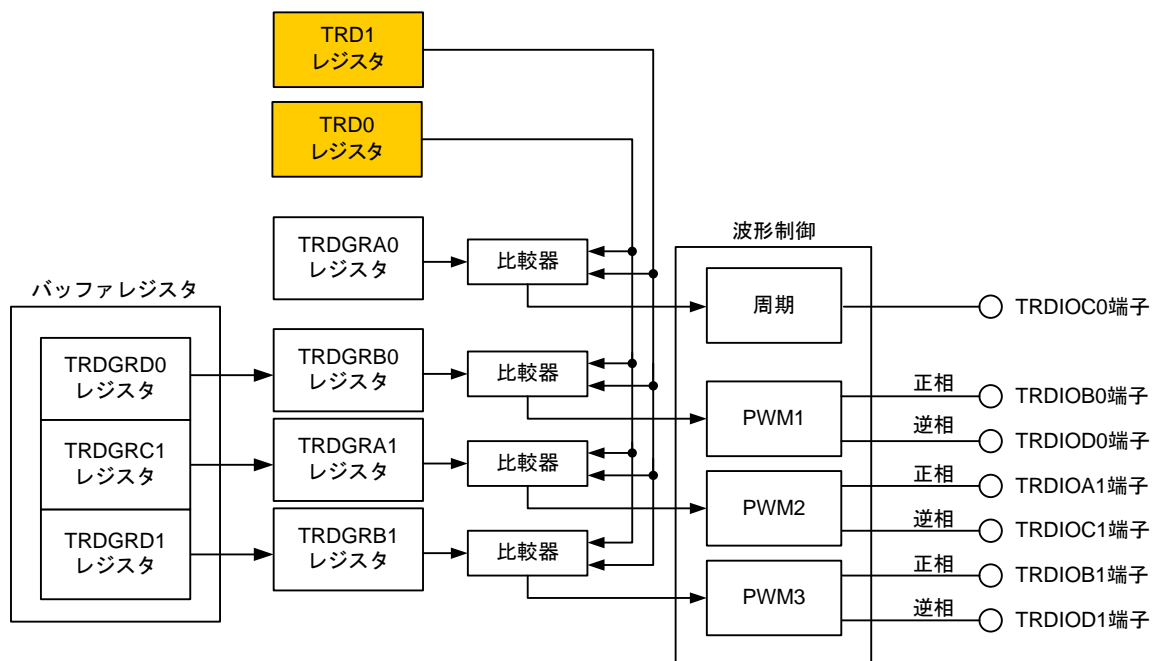
リセット同期PWMモードの動作例

正逆相とも 初期出力レベルを“H”、アクティブレベルを“L”とした場合の出力波形

Code : R8CコースDate : Rev. 2. 20Page: 44 of 63

相補PWMモード

インバータや三相波制御で使えるモード。正相、逆相それぞれ3本、計6本のPWM波形を出力。三角波変調で短絡防止時間設定機能あり。チャンネル0とチャンネル1のジェネラルレジスタを組み合わせるが、リセット同期PWMモードと異なり、タイマカウンタを2つ使用し、TRDGRD0、TRDGRC1、TRDGRD1レジスタをバッファレジスタとして占有する。



Code : R8Cコース

Date : Rev. 2. 20

Page: 45 of 63

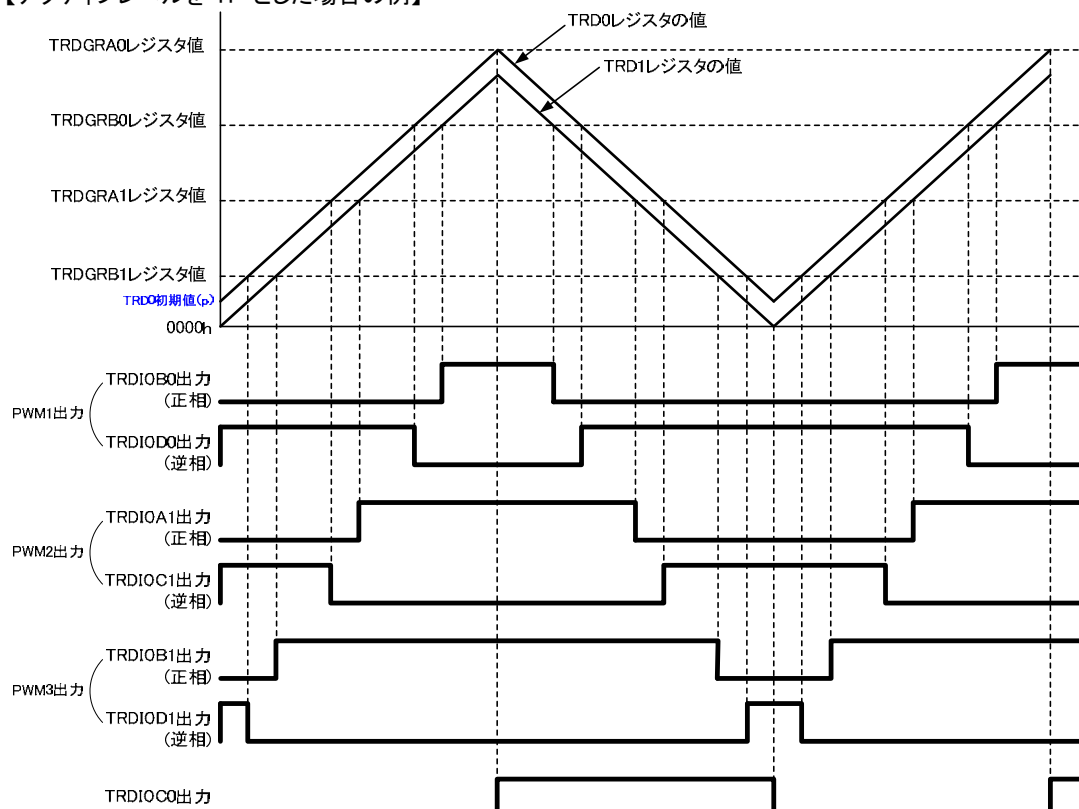
相補PWMモードでは、正相逆相の短絡防止時間の設定もできますので、インバータやブラシレスDCモータの制御に適しています。ブラシレスDCモータは、3相インバータ回路による制御が必要となるため、180° 通電の制御を行うような場合は、相補PWMモードを使うことでソフトウェア負荷を軽減することができます。

ジェネラルレジスタA0はキャリア周期の設定用レジスタとして使用し、ジェネラルレジスタB0、A1、B1には各相の正相および逆相のデューティを決める値を設定します。

タイマTRD0レジスタには、短絡防止時間を設定します。

相補PWMモードでの出力モデル

【アクティブレベルを“H”とした場合の例】

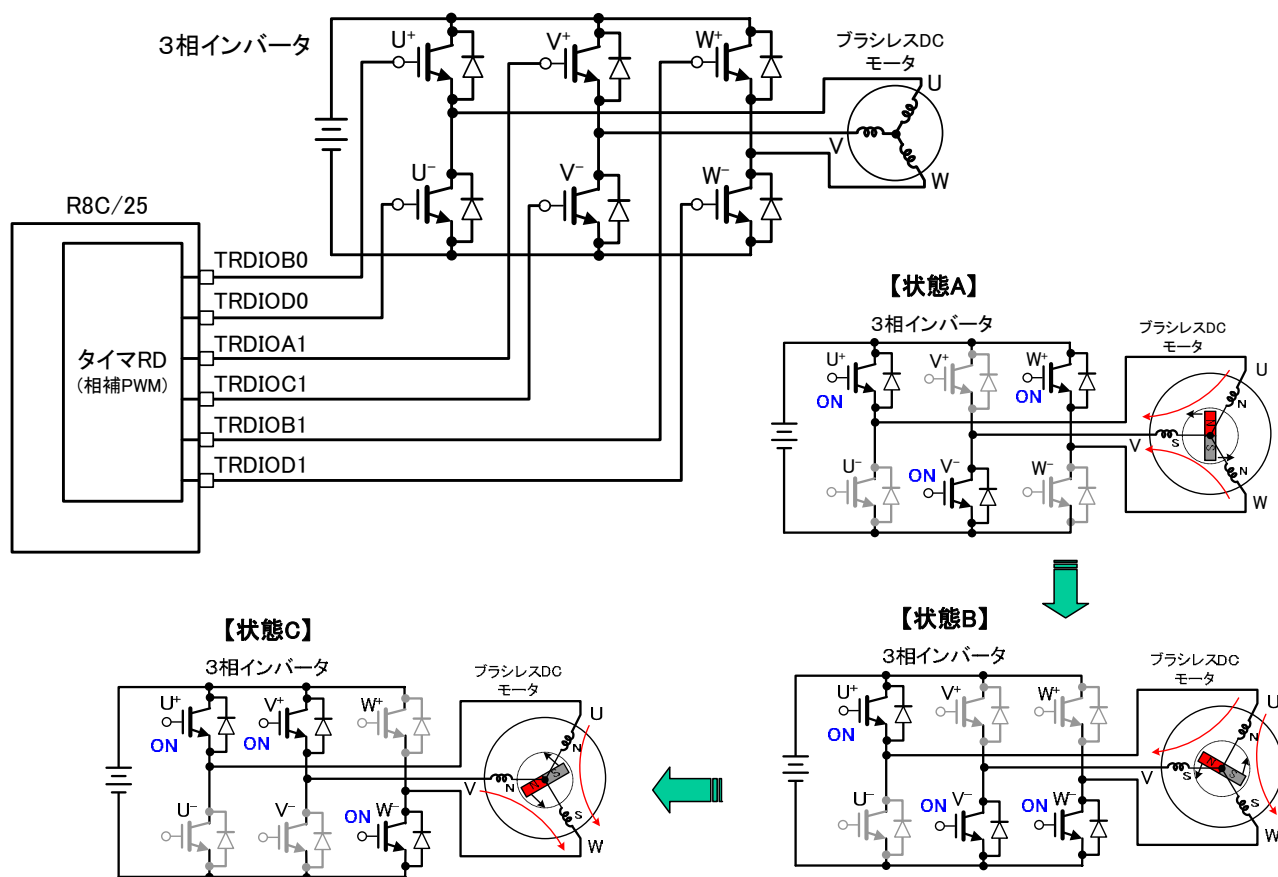


Code : R8Cコース

Date : Rev. 2.20

Page:46 of 63

三相インバータ接続例

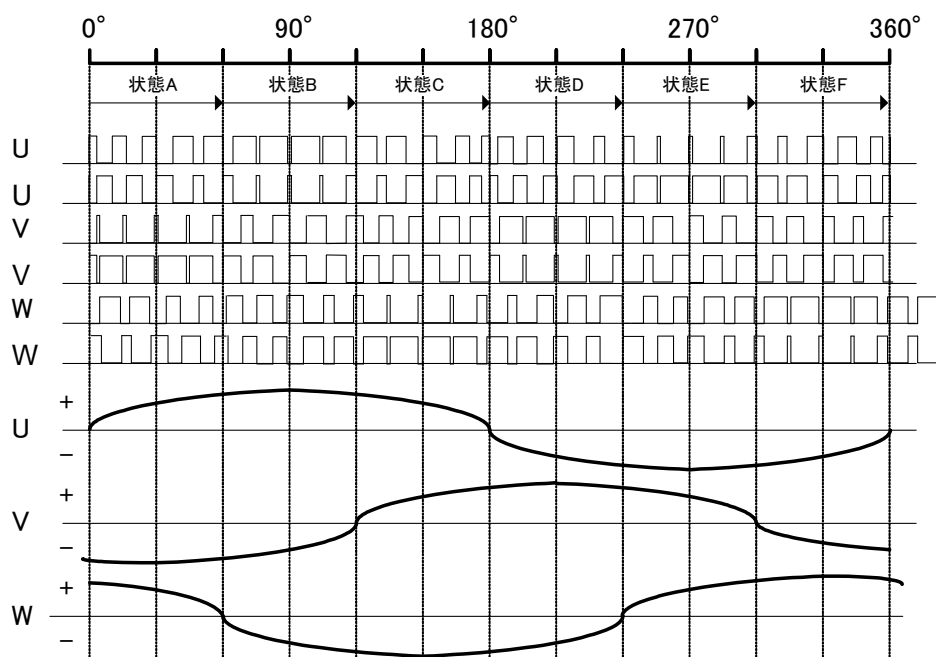


Code : R8Cコース

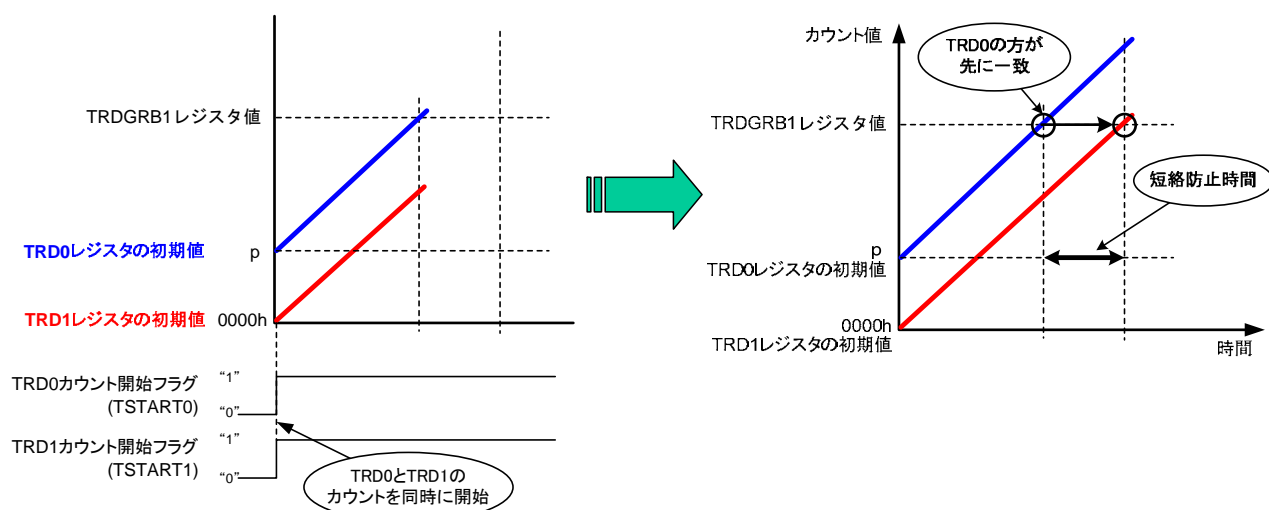
Date : Rev. 2. 20

Page: 47 of 63

<180° 通電正弦波制御の例>

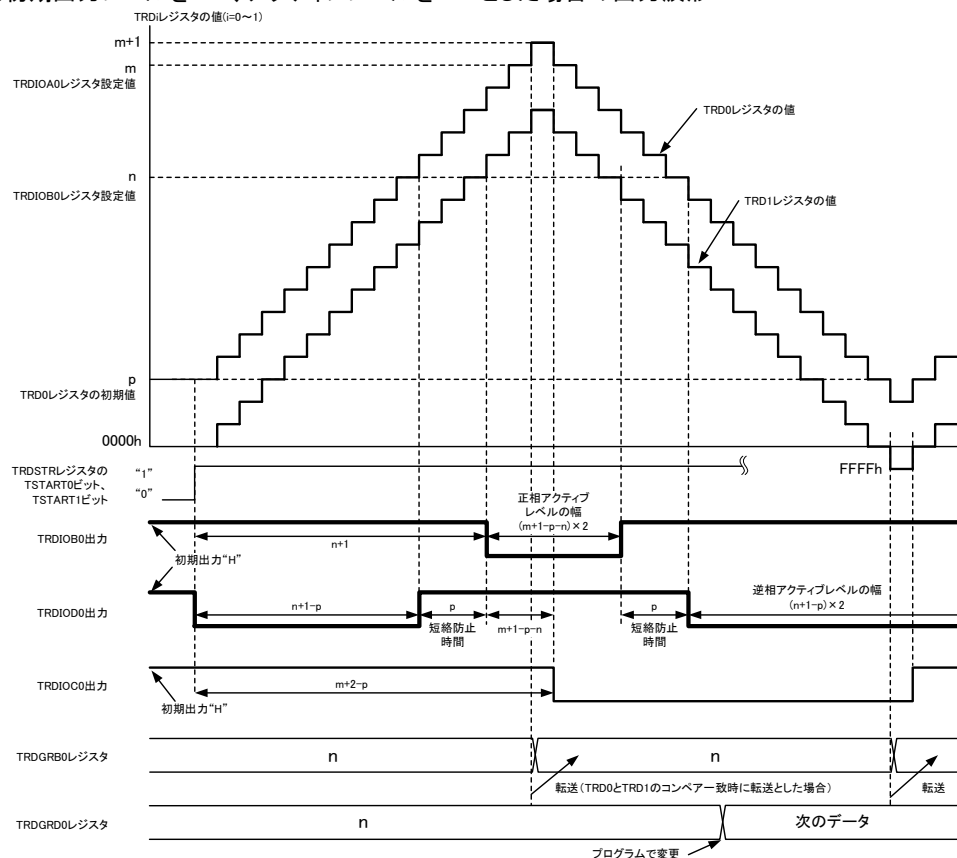


短絡防止時間の設定



相補PWMモードの動作例

正逆相とも初期出力レベルを“H”、アクティブレベルを“L”とした場合の出力波形



Code : R8Cコース

Date : Rev. 2.20

Page:49 of 63

タイマRD関連レジスタ(1)

タイマRDスタートレジスタ（注1）

シンボル TRDSTR	アドレス 0137h番地	リセット後の値 11111100b	
<div> <div> <div>b7</div><div>b6</div><div>b5</div><div>b4</div><div>b3</div><div>b2</div><div>b1</div><div>b0</div> </div> <div> <div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div> </div> </div>	ビット シンボル	ビット名	機能
TSTART0	TRD0カウント開始フラグ(注4)	0: カウント停止 (注2) 1: カウント開始	RW
TSTART1	TRD1カウント開始フラグ(注5)	0: カウント停止 (注3) 1: カウント開始	RW
CSEL0	TRD0カウント動作選択ビット	0: TRDGRA0レジスタとのコンペアー致で カウント停止 1: TRDGRA0レジスタとのコンペアー致で カウント継続	RW
CSEL1	TRD1カウント動作選択ビット	0: TRDGRA1レジスタとのコンペアー致で カウント停止 1: TRDGRA1レジスタとのコンペアー致で カウント継続	RW
— (b7-b4)	何も配置されていない。書く場合“0”を書いてください。 読んだ場合、その値は“1”。		RW

注1. TRDSTRレジスタはMOV命令を使用して書いてください（ビット処理命令を使用しないでください）。

注2. CSEL0ビットが“1”に設定されているとき、TSTART0ビットへ“0”を書いてください。

注3. CSEL1ビットが“1”に設定されているとき、TSTART1ビットへ“0”を書いてください。

注4. CSEL0ビットが“0”でコンペアー致信号（TRDIOA0）が発生したとき、“0”（カウント停止）になります。

注5. CSEL1ビットが“0”でコンペアー致信号（TRDIOA1）が発生したとき、“0”（カウント停止）になります。

タイマRDモードレジスタ

シンボル TRDMR	アドレス 0138h番地	リセット後の値 00001110b	
<div> <div> <div>b7</div><div>b6</div><div>b5</div><div>b4</div><div>b3</div><div>b2</div><div>b1</div><div>b0</div> </div> <div> <div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div><div>×</div> </div> </div>	ビット シンボル	ビット名	機能
SYNC	タイマRD同期ビット	0: TRD0とTRD1は独立動作 1: TRD0とTRD1は同期動作	RW
— (b3-b1)	何も配置されていない。書く場合“0”を書いてください。 読んだ場合、その値は“1”。		—
BFC0	TRDGR0レジスタ機能選択ビット	0: ジェネラルレジスタ 1: TRDGRA0レジスタのバッファレジスタ	RW
BFD0	TRDGRD0レジスタ機能選択ビット	0: ジェネラルレジスタ 1: TRDGRB0レジスタのバッファレジスタ	RW
BFC1	TRDGR1レジスタ機能選択ビット	0: ジェネラルレジスタ 1: TRDGRA1レジスタのバッファレジスタ	RW
BFD1	TRDGRD1レジスタ機能選択ビット	0: ジェネラルレジスタ 1: TRDGRB1レジスタのバッファレジスタ	RW

タイマRD関連レジスタ(2)

タイマRD PWMモードレジスタ

シンボル TRDPMR	アドレス 0139h番地	リセット後の値 10001000b
ビット シンボル	ビット名	機能
PWMB0	TRD10B0 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
PWMC0	TRD10C0 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
PWMD0	TRD10D0 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
— (b3)	何も配置されていない。書く場合“0”を書いてください。 読んだ場合、その値は“1”。	—
PWMB1	TRD10B1 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
PWMC1	TRD10C1 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
PWMD1	TRD10D1 PWMモード選択ビット	0 : タイマモード 1 : PWMモード
— (b7)	何も配置されていない。書く場合“0”を書いてください。 読んだ場合、その値は“1”。	—

タイマRDアウトプットマスタ許可レジスタ1

シンボル TRDOER1	アドレス 013Bh番地	リセット後の値 FFh
ビット シンボル	ビット名	機能
EA0	TRD10A0出力禁止ビット	PWMモードでは“1”(TRD10A0端子はプログラマブル入出力ポート)にしてください。
EB0	TRD10B0出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10B0端子はプログラマブル入出力ポート)
EC0	TRD10C0出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10C0端子はプログラマブル入出力ポート)
ED0	TRD10D0出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10D0端子はプログラマブル入出力ポート)
EA1	TRD10A1出力禁止ビット	PWMモードでは“1”(TRD10A1端子はプログラマブル入出力ポート)にしてください。
EB1	TRD10B1出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10B1端子はプログラマブル入出力ポート)
EC1	TRD10C1出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10C1端子はプログラマブル入出力ポート)
ED1	TRD10D1出力禁止ビット	0 : 出力許可 1 : 出力禁止 (TRD10D1端子はプログラマブル入出力ポート)

タイマRD関連レジスタ(3)

タイマRDアウトプットマスタ許可レジスタ2

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRDOER2	アドレス 013Ch番地	リセット後の値 01111111b
ビット シンボル	ビット名	機能	RW
— (b6-b0)	何も配置されていない。書く場合“0”を書いてください。 読んだ場合、その値は“1”。		—
PTO	パルス出力強制遮断信号入力 INT0有効ビット(注1)	0: パルス出力強制遮断入力無効 1: パルス出力強制遮断入力有効(INT0端子に“L”を入力すると、TRDOER1レジスタの全ビットが“1”(出力禁止)になる)	RW

タイマRDアウトプット制御レジスタ (注1)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRDOCR	アドレス 013Dh番地	リセット後の値 00h
ビット シンボル	ビット名	機能	RW
TOA0	TRDIOA0出力レベル選択ビット	PWMモードでは“0”にしてください。	RW
TOB0	TRDIOB0出力レベル選択ビット(注2)		RW
TOC0	TRDIOC0出力レベル選択ビット(注2)	0: 初期出力はアクティブでないレベル 1: 初期出力はアクティブレベル	RW
TOD0	TRDIOD0出力レベル選択ビット(注2)		RW
TOA1	TRDIOA1出力レベル選択ビット	PWMモードでは“0”にしてください。	RW
TOB1	TRDIOB1出力レベル選択ビット(注2)		RW
TOC1	TRDIOC1出力レベル選択ビット(注2)	0: 初期出力はアクティブでないレベル 1: 初期出力はアクティブレベル	RW
TOD1	TRDIOD1出力レベル選択ビット(注2)		RW

注1. TRDOCRレジスタは、TRDSTRレジスタのTSTART0、TSTART1ビットがともに“0”(カウント停止)のとき書いてください。

注2. 端子の機能が波形出力の場合、TRDOCRレジスタを設定したとき、初期出力レベルが出力されます。

タイマRD制御レジスタi (i=0~1)

b7 b6 b5 b4 b3 b2 b1 b0	シンボル TRDCR0 TRDCR1	アドレス 0140h番地 0150h番地	リセット後の値 00h 00h
ビット シンボル	ビット名	機能	RW
TCK0	カウントソース選択ビット	b2b1b0 0 0 0: f1 0 0 1: f2 0 1 0: f4 0 1 1: f8 1 0 0: f32 1 0 1: TRDCLK入力(注1) 1 1 0: fOC040M 1 1 1: 設定しないでください	RW
TCK1			RW
TCK2			RW
CKEG0		b4b3 0 0: 立ち上がりエッジでカウント 0 1: 立ち下がりエッジでカウント 1 0: 両エッジでカウント 1 1: 設定しないでください	RW
CKEG1	外部クロックエッジ選択ビット(注2)		RW
CCLR0	TRDiカウンタクリア選択ビット	PWMモードでは“001b”(TRDGRAiとのコンペアー 致でTRDiレジスタクリア)にしてください	RW
CCLR1			RW
CCLR2			RW

注1. TRDFCRレジスタのSTCLKビットが“1”(外部クロック入力有効)のとき有効です。

注2. TCK2~TCK0ビットが“101b”(TRDCLK入力)、かつTRDFCRレジスタのSTCLKビットが“1”(外部クロック入力有効)のとき有効です。

タイマRD関連レジスタ(4)

タイマRDステータスレジスタ*i* (*i*=0~1)

シンボル	アドレス		リセット後の値				
TRDSR0	0143h番地		11100000b				
TRDSR1	0153h番地		11100000b				
ビット シンボル	ビット名		機能				RW
IMFA	インプットキャプチャ/ コンペアー致フラグA		[“0” になる要因] 読んだ後 “0” を書く。(注2) [“1” になる要因] TRDiとTRDGRiの値が一致したとき。				RW
IMFB	インプットキャプチャ/ コンペアー致フラグB		[“0” になる要因] 読んだ後 “0” を書く。(注2) [“1” になる要因] TRDiとTRDGRBiの値が一致したとき。				RW
IMFC	インプットキャプチャ/ コンペアー致フラグC		[“0” になる要因] 読んだ後 “0” を書く。(注2) [“1” になる要因] TRDiとTRDGRCiの値が一致したとき。(注3)				RW
IMFD	インプットキャプチャ/ コンペアー致フラグD		[“0” になる要因] 読んだ後 “0” を書く。(注2) [“1” になる要因] TRDiとTRDGRDiの値が一致したとき。(注3)				RW
OVF	オーバフローフラグ		[“0” になる要因] 読んだ後 “0” を書く。(注2) [“1” になる要因] TRDiがオーバフローしたとき。				RW
UDF	アンダフローフラグ (注1)		PWMモードでは無効です。				RW
— (b7-b6)	何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“1”。						RW

注1. TRDSR0レジスタのb5には何も配置されていません。b5に書く場合、“0”を書いてください。読んだ場合、その値は“1”です。

注2. 書き込み結果は次のようになります。

- ・読んだ結果が“1”の場合、同じビットに“0”を書くと“0”になります。
- ・読んだ結果が“0”の場合、同じビットに“0”を書いても変化しません(読んだ後で“0”から“1”に変化した場合に、“0”を書いても“1”のままです)。
- ・“1”を書いた場合は変化しません。

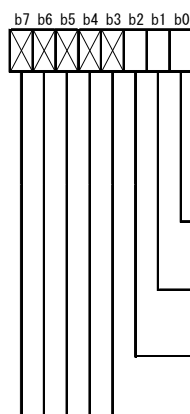
注3. TRDMRレジスタのBFjiビット(j=CまたはD)が“1”(TRDGRjiはバッファレジスタ)の場合も含む。

タイマRD割り込み許可レジスタ*i* (*i*=0~1)

シンボル	アドレス		リセット後の値				
TRDIER0	0144h番地		11100000b				
TRDIER1	0154h番地		11100000b				
ビット シンボル	ビット名		機能				RW
IMEA	インプットキャプチャ/コンペアー一致割り込み許可ビットA		0: IMFAビットによる割り込み(IMIA)禁止 1: IMFAビットによる割り込み(IMIA)許可				RW
IMEB	インプットキャプチャ/コンペアー一致割り込み許可ビットB		0: IMFBビットによる割り込み(IMIB)禁止 1: IMFBビットによる割り込み(IMIB)許可				RW
IMEC	インプットキャプチャ/コンペアー一致割り込み許可ビットC		0: IMFCビットによる割り込み(IMIC)禁止 1: IMFCビットによる割り込み(IMIC)許可				RW
IMED	インプットキャプチャ/コンペアー一致割り込み許可ビットD		0: IMFDビットによる割り込み(IMID)禁止 1: IMFDビットによる割り込み(IMID)許可				RW
OVIE	オーバフロー/アンダフロー割り込み許可ビット		0: OVFビットによる割り込み(OVI)禁止 1: OVFビットによる割り込み(OVI)許可				RW
— (b7-b5)	何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“1”。						—

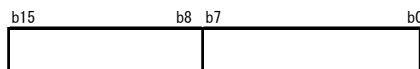
タイマRD関連レジスタ(5)

タイマRD PWMモードアウトプットレベル制御レジスタ*i* (*i*=0~1)



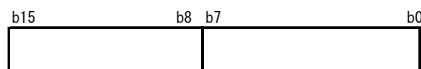
シンボル	アドレス	リセット後の値	
TRDPOCR0	0145h番地	11111000b	
TRDPOCR1	0155h番地	11111000b	
ビット シンボル	ビット名	機能	RW
POLB	PWMモードアウトプットレベル制御ビットB	0 : TRDIOBiの出力レベルは“L” アクティブ 1 : TRDIOBiの出力レベルは“H” アクティブ	RW
POLC	PWMモードアウトプットレベル制御ビットC	0 : TRDIOCiの出力レベルは“L” アクティブ 1 : TRDIOCiの出力レベルは“H” アクティブ	RW
POLD	PWMモードアウトプットレベル制御ビットD	0 : TRDIODiの出力レベルは“L” アクティブ 1 : TRDIODiの出力レベルは“H” アクティブ	RW
— (b7-b3)	何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“1”。		—

タイマRDカウンタ*i* (*i*=0~1) (注1)



シンボル	アドレス	リセット後の値
TRD0	0147h～0146h番地	00h
TRD1	0157h～0156h番地	00h
機能		設定範囲
カウントソースをカウント。カウント動作はアップカウント。オーバーフローすると、TRDSR <i>i</i> レジスタのOVFビットが“1”になる。		0000h～FFFFh

タイマRDジェネラルレジスタAi、Bi、Ci、Di (*i*=0~1) (注1)



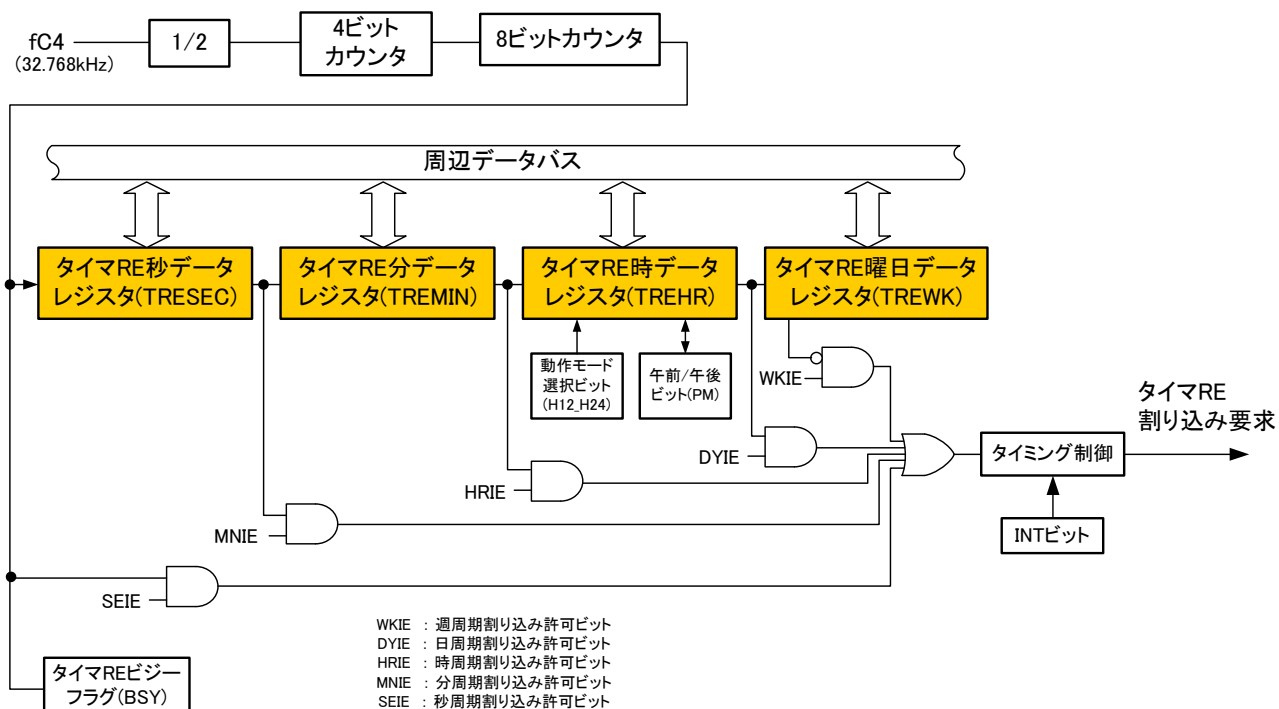
シンボル	アドレス	リセット後の値	
TRDGRA0	0149h～0148h番地	FFFFh	
TRDGRB0	014Bh～014Ah番地	FFFFh	
TRDGRD0	014Dh～014Ch番地	FFFFh	
TRDGRA1	0159h～0158h番地	FFFFh	
TRDGRB1	015Bh～015Ah番地	FFFFh	
TRDGRD1	015Dh～015Ch番地	FFFFh	
TRDGRD1	015Fh～015Eh番地	FFFFh	
機能			RW
ジェネラルレジスタまたはバッファレジスタ。TRDGRAiレジスタにはPWM周期を設定。 TRDGRBi～TRDGRDiにはPWM出力の変化点を設定する。TRDGRCiおよびTRDGRDiをバッファ レジスタとして使用する場合は、次のPWM出力の変化点を設定する。			RW

注1. TRDGRA*i*~TRDGRDiレジスタは16ビット単位でアクセスしてください。8ビット単位でアクセスしないでください。

8.2.4 タイマRE

タイマREの構成

【リアルタイムクロックモード時】



Code : R8Cコース

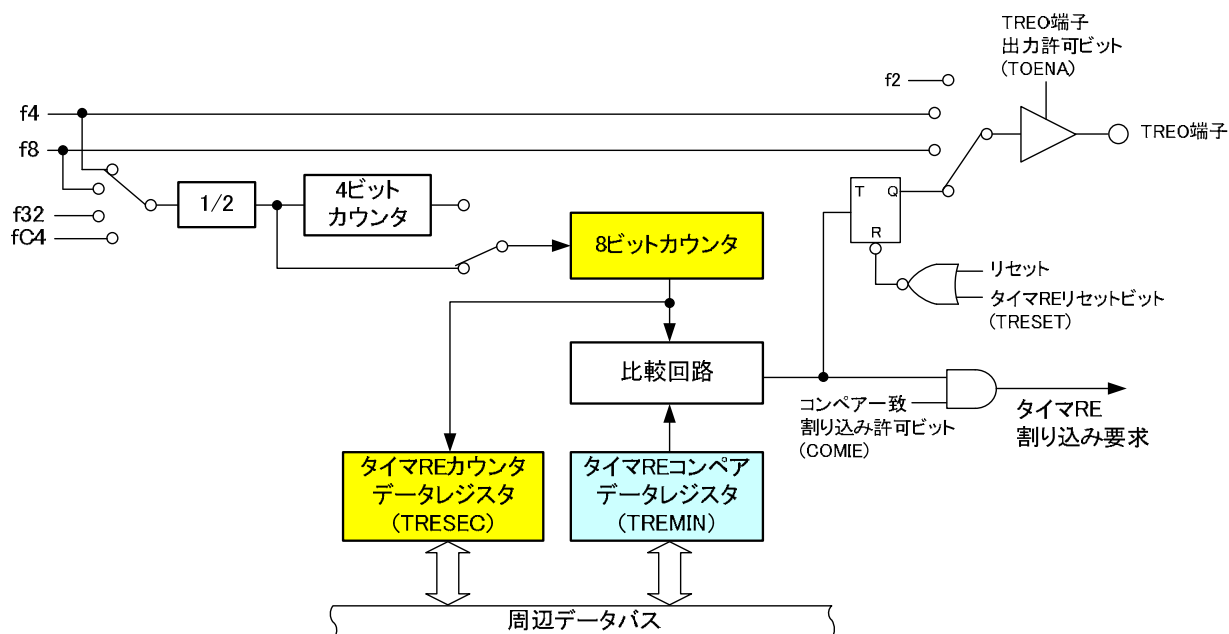
Date : Rev. 2.20

Page:55 of 63

タイマREはリアルタイムクロックとして使用できるタイマで、
ビットカウンタ、8ビットカウンタ、秒データ、分データ、時
データ、曜日データをそれぞれ格納するレジスタで構成される。

タイマREの構成

【アウトプットコンペアモード時】



Code : R8Cコース

Date : Rev. 2. 20

Page: 56 of 63

リアルタイムクロックとして使用しない場合は、アウトプットコンペアモードで通常の周期タイマとしても使用できる。

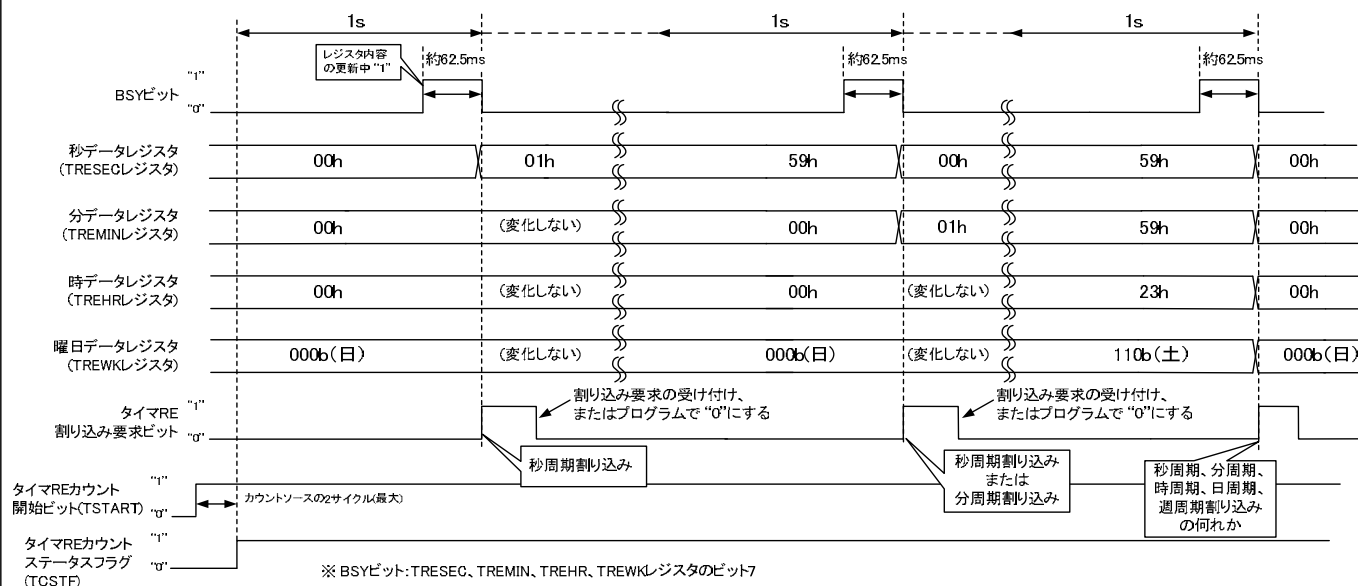
この場合、8ビットカウンタがフリーランカウンタとなり、TRESECレジスタが8ビットカウンタの読み出し用レジスタ、TREMINレジスタをコンペアマッチ用レジスタとして使用する。

リアルタイムクロックモード

項 目	仕 様
カウントソース	fc4
カウント動作	アップカウント
カウント開始条件	TRECR1レジスタのTSTARTビットへの“1”(カウント開始)書き込み
カウント停止条件	TRECR1レジスタのTSTARTビットへの“0”(カウント停止)書き込み
割り込み要求発生 タイミング	次のうちいずれか一つを選択 ・秒データの更新 ・分データの更新 ・時データの更新 ・曜日データの更新 ・曜日データが“000b(日曜日)”になったとき
TREO端子機能	プログラマブル入出力ポート、またはf2、f4、f8のいずれかを出力
タイマの読み出し動作	TRESEC、TREMIN、TREHR、TREWKレジスタを読み出すと、カウント値が読み出される。 なお TRESEC、TREMIN、TREHR レジスタの値はBCDコード。
タイマの書き込み動作	タイマ停止ときに書き込める。(TRECR1レジスタのTSTARTビットとTCSTFビットがともに“0”) なお TRESEC、TREMIN、TREHR レジスタに書き込む値はBCDコード。
選択機能	12時間モード／24時間モード切替が可能

リアルタイムクロックの動作例

【24時間モード時】



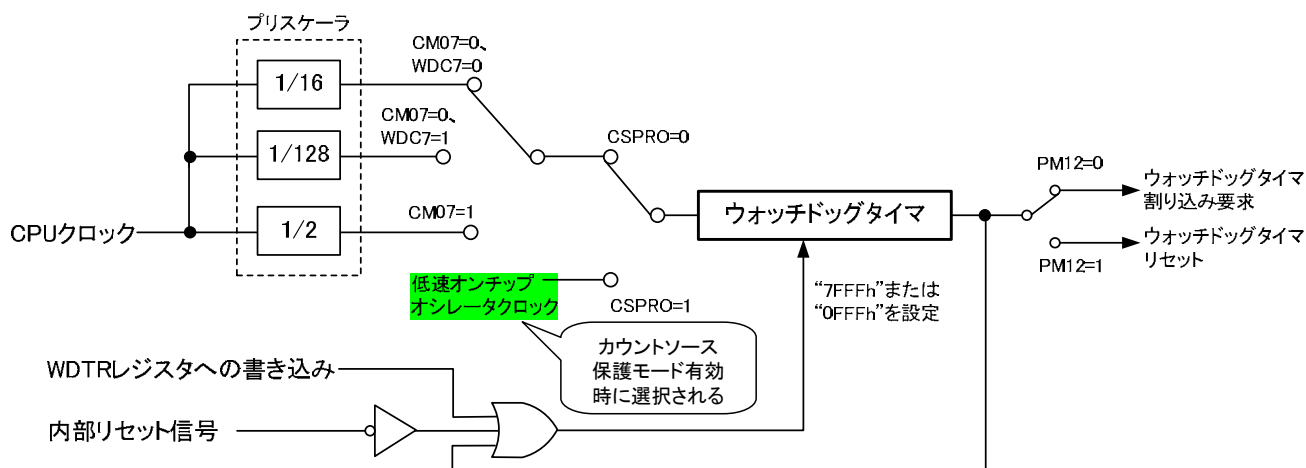
Code : R8Cコース

Date : Rev. 2. 20

Page: 58 of 63

8.3 ウォッチドッグタイマ

ウォッチドッグタイマ



【カウントソース保護モード無効時のウォッチドッグタイマの周期】

$$\text{ウォッチドッグタイマの周期} = \frac{\text{プリスケアラの分周比 (16/128/2)} \times \text{ウォッチドッグタイマのカウント値 (7FFF)}}{\text{CPUクロック}}$$

CPUクロック=20MHz、プリスケアラ分周比=16分周とした場合、ウォッチドッグタイマのアンダフロー周期は、約26.2ms

Code : R8Cコース

Date : Rev. 2.20

Page:59 of 63

ウォッチドッグタイマリセットレジスタ

シンボル	アドレス	リセット後の値	機能	
WDTR	000Dh番地	不定	このレジスタに対する書き込み命令で、ウォッチドッグタイマは初期化される。ウォッチドッグタイマの初期値は、カウントソース保護モード無効時に“7FFFh”、カウントソース保護モード有効時に“0FFFh”が設定される。	RW
				WO

ウォッチドッグタイマスタートレジスタ

シンボル	アドレス	リセット後の値	機能	
WDTS	000Eh番地	不定	このレジスタに対する書き込み命令で、ウォッチドッグタイマはスタートする。	RW
				WO

ウォッチドッグタイマ保護モードレジスタ

シンボル	アドレス	リセット後の値(注1)	機能	
CSPR	001Ch番地	0000000b		RW
ビットシンボル	ビット名			
— (b6-b0)	予約ビット	“0”にしてください。		RW
CSPRO	カウントソース保護モード選択ビット(注2)	0: カウントソース保護モード無効 1: カウントソース保護モード有効		RW

注1. OFSレジスタのCSPROINIビットに“1”を書いたとき、リセット後の値は“10000000b”になります。
 注2. CSPROビットを“1”にするためには、“0”を書いた後、続いて“1”を書いてください。

カウントソース保護モード

【カウントソース保護モードの有効／無効による違い】

項 目	仕 様	
	カウントソース保護モード無効時	カウントソース保護モード有効時
カウントソース	CPUクロック	低速オンチップオシレータ
カウント周期	$\text{プリスケアラの分周比}(n) \times \text{ウォッチドッグタイマのカウント値}(32768)$ CPUクロック n: 16または128 (WDCレジスタのWDC7ビットで選択) (例: CPUクロック20MHzで、プリスケアラ16分周の場合は、約26.2ms)	ウォッチドッグタイマのカウント値(4096) (例: 低速オンチップオシレータクロックが125kHzの場合、約32.8ms)
カウント停止条件	ストップモード、ウェイトモードへ移行 (解除後、保持されていた値からカウントを継続)	なし (カウント開始後、ウェイトモードでも停止せず、ストップモードにはならない。)
アンダフロー時の動作	・PM1レジスタのPM12ビットが“0”のときウォッチドッグタイマ割り込み ・PM1レジスタのPM12ビットが“1”のときウォッチドッグタイマリセット	ウォッチドッグタイマリセットのみ
レジスタの内容		CSPRLレジスタのCSPROビットを“1”(カウントソース保護モード有効)にすると、以下のレジスタが自動的に設定される ・ウォッチドッグタイマに0FFFFhを設定 ・CM1レジスタのCM14ビットを“0”(低速オンチップオシレータを発振) ・PM1レジスタのPM12ビットを“1”(ウォッチドッグタイマのアンダフロー時、ウォッチドッグタイマリセット) またカウントソース保護モードでは、以下の状態になる ・CM1レジスタのCM10ビットへの書き込み禁止 (ストップモードに移行しない) ・CM1レジスタのCM14ビットへの書き込み禁止 (低速オンチップオシレータは停止しない)

オプション機能選択レジスタにより、リセット解除後のウォッチドッグタイマの動作を変更可能。

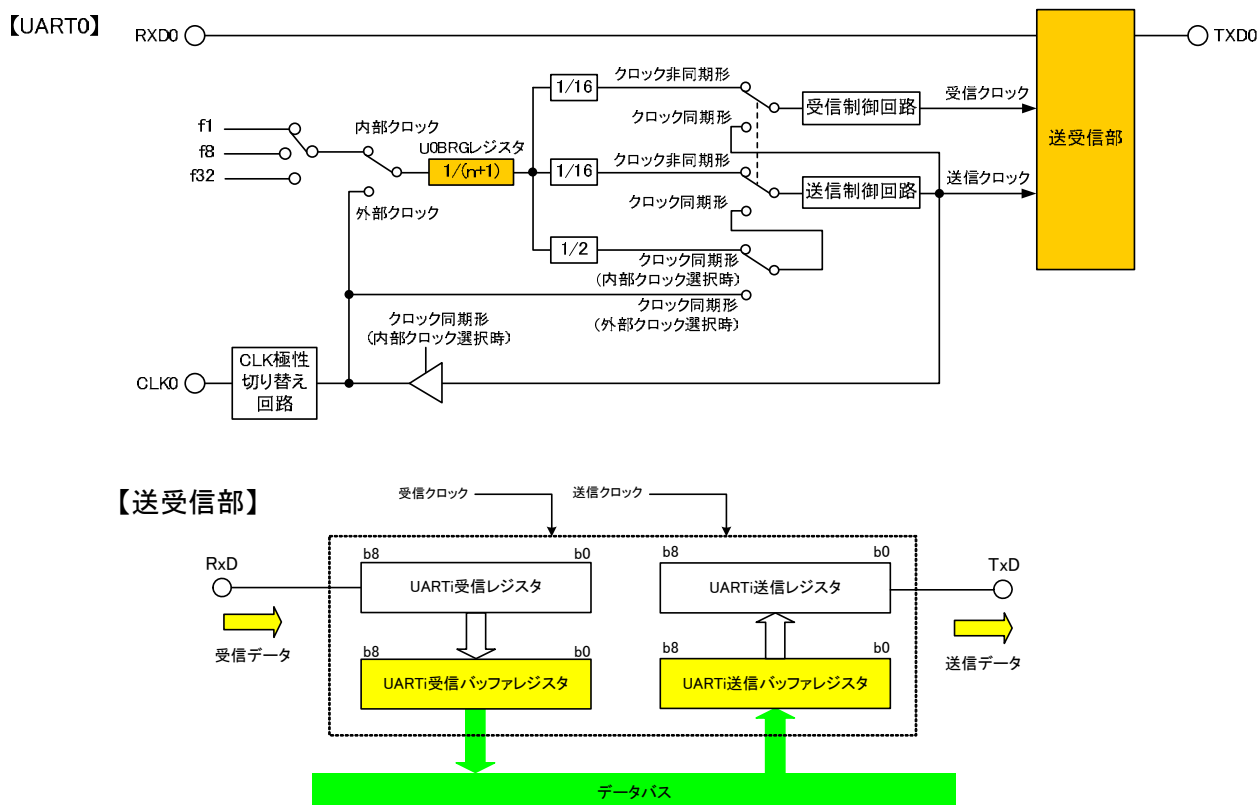
オプション機能選択レジスタ (注1)



ビット名/ビットシンボル	機能	R	W
ウォッチドッグタイマ起動選択ビット/WDTON	0: リセット後、ウォッチドッグタイマは自動的に起動 1: リセット後、ウォッチドッグタイマは停止状態	○	○
予約ビット	“1”にしてください	○	○
ROMコードプロテクト解除ビット/ROMCR	0: ROMコードプロテクト解除 1: ROMCP1有効	○	○
ROMコードプロテクトビット/ROMCP1	0: ROMコードプロテクト有効 1: ROMコードプロテクト解除	○	○
予約ビット	“1”にしてください	○	○
電圧検出0回路起動ビット (注2) /LVDOON	0: リセット後、電圧監視0リセット有効 1: リセット後、電圧監視0リセット無効	○	○
予約ビット	“1”にしてください	○	○
リセット後カウントソース保護モード選択ビット/CSPROINI	0: リセット後、カウントソース保護モード有効 1: リセット後、カウントソース保護モード無効	○	○

8.4 シリアルインタフェース

シリアルインタフェースの構成



Code : R8Cコース

Date : Rev. 2.20

Page: 61 of 63

※ UART1の入出力端子は、リセット解除後ポートとして動作しています。
TXD1、RXD1端子として使用する場合は、ポートモードレジスタの“U1PINSELビット”を
“1”にセットする必要があります。
またUART1機能選択レジスタに0Fhを書き込んでください。

ポートモードレジスタ

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	0

PMR 00F8h番地 リセット後の値: 00h

ビットシンボル	ビット名	機能	R	W
— (b3-b0)	予約ビット	“0”にしてください	—	—
U1PINSEL	ポートCLK1/TXD1/RXD1切り替えビット	0 : 入力ポートP6_5、P6_6、P6_7 1 : CLK1、TXD1、RXD1	○	○
— (b6-b5)	予約ビット	“0”にしてください	—	—
U1CSEL	SSU/I2Cバス切り替えビット	0 : SSU機能を選択 1 : I2Cバスインタフェース機能を選択	○	○

UART1機能選択レジスタ

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	0

シンボル U1SR アドレス 00F5h番地 リセット後の値 不定

機能	RW
UART1を使用する場合、0Fhにしてください。 UART1がクロック同期形またはクロック非同期形で使用可能となります。 0Fh以外の値を設定しないで下さい。読んだ場合、その値は不定。	WO

シリアルインタフェース関連レジスタ(1)

UARTiビットレートレジスタ (i=0~1) (注1、2)

b7 b0	シンボル U0BRG U1BRG	アドレス 00A1h番地 00A9h番地	リセット後の値 不定 不定
		機 能	設定範囲
	設定値をnとすると、UiBRGはカウントソースをn+1分周する		00h~FFh RW WO

注1. 送受信停止中に書いてください。

注2. MOV命令を使用して書いてください。

UARTi送信バッファレジスタ (i=0~1) (注1、2)

(b15)	(b8)	b7	b0	b7	b0	

注1. 転送データ長が9ビットの場合、上位バイト→下位バイトの順で書いてください。

注2. MOV命令を使用して書いてください

UARTi受信バッファレジスタ (i=0~1) (注1)

(b15)	(b8)	b7	b0	b7	b0	

注1. U0RBレジスタは必ず16ビット単位で読み出してください。

注2. SUM、PER、FER、OERビットは、UiMRレジスタのSMD2~SMD0ビットを“000b”（シリアルインタフェースは無効）にしたとき、またはUiC1レジスタのREビットを“0”（受信禁止）にしたとき、“0”（エラーなし）になります（SUMビットは、PER、FER、OERビットがすべて“0”（エラーなし）になると、“0”（エラーなし）になります）。また、PER、FERビットは、U0RBレジスタの上位バイトを読み出したとき、“0”になります。

シリアルインタフェース関連レジスタ(2)

UARTi送受信モードレジスタ

b7 b6 b5 b4 b3 b2 b1 b0								シンボル	アドレス	リセット後の値		
0								UOMR	00A0h番地	00h		
								U1MR	00A8h番地	00h		
								ビット シンボル	ビット名	機能	RW	
								SMD0	シリアルI/Oモード選択ビット (注2)	b2 b1 b0 0 0 0 : シリアルインタフェースは無効 0 0 1 : クロック同期形シリアルI/Oモード 1 0 0 : UARTモード転送データ長7ビット 1 0 1 : UARTモード転送データ長8ビット 1 1 0 : UARTモード転送データ長9ビット 上記以外 : 設定しないでください	RW	
											SMD1	RW
											SMD2	RW
								CKDIR	内/外部クロック選択ビット	0 : 内部クロック 1 : 外部クロック (注1)	RW	
								STPS	ストップビット長選択ビット	0 : 1ストップビット 1 : 2ストップビット	RW	
								PRY	パリティ奇/偶選択ビット	PRYE=1のとき有効 0 : 奇数パリティ 1 : 偶数パリティ	RW	
								PRYE	パリティ許可ビット	0 : パリティ禁止 1 : パリティ許可	RW	
								— (b7)	予約ビット	“0” にしてください。	RW	

注1. PD1レジスタのP1_6ビットを“0” (入力) にしてください。

注2. U1MRレジスタのSMD2～SMD0ビットを“000b”、“100b”、“101b”、“110b”以外にしないでください。

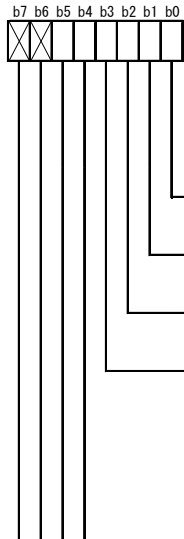
UARTi送受信制御レジスタ0

b7 b6 b5 b4 b3 b2 b1 b0								シンボル	アドレス	リセット後の値	
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>								U0C0	00A4h番地	08h	
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>								U1C0	00ACh番地	08h	
								ビット シンボル	ビット名	機能	RW
								CLK0	BRGカウントソース 選択ビット (注1)	b1 b0 0 0 : f1S10を選択 0 1 : f8S10選択 1 0 : f32S10を選択 1 1 : 設定しないでください	RW
								CLK1			RW
								— (b2)	予約ビット	“0” にしてください。	RW
								TXEPT	送信レジスタ空フラグ	0 : 送信レジスタにデータあり (送信中) 1 : 送信レジスタにデータなし (送信完了)	RO
								— (b4)	何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“0”。		—
								NCH	データ出力選択ビット	0 : TXDi端子はCMOS出力 1 : TXDi端子はNチャネルオープンドレイン出力	RW
								CKPOL	CLK極性選択ビット	0 : 転送クロックの立ち下がりで送信データ 出力、立ち上がりで受信データ入力 1 : 転送クロックの立ち上がりで送信データ 出力、立ち下がりで受信データ入力	RW
								UFORM	転送フォーマット選択 ビット	0 : LSBファースト 1 : MSBファースト	RW

注1. BRGカウントソースを変更した場合は、U1BRGレジスタを再設定してください。

シリアルインタフェース関連レジスタ(3)

UARTi送受信制御レジスタ1

								シンボル	アドレス	リセット後の値
								U0C1	00A5h番地	02h
								U1C1	00ADh番地	02h
ビット シンボル	ビット名							機能		RW
TE	送信許可ビット								0 : 送信禁止 1 : 送信許可	RW
TI	送信バッファ空フラグ								0 : UiTBにデータあり 1 : UiTBにデータなし	RO
RE	受信許可ビット								0 : 受信禁止 1 : 受信許可	RW
RI	受信完了フラグ(注1)								0 : UiRBにデータなし 1 : UiRBにデータあり	RO
UiIRS	UARTi送信割り込み要因選択ビット								0 : 送信バッファ空 (TI=1) 1 : 送信完了 (TXEPT=1)	RW
UiRRM	UARTi連続受信モード許可ビット (注2)								0 : 連続受信モード禁止 1 : 連続受信モード許可	RW
— (b7-b6)	何も配置されていない。書く場合、“0”を書いてください。読んだ場合、その値は“0”。									—

注1. RIビットはUiRBレジスタの上位バイトを読み出したとき、“0”になります。

注2. UARTモード時、UiRRMビットは“0” (連続受信モード禁止) にしてください。

8.4.1 クロック同期形シリアルI/Oモード

クロック同期形シリアルI/Oモードの仕様

項 目	仕 様
転送データフォーマット	8ビット長 固定
転送クロック	<ul style="list-style-type: none"> ・内部クロック選択時 : $f_i/2(n+1)$ $f_i = f_1/f_8/f_{32}$, $n = \text{UARTi 転送速度レジスタの値 (0~FFh)}$ ・外部クロック選択時 : CLKi 端子からの入力 (最大5MHz ($V_{cc}=5V$時))
送信開始条件	<ul style="list-style-type: none"> ・送信許可状態 (送信許可ビット="1") ・UARTi 送信バッファレジスタに送信データあり (UARTi 送信バッファ空フラグ="0")
受信開始条件	<ul style="list-style-type: none"> ・受信許可状態 (受信許可ビット="1") ・送信許可状態 (送信許可ビット="1") ・UARTi 送信バッファレジスタに送信データ (ダミーデータ) あり (UARTi 送信バッファ空フラグ="0")
割り込み要求発生 タイミング	送信割り込み (1) UARTi 送信バッファ → UARTi 送信レジスタへ転送時 (送信開始時) または (2) データ送信完了時 (送信レジスタ空フラグ="1") 受信割り込み 受信完了時 (UARTi 受信レジスタ → UARTi 受信バッファへデータ転送時)
エラー検出	オーバランエラー※のみ (UARTi 受信バッファレジスタの読み出しを行う前に次のデータの受信を開始し、データの7ビット目を受信した時点で発生)
選択機能	<ul style="list-style-type: none"> ・転送クロックの極性選択 ・送受信データのLSBファースト、MSBファースト選択 ・TxD端子の出力形式 (CMOS出力 / Nchオープンドレイン出力) 選択 ・連続受信モード

※ オーバランエラーが発生した場合、UARTi 受信バッファレジスタの内容は不定。またUARTi 受信割り込み要求ビットも変化しない。

Code : R8Cコース

Date : Rev. 2.20

Page: 65 of 63

連続受信モード

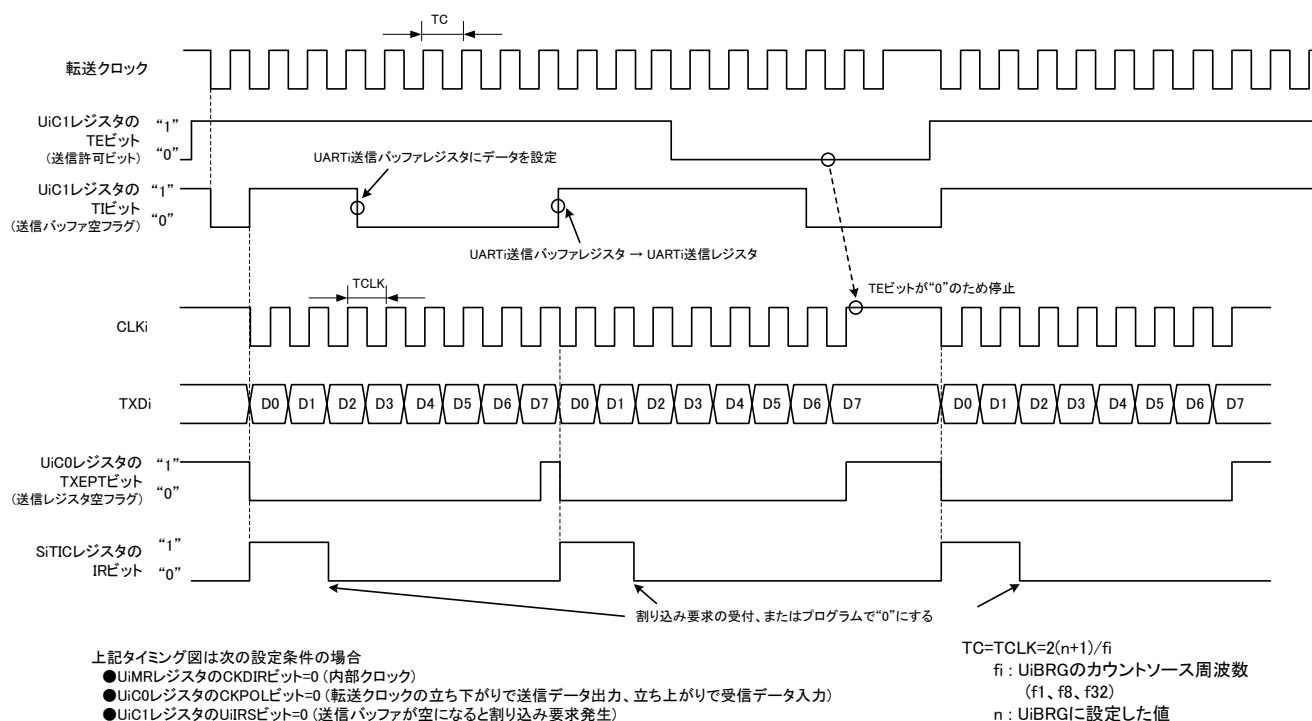
UART0 受信バッファレジスタの読み出しで送信バッファ空フラグ="0" となり、次データの受信が可能となるモード (送信バッファにダミーデータの書き込みが不要)

連続受信モードは、受信開始前に1度だけダミーデータを送信バッファに書き込み、受信開始時にダミーリード (受信バッファレジスタの読み出し) する必要があります。

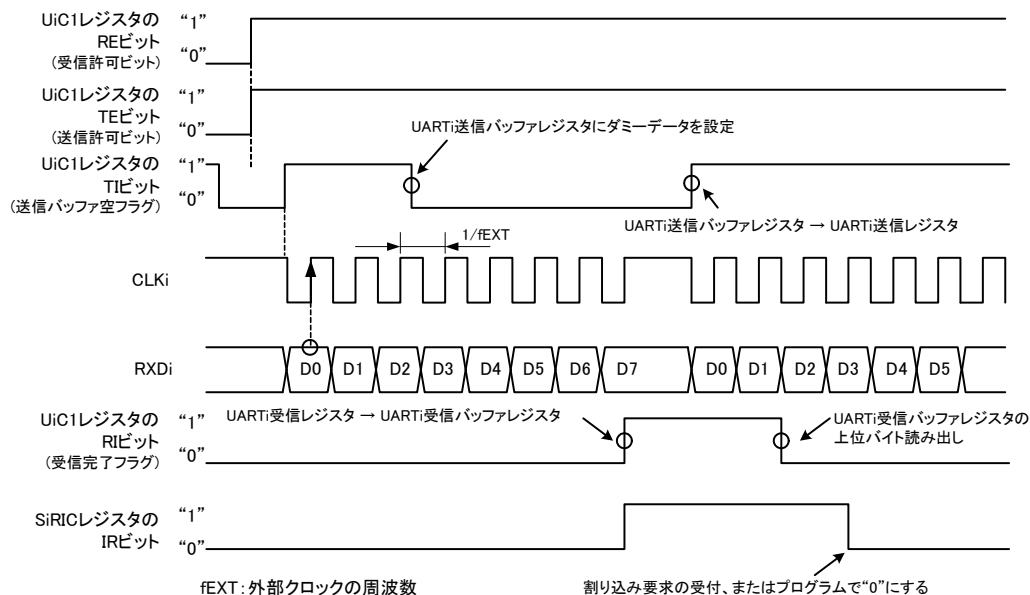
オーバランエラー

オーバランエラーが発生した場合は、受信割り込みが発生しないため、通信速度に合わせてタイマ等で受信完了フラグおよびエラーフラグをチェックしてください。

送信タイミング例



受信タイミング例



上記タイミング図は次の設定条件の場合

- UiMRレジスタのCKDIRビット=1 (外部クロック)
- UiC0レジスタのCKPOLビット=0 (転送クロックの立ち下がりで送信データ出力、立ち上がりで受信データ入力)

Code : R8Cコース

Date : Rev. 2.20

Page: 67 of 63

受信バッファの読み出しは必ず16ビット単位で行い、エラーフラグと受信データを同時に読み出してください(受信バッファの上位バイトを読み出すことで受信完了フラグがクリアされます)。

エラーが発生している場合は、エラーフラグとUARTi受信バッファレジスタを初期化した後、再度受信を行ってください。

<初期化手順>

- (1) 受信許可ビットを“0”(受信禁止)にする
- (2) シリアルI/Oモード選択ビットを“000b”(シリアルインタフェースは無効)にする
- (3) シリアルI/Oモード選択ビットを再設定する
- (4) 送信バッファにダミーデータを設定する
- (5) 受信許可ビットを再度“1”(受信許可)にする

※ 全てのエラーフラグは上記(1)または(2)の処理でクリアされます。

連続受信モードを選択している場合、(4)は省略し、(5)の後に受信バッファをダミーリードしてください。

8.4.2 クロック非同期形シリアルI/Oモード

クロック非同期形シリアルI/Oモードの仕様

項 目	仕 様
転送データフォーマット	キャラクタビット: 7ビット/8ビット/9ビット 選択可 パリティビット : 奇数/偶数/なし 選択可 ストップビット : 1ビット/2ビット 選択可
転送クロック	・内部クロック選択時: $f_i/16(n+1)$ $f_i = f_1/f_8/f_{32}$ 、 $n = \text{UARTi転送速度レジスタの値}(0 \sim \text{FFh})$ ・外部クロック選択時: $f_{\text{EXT}}/16(n+1)$ $f_{\text{EXT}} = \text{CLKiからの入力(最大5MHz、Vcc=5V時)}$ $n = \text{UARTi転送速度レジスタの値}(0 \sim \text{FFh})$
送信開始条件	・送信許可状態(送信許可ビット="1") ・UARTi送信バッファレジスタに送信データあり(UARTi送信バッファ空フラグ="0")
受信開始条件	・受信許可状態(受信許可ビット="1") ・スタートビットの検出
割り込み要求発生 タイミング	送信割り込み (1) UARTi送信バッファ → UARTi送信レジスタへ転送時(送信開始時) または (2) データ送信完了時(送信レジスタ空フラグ="1") 受信割り込み 受信完了時(UARTi受信レジスタ → UARTi受信バッファへデータ転送時)
エラー検出	オーバランエラー※1、フレーミングエラー、パリティエラー、エラーサムフラグ
選択機能※2	・送受信データのLSBファースト、MSBファースト選択(キャラクタビット長が8ビット時のみ選択可) ・TxD端子の出力形式(CMOS出力/Nchオープンドレイン出力)選択

※1. オーバランエラーが発生した場合、UARTi受信バッファレジスタの内容は不定。またUARTi受信割り込みの要求ビットも変化しない。

※2. クロック非同期形シリアルI/Oモードでは、転送クロック極性は選択不可(CLK極性選択ビット="0"(転送クロック立ち下がりで送信データ出力、立ち上がりで受信データ入力)で使用する)。

Code : R8Cコース

Date : Rev. 2.20

Page: 68 of 63

・ オーバランエラー

UARTi受信バッファレジスタを読み出す前に次のデータ受信を開始し、次のデータの最終ストップビットの1つ前のビットを受信したときに発生

・ フレーミングエラー

設定した個数のストップビットが検出されなかった時に発生

・ パリティエラー

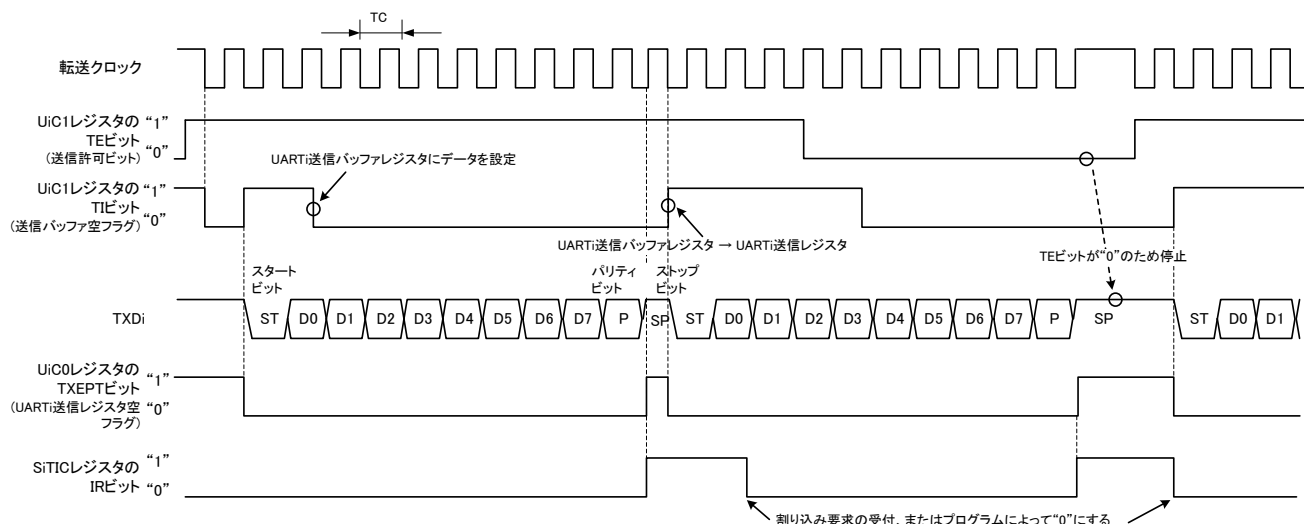
パリティ許可時にパリティビットとキャラクタビット中の“1”の個数(偶数個/奇数個)が設定した個数でなかったときに発生

・ エラーサムフラグ

オーバランエラー、フレーミングエラー、パリティエラーの何れか1つでも発生した場合に“1”にセットされる

オーバランエラー、フレーミングエラー、パリティエラー全てのエラーフラグが“0”(エラーなし)になった時点で“0”になる

送信タイミング例



上記タイミング図は次の設定条件の場合

- UiMRレジスタのPRYEビット=1 (パリティ許可)
 - UiMRレジスタのSTPSビット=0 (1ストップビット)
 - UiC1レジスタのUiIRSビット=1 (送信完了で割り込み要求を発生)
- i=0, 1

$$TC = 16(n+1)/f_j \text{ または } 16(n+1)/f_{EXT}$$

f_j : 内部クロック選択時のUiBRGのカウントソース周波数(f_1 , f_8 , f_{32})

f_{EXT} : 外部クロック選択時のUiBRGのカウントソース周波数

n : UiBRGに設定した値

Code: R8Cコース

Date: Rev. 2.20

Page: 69 of 63

UARTi転送速度レジスタの設定値算出方法

<内部クロック選択時>

$$n = \frac{f_j}{\text{ビットレート} \times 16} - 1$$

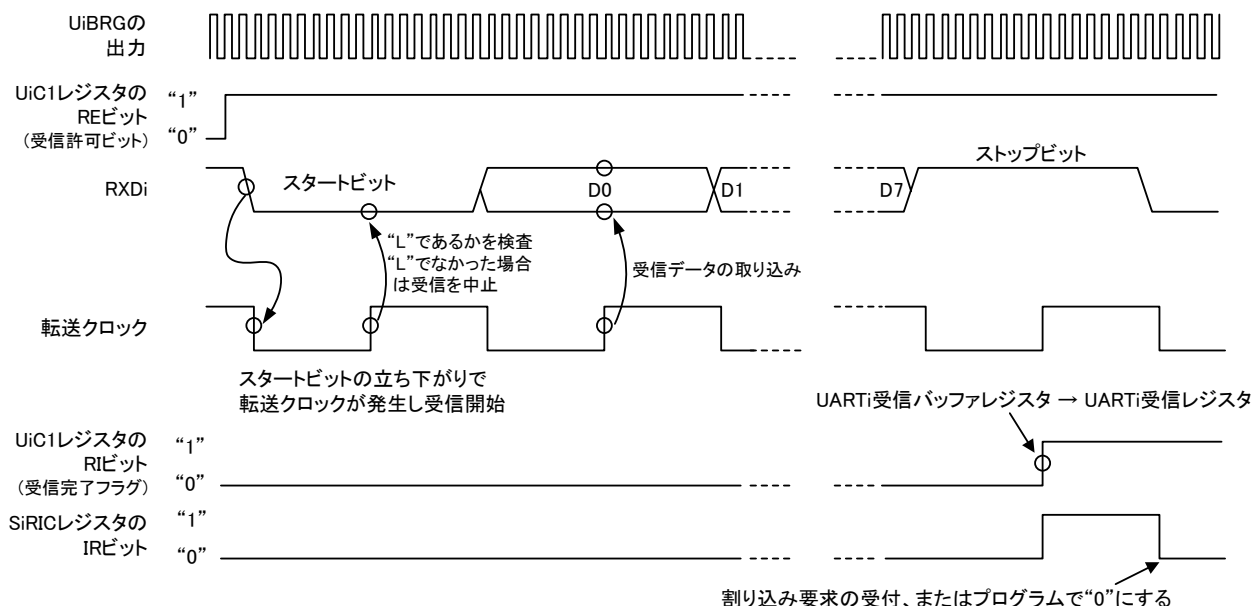
f_j : カウントソースの周波数

<外部クロック選択時>

$$n = \frac{f_{EXT}}{\text{ビットレート} \times 16} - 1$$

f_{EXT} : カウントソースの周波数

受信タイミング例



上記タイミング図は次の設定条件の場合

- UiMRLレジスタのPRYEビット=0 (パリティ禁止)
- UiMRLレジスタのSTPSビット=0 (1ストップビット) $i=0,1$

Code : R8Cコース

Date : Rev. 2.20

Page: 70 of 63

【ビットレートの設定例(内部クロック選択時)】

ビットレート (bps)	カウント ソース	メインクロック=20MHz		
		設定値(n)	実時間(bps)	誤差(%)
1200	f8	129(81h)	1201.92	0.16
2400	f8	64(40h)	2403.85	0.16
4800	f8	32(20h)	4734.85	-1.36
9600	f1	129(81h)	9615.38	0.16
14400	f1	86(56h)	14367.82	-0.22
19200	f1	64(40h)	19230.77	0.16
28800	f1	42(2Ah)	29069.77	0.94
31250	f1	39(27h)	31250.00	0.00
38400	f1	32(20h)	37878.79	-1.36
51200	f1	23(17h)	52083.33	1.73

8.5 A/Dコンバータ

A/Dコンバータ

項 目	仕 様
A/D変換方式	逐次比較変換方式
分解能	8ビット／10ビット 選択可
動作モード	<ul style="list-style-type: none"> ●単発モード ●繰り返しモード(8ビットモード時のみ)
アナログ入力端子	12ch(AN0～AN11)注
A/D変換開始条件	<ul style="list-style-type: none"> ●ソフトウェアトリガ A/D変換開始フラグ(ADST)により変換開始 ●キャプチャ ADST=1の状態でタイマRD割り込み要求が発生した場合
1端子あたりの変換速度	<ul style="list-style-type: none"> ●サンプル&ホールドなしの場合 分解能8ビット時 : 49 φ AD※サイクル 分解能10ビット時 : 59 φ ADサイクル ●サンプル&ホールドありの場合 分解能8ビット時 : 28 φ ADサイクル 分解能10ビット時 : 33 φ ADサイクル
Vrefカット機能あり	A/Dコンバータ未使用時に、ラダー抵抗に流れる電流を遮断し、消費電力を抑えることが可能

※ φ AD = fOCO-F/f1/f2/f4 の何れか

注. アナログ入力端子として使用する場合、対応するポートの方向を入力にすること

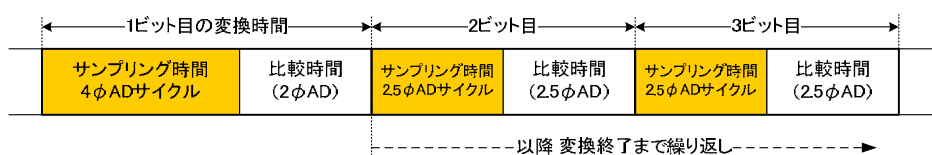
Code : R8Cコース

Date : Rev. 2.20

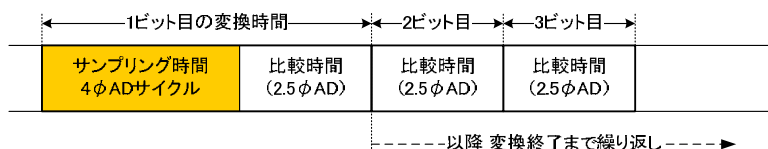
Page: 71 of 63

【サンプル&ホールドの有無によるA/D変換タイミングの違い】

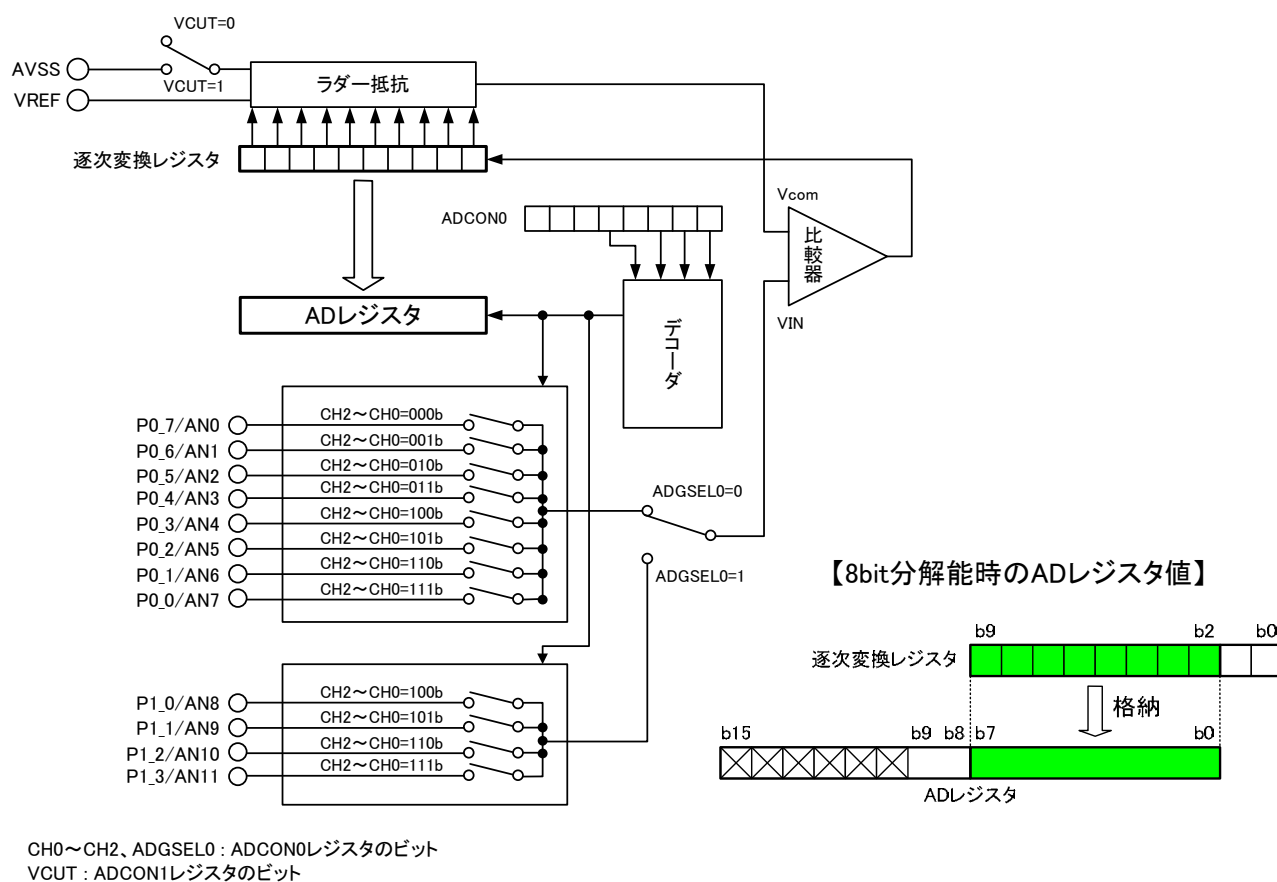
サンプル&ホールド無効時



サンプル&ホールド有効時



A/Dコンバータ構成図



Code : R8Cコース

Date : Rev. 2.20

Page: 72 of 63

A/Dコンバータ関連レジスタ(1)

A/D制御レジスタ0(注1)

b7 b6 b5 b4 b3 b2 b1 b0							
シンボル ADCON0		アドレス 00D6h番地		リセット後の値 0000000b			
ビット シンボル	ビット名		機能			RW	
CH0	アナログ入力端子選択 ビット		ADGSEL0 = 0のとき b2 b1 b0 0 0 0 : AN0 0 0 1 : AN1 0 1 0 : AN2 0 1 1 : AN3		ADGSEL0 = 1のとき b2 b1 b0 ----- 000b~011bは ----- 設定しないでください	RW	
CH1						RW	
CH2			1 0 0 : AN4 1 0 1 : AN5 1 1 0 : AN6 1 1 1 : AN7		1 0 0 : AN8 1 0 1 : AN9 1 1 0 : AN10 1 1 1 : AN11	RW	
MD	A/D動作モード選択 ビット(注2)		0 : 単発モード 1 : 繰り返しモード			RW	
ADGSEL0	A/D入力グループ選択 ビット(注4)		0 : ポートP0グループ選択 (AN0~AN7) 1 : ポートP1グループ選択 (AN8~AN11)			RW	
ADCAP	A/D変換自動開始ビット		0 : ソフトウェアトリガ(ADSTビット)で開始 1 : タイマRD(相補PWMモード)で開始			RW	
ADST	A/D変換開始フラグ		0 : A/D変換停止 1 : A/D変換開始			RW	
CKS0	周波数選択ビット0 (注3)		[ADCON1レジスタのCKS1=0の場合] 0 : f4を選択 1 : f2を選択 [ADCON1レジスタのCKS1=1の場合] 0 : f1を選択(注3) 1 : f0C0-Fを選択			RW	

注1. A/D変換中にADCON0レジスタの内容を書き換えた場合、変換結果は不定となります。

注2. A/D動作モードを変更した場合は、あらかじめアナログ入力端子を選択してください。

注3. φADの周波数を10MHz以下にしてください。

注4. アナログ入力端子はCH0~CH2ビットとADGSEL0ビットの組み合わせで選択できます。

A/Dコンバータ関連レジスタ(2)

A/D制御レジスタ1(注1)

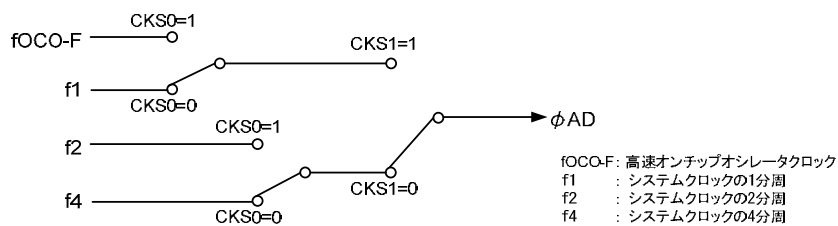
ビット シンボル	ビット名	機能	RW
— (b2-b0)	予約ビット	“0” にしてください。	RW
BITS	8/10ビットモード選択 ビット(注2)	0 : 8ビットモード 1 : 10ビットモード	RW
CKS1	周波数選択ビット1	0 : f2またはf4を選択 1 : f0C0-Fまたはf1を選択	RW
VCUT	Vref接続ビット (注3)	0 : Vref未接続 1 : Vref接続	RW
— (b7-b6)	予約ビット	“0” にしてください。	RW

注1. A/D変換中にADCON1レジスタの内容を書き換えた場合、変換結果は不定となります。

注2. 繰り返しモード時は、BITSビットを“0” (8ビットモード) にしてください。

注3. VCUTビットを“0” (未接続) から“1” (接続) にしたときは、1 μ s以上経過した後A/D変換を開始してください。

【動作クロック選択】



Code : R8Cコース

Date : Rev. 2.20

Page: 74 of 63

A/Dコンバータ関連レジスタ(3)

A/D制御レジスタ2(注1)

ビット シンボル	ビット名	機能	RW
SMP	A/D変換方式選択ビット	0 : サンプル&ホールドなし 1 : サンプル&ホールドあり	RW
— (b3-b1)	予約ビット	“0” にしてください。	RW
— (b7-b4)	何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“0”。		—

注1. A/D変換中にADCON2レジスタの内容を書き替えた場合、変換結果は不定となります。

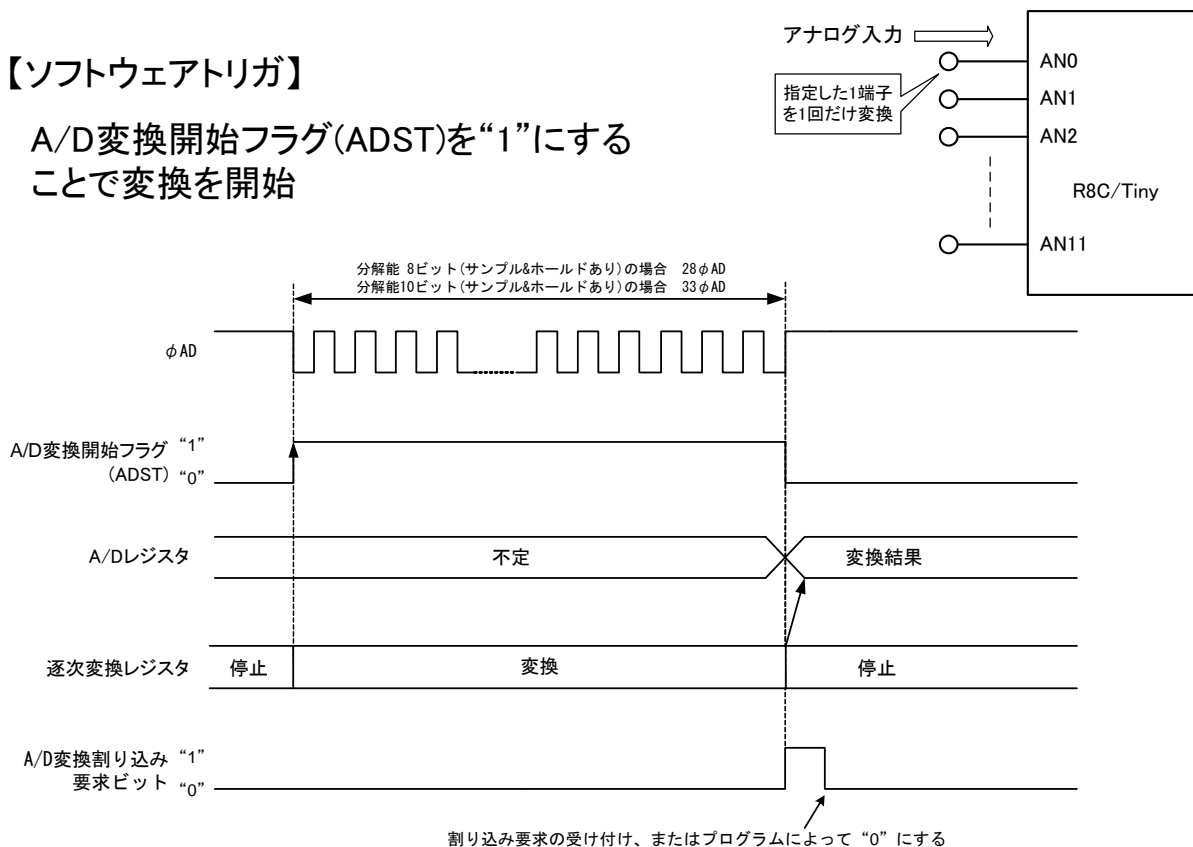
A/Dレジスタ

シンボル AD	アドレス 00C1h-00C0h番地	リセット後の値 不定	機能	RW
			ADCON1レジスタのBITSビットが “1” (10ビットモード) の場合	RW
			ADCON1レジスタのBITSビットが “0” (8ビットモード) の場合	RW
			A/D変換結果の下位8ビット	RO
			A/D変換結果の上位2ビット	RO
			何も配置されていない。書く場合、“0”を書いてください。 読んだ場合、その値は“0”。	—

単発モードの動作(1)

【ソフトウェアトリガ】

A/D変換開始フラグ(ADST)を“1”にすることで変換を開始



Code : R8Cコース

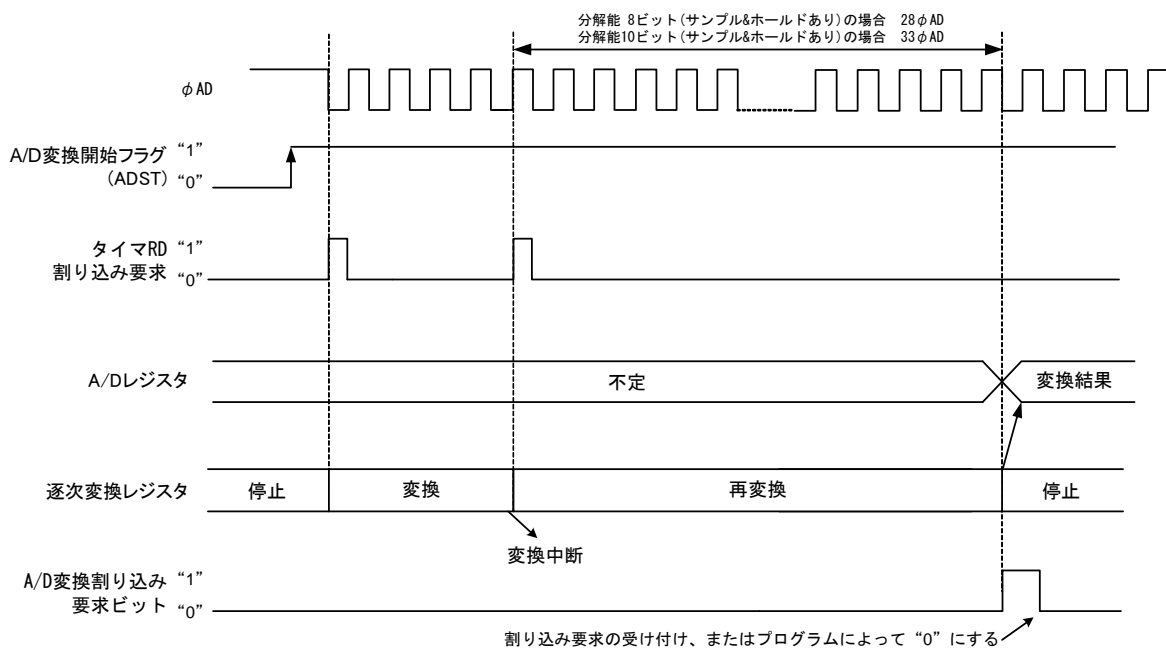
Date : Rev. 2. 20

Page: 76 of 63

単発モードの動作(2)

【キャプチャ】

A/D変換開始フラグ(ADST)が“1”の状態、タイマRD割り込み要求が発生すると変換を開始

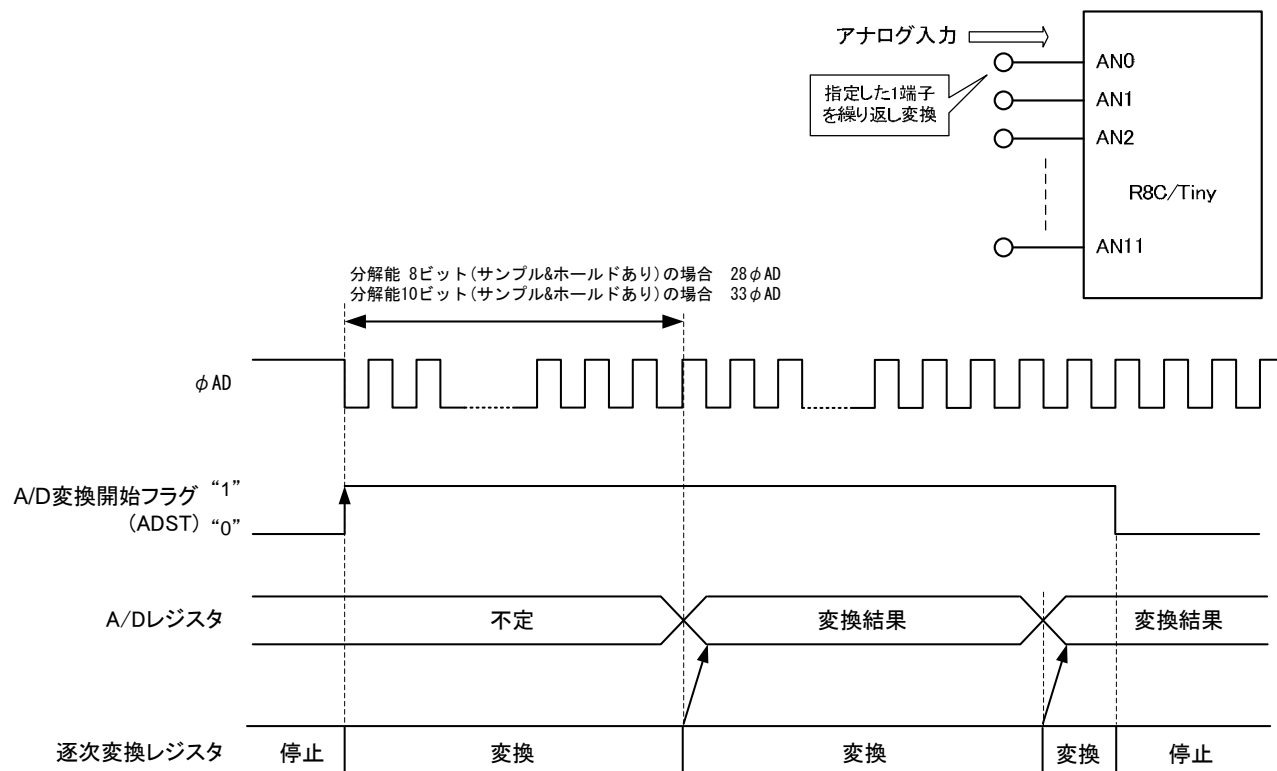


Code : R8Cコース

Date : Rev. 2.20

Page: 77 of 63

繰り返しモードの動作(ソフトウェアトリガ時)



Code : R8Cコース

Date : Rev. 2. 20

Page: 78 of 63

8.6 フラッシュメモリ

フラッシュメモリ制御

項 目		性 能
フラッシュメモリの動作モード		<ul style="list-style-type: none"> ・CPU書き換えモード ・標準シリアル入出力モード ・パラレル入出力モード
プログラム、イレーズ方式		<ul style="list-style-type: none"> ・プログラム: バイト単位 ・イレーズ: ブロック消去
プログラム、イレーズ制御方式		ソフトウェアコマンドによる制御(5コマンド)
プログラム、イレーズ回数	プログラムROM	100回(データフラッシュ内蔵品: 1,000回)
	データフラッシュ	10,000回
書き換え禁止機能		<ul style="list-style-type: none"> ・IDコードチェック機能(標準シリアル入出力モード対応) ・ROMコードプロテクト(パラレル入出力モード対応)

<CPU書き換えモード>

ライタを使用せずユーザプログラムでソフトウェアコマンドを実行することにより、CPUを基盤に実装したままユーザROM領域を書き換えるモード。プログラム動作中のデータ書き換えやデータバックアップなどで使用する。以下の2モードを持つ。

- ・イレーズライト0モード(EW0モード)
- ・イレーズライト1モード(EW1モード)

<標準シリアル入出力モード>

シリアルライタを使用し、ユーザROM領域を書き換えるモード。ブートROM領域内のブートプログラム(フラッシュ書き換え制御プログラム)によりアプリケーションプログラムを書き換える。アプリケーションプログラムをフラッシュROMに書き込む時に使用する。以下の2モードを持つ。

- ・標準シリアル入出力モード2(二線式のクロック非同期形シリアルI/O)
- ・標準シリアル入出力モード3(一線式のクロック非同期形シリアルI/O)

Code : R8Cコース

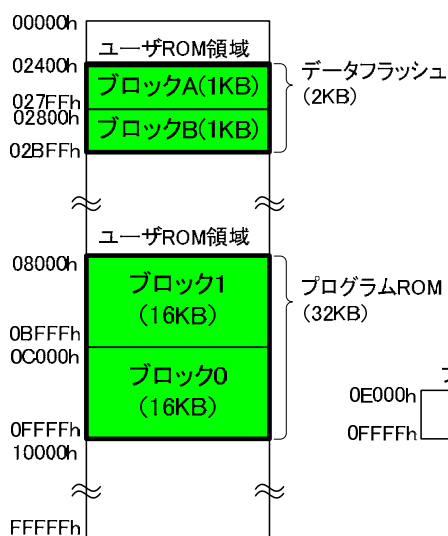
Date : Rev. 2.20

Page: 79 of 63

ブートプログラム

ブートROM領域にあらかじめ格納されているフラッシュ書き込み制御用のプログラムで、一般的に標準ブートプログラムとユーザブートプログラムがある。標準ブートプログラムはメーカーが提供するプログラムで、あらかじめブート領域に書き込まれている。これに対しユーザブートプログラムは、ユーザがフラッシュへの書き込み制御プログラムを作成し、市販のフラッシュライタを使用してユーザROM領域あるいはブートROM領域に書き込みを行う。

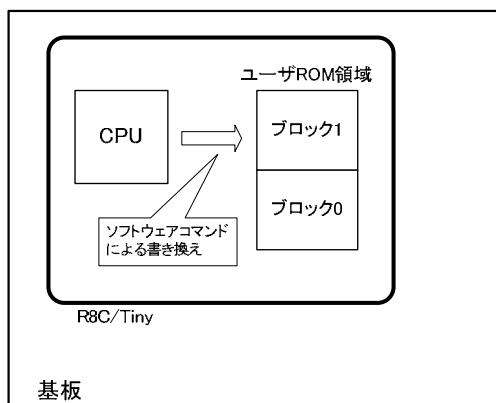
R8Cファミリは、標準ブートプログラムによる書き込み制御のみをサポートしている。



【R8C/25 ROM32KByte品のフラッシュメモリブロック配置】

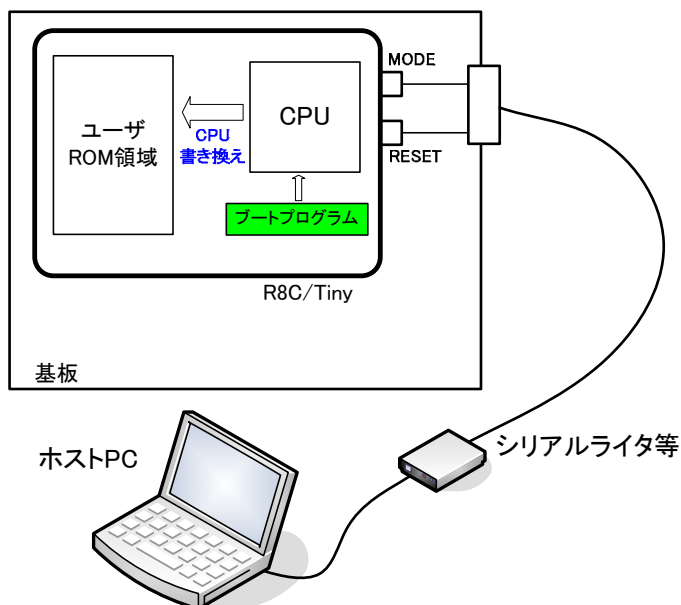
CPU書き換えモードと標準シリアル入出力モードの違い

【CPU書き換えモード】



ユーザプログラム実行中に
書き換えが可能なモード

【標準シリアル入出力モード】



Code : R8Cコース

Date : Rev. 2.20

Page:80 of 63

フラッシュメモリ制御レジスタ0

b7 b6 b5 b4 b3 b2 b1 b0								シンボル	アドレス	リセット後の値																	
<table border="1"><tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr><tr><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td></tr></table>								b7	b6	b5	b4	b3	b2	b1	b0			0	0					FMR0	01B7h番地	00000001b	
b7	b6	b5	b4	b3	b2	b1	b0																				
		0	0																								
ビットシンボル								ビット名	機能	RW																	
FMR00								RY/BYステータスフラグ	0: ビジー(書き込み、消去実行中) 1: レディ	RO																	
FMR01								CPU書き換えモード選択ビット(注1)	0: CPU書き換えモード無効 1: CPU書き換えモード有効	RW																	
FMR02								ブロック0、ブロック1書き換え許可ビット(注2、6)	0: 書き換え禁止 1: 書き換え許可	RW																	
FMSTP								フラッシュメモリ停止ビット(注3、5)	0: フラッシュメモリ動作 1: フラッシュメモリ停止 (低消費電力状態、フラッシュメモリ初期化)	RW																	
— (b5~b4)								予約ビット	“0” にしてください。	RW																	
FMR06								プログラムステータスフラグ(注4)	0: 正常終了 1: エラー終了	RO																	
FMR07								イレーズステータスフラグ(注4)	0: 正常終了 1: エラー終了	RO																	

注1. “1”にするときは、“0”を書いた後、続けて“1”を書いてください。“0”を書いた後、“1”を書くまでに割り込みが入らないようにしてください。

このビットはリードアレイモードにしてから“0”にしてください。

注2. “1”にするときは、FMR01ビットが“1”の状態、このビットに“0”を書いた後、続けて“1”を書いてください。“0”を書いた後、“1”を書くまでに割り込みが入らないようにしてください。

注3. このビットは、フラッシュメモリ以外の領域のプログラムで書いてください。

注4. クリアステータスコマンドを実行すると“0”になります。

注5. FMR01ビットが“1”(CPU書き換えモード)のとき有効です。FMR01ビットが“0”のとき、FMSTPビットに“1”を書くとFMSTPビットは“1”になりますが、フラッシュメモリは低消費電力状態にならず、初期化もされません。

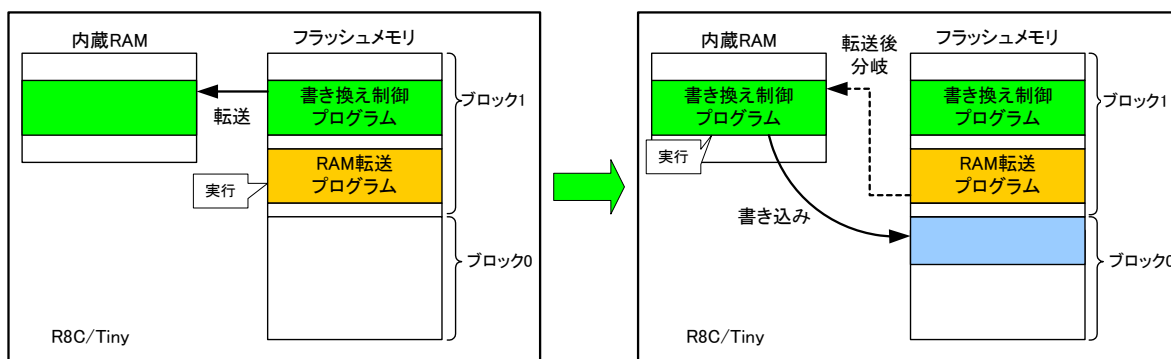
注6. FMR01ビットを“0”(CPU書き換えモード無効)にすると、FMR02ビットは“0”(書き換え禁止)になります。

8.6.1 CPU書き換えモード

EW0モードとEW1モード

項 目	EW0モード	EW1モード
書き換え制御プログラムが配置できる領域	ユーザROM領域	ユーザROM領域
書き換え制御プログラムを実行できる領域	フラッシュメモリ以外の領域で実行可能 (RAM領域へ書き換え制御プログラムを転送して実行させる)	ユーザROM領域またはRAM領域で実行可能
書き換え可能な領域	ユーザROM領域	書き換え制御プログラムが格納されているブロックを除くユーザROM領域
プログラム、イレーズ中のCPUの状態	動作	ホールド状態 (入出力ポートは実行前の状態を保持)
CPU書き換えモード時の動作周波数	5MHz以下	制限なし(CPU動作クロック)

<EW0モードでの書き込み動作例>



Code : R8Cコース

Date : Rev. 2.20

Page:81 of 63

フラッシュメモリ制御レジスタ1

シンボル	アドレス	リセット後の値	
FMR1	01B5h番地	1000000Xb	
ビットシンボル	ビット名	機能	RW
(b0)	予約ビット	読んだ場合、不定	RO
FMR11	EW1モード選択ビット (注1、2)	0 : EW0モード 1 : EW1モード	RW
— (b4-b2)	予約ビット	“0” にしてください。	RW
FMR15	ブロック0書き換え禁止ビット (注2、3)	0 : 書き換え許可 1 : 書き換え禁止	RW
FMR16	ブロック1書き換え禁止ビット (注2、3)	0 : 書き換え許可 1 : 書き換え禁止	RW
— (b7)	予約ビット	“1” にしてください。	RW

注1. “1” にするときは、FMR01ビットが“1” (CPU書き換えモード有効) の状態で、このビットに“0”を書いた後、続けて“1”を書いてください。“0”を書いた後、“1”を書くまでに割り込みが入らないようにしてください。

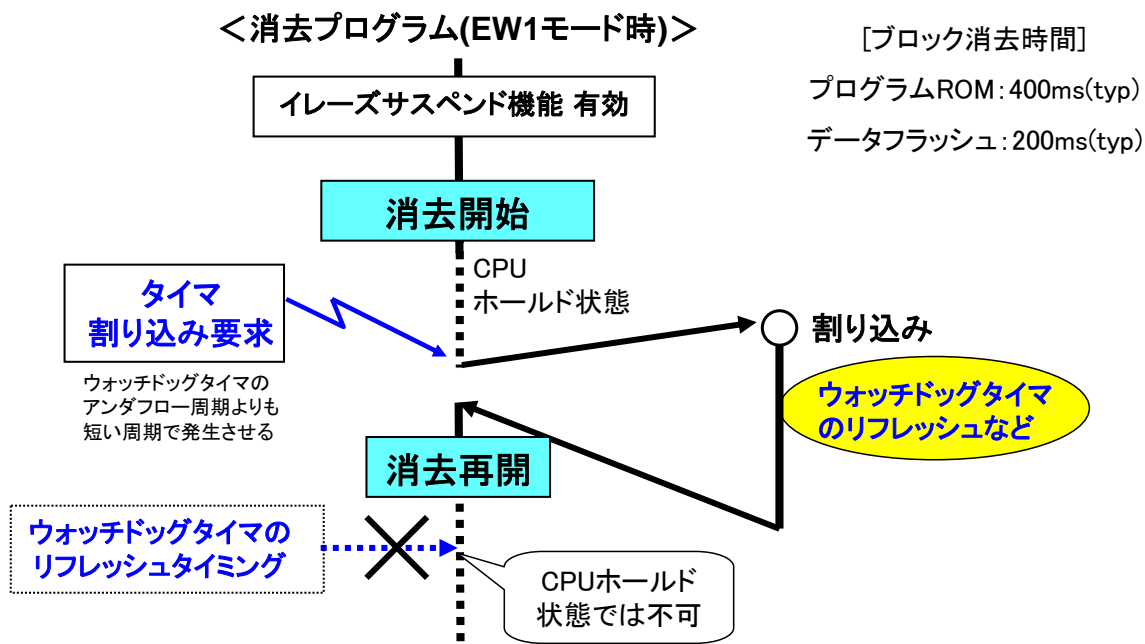
注2. FMR01ビットを“0” (CPU書き換えモード無効) にすると、“0”になります。

注3. FMR01ビットが“1” (CPU書き換えモード有効) のとき、FMR15およびFMR16ビットに書けます。“0”にするときは、このビットに“1”書いた後、続けて“0”を書いてください。“1”にするときは、このビットに“1”を書いてください。

イレーズ／プログラムサスペンド機能

イレーズ(またはプログラム)サスペンド機能とは、イレーズ(またはプログラム)処理を一時的に中断する機能。

EW1モードでは、割り込み要求によってイレーズ(またはプログラム)処理を中断し、CPUのホールド状態を解除することができる。これにより、ブロックイレーズ中(またはプログラム中)にもウォッチドッグタイマのリフレッシュなどが可能となる。



Code : R8Cコース

Date : Rev. 2.20

Page: 82 of 93

フラッシュメモリ制御レジスタ4

ビット	シンボル	アドレス	リセット後の値	
b7 b6 b5 b4 b3 b2 b1 b0	0	01B3h番地	01000000b	
ビット	シンボル	ビット名	機能	RW
FMR40	サスペンド機能許可ビット(注1)	0: 禁止 1: 許可		RW
FMR41	イレーズサスペンドリクエストビット(注2)	0: イレーズリスタート 1: イレーズサスペンドリクエスト		RW
FMR42	イレーズサスペンドリクエストビット(注3)	0: プログラムリスタート 1: プログラムサスペンドリクエスト		RW
FMR43	イレーズコマンドフラグ	0: イレーズ未実行 1: イレーズ実行中		RO
FMR44	プログラムコマンドフラグ	0: プログラム未実行 1: プログラム実行中		RO
— (b5)	予約ビット	“0” にしてください。		RO
FMR46	リードステータスフラグ	0: リード禁止 1: リード許可		RO
FMR47	低消費電流リードモード許可ビット(注1、4、5)	0: 禁止 1: 許可		RW

- 注1. “1”にするときは、このビットに“0”を書いた後、続けて“1”を書いてください。
“0”を書いた後、“1”を書くまでに割り込みが入らないようにしてください。
- 注2. このビットはFMR40ビットが“1”(許可)のときのみ有効になり、イレーズコマンド発行からイレーズ終了までの期間のみ、書き込みが可能となります。(上記期間以外は“0”になります。)
EW0モードではこのビットはプログラムによって“0”、“1”書き込みが可能となります。
EW1モードではFMR40ビットが“1”のとき、消去中にマスカブル割り込みが発生すると自動的に“1”になります。プログラムによって“1”を書き込むことはできません。(“0”書き込みは可能)
- 注3. FMR42ビットはFMR40ビットが“1”(許可)のときのみ有効になり、プログラムコマンド発行から自動書き込み終了までの期間のみ、このビットへの書き込みが可能となります。(上記期間以外は“0”になります。)
EW0モードではこのビットはプログラムによって“0”、“1”書き込みが可能となります。
EW1モードではFMR40ビットが“1”のとき、自動書き込み中にマスカブル割り込みが発生すると自動的に“1”になります。プログラムによって“1”を書き込むことはできません。(“0”書き込みは可能)
- 注4. 高速クロックモード、高速オンチップオシレータモードでは、FMR47ビットを“0”(禁止)にしてください。
- 注5. このモードはプログラム、イレーズ時には使用できません。FMR47ビットを“0”(禁止)にしてください。

8.6.2 標準シリアル入出力モード

IDコードチェック機能

標準シリアル入出力モード時のROMコードプロテクト機能。

シリアルライターから送られるチェック用のIDコードとフラッシュROM内に書かれている“7バイトのIDコード”を判定し、一致しない場合はライターから送られてくるコマンドを受け付けず、フラッシュROM内の読み出しおよび書き込みを禁止する。

固定ベクタ領域

	未定義命令ベクタ
0FFDFh	ID1
	INTO命令ベクタ
0FFE3h	ID2
	BRK命令ベクタ
	アドレス一致ベクタ
0FFEBh	ID3
	シングルステップベクタ
0FFEFh	ID4
	発振停止検出/ウォッチドッグ タイマ/電圧監視1.2ベクタ
0FFF3h	ID5
	アドレスブレイク
0FFF7h	ID6
	予約
0FFFBh	ID7
	リセットベクタ
0FFFFh	

IDコードの格納領域は、特殊割り込みなどのベクタテーブルの最上位1バイトと重なっているが、割り込み発生時の各ベクタテーブル最上位1バイトは無視される。

「ID1」が最上位のコードとなり、ID1-ID2・・・ID6-ID7の順でIDコードとして認識される。

IDコード設定例

「IDコードチェック機能」を有効にするには、IDコード格納番地にIDコードをあらかじめ設定したプログラムをフラッシュメモリに書き込んでおく必要がある。

【書き込み方法】

アセンブラ(as30)の指示命令“.ID”を使用して、
C言語スタートアッププログラム(fvector.c)内で設定

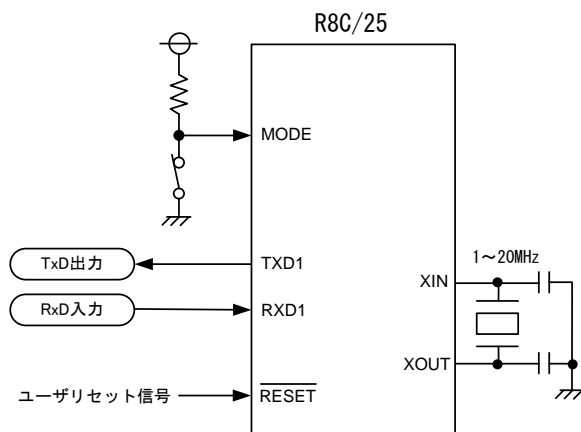
【設定(書き込み)例】

```
      :  
      // IDコード 12 34 56 78 9A BC FF を設定する。  
      _asm( ".id ""¥"#123456789ABC¥"" );  
      :
```

標準シリアル入出力モード時の接続例

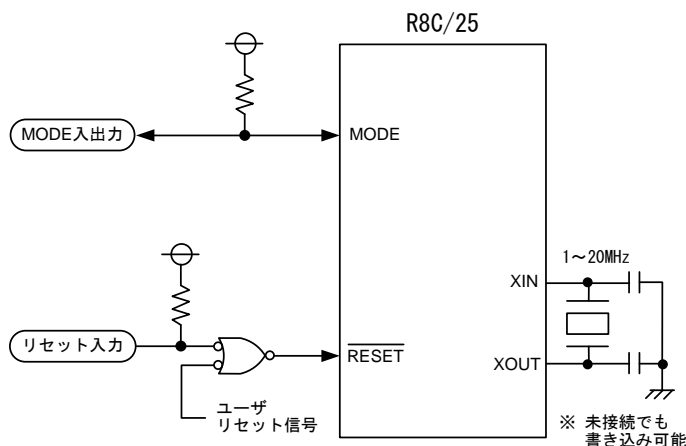
リセット解除後のMODE端子の入力レベルによりシングルチップモード／標準シリアル入出力モードでの起動を制御する。

M16C Flash Starter(一般的なシリアル接続)による書き込み時
(標準シリアル入出力モード2で起動)



スイッチでMODE端子の入力を制御することで、シングルチップモード(通常モード)と標準シリアル入出力モード(ブートモード)を手動で切り替える場合の接続例。
標準シリアル入出力モード2の場合、必ず発振子(1~20MHz)を接続する必要がある。

エミュレータ(E8/E8a)および市販のフラッシュライタ接続時
(標準シリアル入出力モード3で起動)



標準シリアル入出力モード3は、フラッシュライタやエミュレータ(E8/E8a)を接続することによって、シングルチップモード(通常モード)と標準シリアル入出力モード(ブートモード)をエミュレータおよびフラッシュライタ側で自動的に切り替える。
またCPUをオンチップオシレータで動作させている場合は、オンチップオシレータで通信を行うので、書き込み用に外部発振子を接続する必要はない。

Code : R8Cコース

Date : Rev. 2.20

Page:85 of 63

MODE端子の入力レベル

MODE端子=“H”でシングルチップモード(通常モード)、MODE端子=“L”で標準シリアル入出力モード(ブートモード)での起動となる。

通常(エミュレータ(E8/E8a)などを使用して開発する場合)、MODE端子をプルアップしておく。

標準シリアル入出力モード3

エミュレータやフラッシュライタとの通信はMODE端子1本で行うが、シリアル通信そのものはUART1を使用するため、E8aを接続してデバッグする場合、アプリケーション側でUART1によるシリアル通信はできない。(CLK1/P6_5、TXD1/P6_6、RXD1/P6_7端子は入出力ポートとして使用可能)

8.6.3 パラレル入出力モード

ROMコードプロテクト

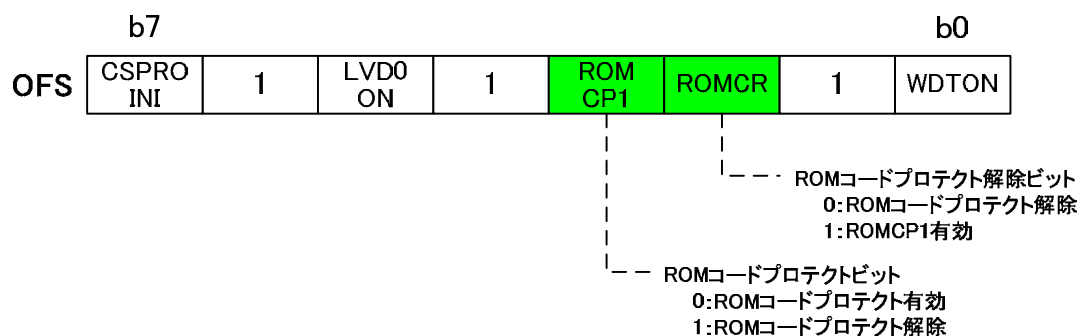
パラレル入出力モード時のプロテクト機能。

オプション機能選択レジスタの“ROMコードプロテクトビット(ROMCP1)”および“ROMコードプロテクト解除ビット(ROMCR)”により読み出しおよび書き込みの禁止／許可を制御する。

オプション機能選択レジスタ

アドレス:0FFFFh

出荷時の値:FFh





付録

- A. NC30起動オプション一覧
 - B. アセンブリ言語レベルの最適化
-

起動オプション(1)

コンパイルドライバ制御

オプション	機 能
-c	リロケータブルファイル(.obj)を作成し、処理を終了する
-D 識別子	識別子を定義する(#define と同じ)
-I ディレクトリ名	#include で指定するファイルのディレクトリを指定する
-preinclude	インクルードファイルを指定する
-E	プリプロセッサのみ起動し結果を標準出力に出力する
-P	プリプロセッサのみ起動しファイル(.i)を出力する
-S	アセンブリ言語ソースファイル(.a30)を作成し、処理を終了する
-U プリデファインドマクロ名	指定したプリデファインドマクロ定数を未定義にする
-silent	起動時のコピーライトメッセージを出力しない
-dsource (-dS)	C言語ソースリストをコメントとして出力した、アセンブリ言語ソースファイル(.a30)を作成する
-dsource_in_list (-dSL)	“-dsource”の機能に加え、アセンブリ言語リストファイル(.lst)を作成する
-lang	ソースファイルの言語(C/C++/EC++)を指定する

Code : R8Cコース

Date : Rev. 2. 20

Page: 1 of 11

※ ルネサス統合開発環境使用時は“-c”オプションが自動的に指定される。

出力ファイル指定

オプション	機 能
-o file 名	ln30 の生成するアブソリュートモジュールファイル、マップファイルなどの名称を指定する(拡張子の指定は不可) 例)-o test → test.x30 を出力
-dir ディレクトリ名	ln30 の生成するアブソリュートモジュールファイルの出力先ディレクトリを指定する

バージョン情報表示

オプション	機 能
-v	実行中のコマンドのプログラム名などを表示する
-V	コンパイラの各プログラムの起動メッセージを表示し、処理を終了する(コンパイル処理は行わない)

※ ルネサス統合開発環境使用時、出力ファイル指定オプション“-o”, “-dir”については、“プロジェクト”作成時に自動設定される。

デバッグ用

オプション	機 能
-g	デバッグ情報をアセンブリ言語ソースファイルに出力する C 言語レベルデバッグを行う場合は必須
-genter	関数呼び出し時に必ず enter 命令を出力する デバッガのスタックトレース機能を使用するときは必須
-gno_reg	レジスタ変数に対するデバッグ情報の出力を抑止する

“-g” オプションは、常に指定してコンパイルすることを推奨。

※ ルネサス統合開発環境使用時は“-c”オプションが自動的に指定される。

起動オプション(4)

最適化

オプション	機 能
-O[1~5]	レベル毎に速度及び ROM 容量ともに最小にする最大限の最適化を行う
-OR	ROM容量を重視した最適化を行う
-OS	速度を重視した最適化を行う
-Oconst (-OC)	const 修飾子で宣言した外部変数の参照を定数で置き換える最適化を行う
-Osp_adjust (-OSA)	スタック補正コードを取り除き、ROM効率を最適化する ただし、スタック使用量が多くなる可能性がある
-Ostack_frame_align (-OSFA)	スタックフレームの偶数アライメントを行う
-Oloop_unroll [=ループ回数] (-OLU)	ループ文を回さずに、ループ回数分コードを展開する “ループ回数”省略時は、最大 5 回のループ文が対象となる
-Ocompare_byte_to_word (-OCBTW)	連続した領域のバイト単位の比較をワード単位で行う
-Ostatic_to_inline (-OSTI)	static 宣言された関数を inline 宣言扱いにする
-Ofoward_function_to_inline (-OFFTI)	全てのインライン関数に対してインライン展開を行う (関数呼び出しの後にインライン関数の実体定義ができる)
-goptimize ※1	分岐命令に関する外部参照の最適化を行う（最適化リンクのモジュール間最適化時に使用する付加情報を出力ファイル内部に生成する）

Code : R8Cコース

Date : Rev.2.20

Page:4 of 11

【最適化オプション効果一覧表】

効 果	-O	-OR	-OS	-OSA	-OSTI
速度	良	悪	良	良	良
ROM サイズ	良	良	悪	良	悪
消費スタック	良	同	同	悪	良

以下にオプション“-O1 ~ -O5”の最適化について示す。

- O1 : -O3, -Ono_bit, -Ono_break_source_debug, -Ono_float_const_fold, -Ono_stdlib を有効にする
- O2 : -O1と同様
- O3 : 速度及びROM容量ともに効果のある最大限の最適化を行う
(-O) (デバッグ 情報に影響を与える最適化を行います)
- O4 : -O3 + -Oconst を有効にする
(ROMデータアクセス時にメモリ参照をせず、値を直接読み込む最適化を行う)
- O5 : 共通部分式の最適化(-OR同時指定時)、文字列転送、比較(-OS同時指定時)を強化した最大限の最適化※を行う

※ 異なるポインタ変数が同時に同じメモリ位置を指しており、それら変数を同一関数内で使用している場合、正常なコードを出力できない可能性がある。

※1 “-goptimize” オプションは、デフォルトで指定することを推奨する。

なお “-goptimize” オプション指定時は、必ずアセンブラ(as30)に “-goptimize”、最適化リンク(optlnk)に “-optimize=branch” オプションを指定すること。

起動オプション(5)

最適化(抑止)

オプション	機 能
-Ono_bit (-ONB)	ビット操作をまとめる最適化を抑止する
-Ono_break_source_debug (-ONBSD)	ソース行情報に影響する最適化を抑止する
-Ono_float_const_fold (-ONFCF)	浮動小数点の定数畳み込み処理を抑止する
-Ono_stdlib (-ONS)	標準ライブラリ関数のインライン埋め込みやライブラリ関数の変更等を抑止する
-Ono_logical_or_combine (-ONLOC)	論理 OR をまとめる最適化を抑止する
-Ono_asmopt (-ONA)	アセンブラオプティマイザ(aopt30)による最適化を抑止する

Code : R8Cコース

Date : Rev. 2. 20

Page: 5 of 11

<最適化抑止オプションの有効条件>

-Ono_bit : -0[3~5] / -OR / -OS オプションが指定されている場合

-Ono_break_source_debug : -0[3~5] / -OR オプションが指定されている場合

-Ono_float_const_fold : 条件なし (本コンパイラでは、デフォルトで定数の畳み込み処理を行う)

-Ono_stdlib : 条件なし (標準ライブラリ関数と同じ名前の関数をユーザー側で作成する場合に指定する)

-Ono_logical_or_combine : -0[3~5] / -OR / -OS オプションが指定されている場合

<浮動小数点畳み込み処理例>

[最適化前]

$$(val / 1000e250) * 50.0$$

↓

[最適化後]

$$(val / 20e250)$$

<論理ORをまとめる最適化例>

[最適化前]

$$\text{if}(a \& 0x01 \parallel a \& 0x02 \parallel a \& 0x04)$$

↓

[最適化後]

$$\text{if}(a \& 0x07)$$

起動オプション(6)

生成コード変更(1/3)

オプション	機 能
-fansi	-fNRA、-fNRFAN、-fNRI、-fETI を有効にする
-fnot_reserve_far_and_near (-fNRFAN)	far、near を予約語にしない(_far、_near のみ有効)
-fnot_reserve_asm (-fNRA)	asm を予約語にしない(_asm のみ有効)
-fnot_reserve_inline (-fNRI)	inline を予約語にしない(_inline のみ有効)
-fextend_to_int ※1 (-fETI)	char 型のデータを int 型に拡張して演算を行う (ANSI 規格で定められた拡張を行う)
-ffar_RAM (-fFRAM)	RAM データのデフォルト属性を far に変更する
-fnear_ROM (-fNROM)	ROM データのデフォルト属性を near に変更する
-fno_carry (-fNC)	far ポインタ間接によるデータアクセス時のキャリーフラグ の加算を抑止する
-ffar_pointer (-fFP)	ポインタ変数のデフォルト属性を far にする
-fchar_enumerator (-fCE)	enum 型を signed int 型ではなく unsigned char 型 として扱う

Code : R8Cコース

Date : Rev. 2. 20

Page: 6 of 11

※1 NC30では、デフォルトでコード効率と実行速度を重視したコード生成を行うため char型データの演算については、int型に拡張せずにchar型のままで行います。よってchar型演算を行うときに、演算途中でオーバーフローが発生する可能性がある場合（ANSI規格で定められている標準仕様に従ってchar型をint型に拡張して処理する場合は、“-fextend_to_int”オプションを指定してください）。

< char型演算の例 >

```
void main(void)
{
    char c1, c2 = 200, c3 = 2;

    c1 = c2 * 2 / c3 ;
}
```

→ c2*2の演算時点でオーバーフロー
するため正しい結果が得られない。

起動オプション(7)

生成コード変更(2/3)

オプション	機 能
-fno_even (-fNE)	データ出力時に奇数データと偶数データを分離せず、全て奇数(odd 属性)セクションに配置する
-fconst_not_ROM (-fCNR)	const で指定されたデータ型を ROM データとして扱わない
-fnot_address_volatile (-fNAV)	#pragma ADDRESS で指定した変数を volatile 指定した変数として扱わない
-fsmall_array ※1 (-fSA)	far 型の配列を参照する場合、その配列のサイズが 64K バイト以内であれば、添え字の計算を 16 ビットで行う
-fno_align (-fNA)	関数先頭アドレスのアライメントをしない
-fbit ※2 (-fB)	near 領域に配置した外部変数全てに対して 1bit 操作命令が使用できると仮定してコード生成を行う
-fauto_128 (-fA1)	使用するスタックフレームを最大 128byte に制限する
-fuse_DIV ※3 (-fUD)	除算に対するコード生成を変更する(標準関数を使用せず、DIV 命令を使用して演算を行う)
-fuse_MUL	乗算に対する生成コードを変更する(16bit × 16bitを32bitに格納する場合、キャストしなくても32bitの結果が得られる)

Code : R8Cコース

Date : Rev. 2. 20

Page: 7 of 11

- ※1 サイズが64Kバイトを超える配列を宣言しない場合、常に指定してコンパイルすることを推奨。
- ※2 外部変数に対してビット操作している部分をR8C/Tinyの持つビット命令に展開したい場合に指定する。本オプション指定により、外部変数に対してビット操作している部分は16ビット絶対アドレッシングのビット命令に展開される。したがって、アクセス対象のビットは0h~1FFh番地の8Kbyte領域に配置されていなければならない。
- ※3 除算部分の速度およびサイズを向上させたい場合に指定する。ただし本オプションを指定した場合、除算によるオーバーフローが発生したときに演算結果が保障されないので注意すること。

起動オプション(8)

生成コード変更(3/3)

オプション	機 能
-fenable_register ※1 (-fER)	register 指定された変数を強制的にレジスタに割り当てる
-finfo ※2	"utl30"に必要なインスペクタ情報出力する
-fswitch_other_section (-fSOS)	switch文に対するテーブルコードをプログラムセクションとは別のセクション(switch_tableセクション)に出力する
-fchange_bank_always ※3 (-fCBA)	毎回バンク切り替えを行う
-fauto_over_255 ※4 (-fAO2)	1つの関数内で確保可能なスタックフレームサイズを64Kbyteに 変更する
-fno_switch_table (-fNST)	switch文に対して、比較を行ってから分岐するコードを生成する (本オプションを指定しない場合、コードサイズがより小さくなる ときのみジャンプテーブルを用いたコードを生成する)
-fdouble_32 ※5 (-fD32)	double 型をfloat 型として処理する
-fenable_register_pointer (-fERP)	レジスタ指定されたポインタ変数に対してレジスタを割り当てる
-ferase_static_function=関数名 (-fESF=関数名)	本オプションで指定された関数が“static関数”の場合、 コード生成を行わない
-R8C ※6	R8C/Tiny(ROM64KB 未満)に対応したコードを生成する
-R8CE ※6	R8C/Tiny(ROM64KB 以上)に対応したコードを生成する -fnear_rom は指定されず、ffar_ram の指定が可能

Code : R8Cコース

Date : Rev. 2. 20

Page: 8 of 11

※1 NC30では“register記憶クラス”指定された変数を必ずしもレジスタに割り当てることは行わず、auto変数と同様に扱う。register記憶クラスの指定を行った変数を必ずレジスタに割り当てたい場合は、本オプションを指定する。ただし“register”指定を乱用すると、逆にコード効率を低下させる場合もあるため、使用する際は注意すること。
なお最適化オプションを指定している場合、“register”指定されていない変数でも最適化の一環としてレジスタに割り当てられることがある。

※2 ルネサス統合開発環境使用時は、本オプションが自動的に指定される。SBDATA宣言 & SPECIALページ関数宣言ユーティリティ(utl30)において、本オプションで出力されたアブソリュートファイル(***.abs)が必要となる。

※3 “-R8C/-R8CE”オプション指定時は無視される。

※4 本オプションを指定するとスタックフレームをSB相対でアクセスするコードに置き換えるため、ファイル中で拡張機能“#pragma SBDATA”が使用できなくなる。

※5 プログラム中で、32bit÷16bit演算を行った場合double型での演算となる。float型演算のみ行うシステムであれば、ROMサイズ削減で有効なオプション。

※6 本オプションを指定する場合、必ずアセンブラ(as30)にも“-R8C”オプションが必要(ルネサス統合開発環境使用時は、「CPUオプション」によりコンパイラとアセンブラに同時指定される)。
なおルネサス統合開発環境でプロジェクト作成時に、ターゲットCPUとして「R8C/Tinyシリーズ」を選択した場合は本オプションが自動的に指定される。
標準ライブラリについてはライブラリジェネレータがオプションに応じて自動的に生成し、最適化リンク(optlnk)でそのライブラリをリンクする(-R8C指定時: r8clib.lib, -R8CE指定時: r8celib.lib)。

起動オプション(9)

警告(1/2)

オプション	機 能
<i>-Wnon_prototype</i> (-WNP)	プロトタイプ宣言がされていない関数に対して警告を出力する
<i>-Wunknown_pragma</i> (-WUP)	サポートしていない#pragma を使用した場合、警告を出力する
-Wno_stop (-WNS)	エラーが発生してもコンパイル作業を停止しない
-Wstop_at_warning (-WSAW)	ワーニング発生時にコンパイル処理を停止する
-Wnesting_comment (-WNC)	コメント中に“/*”を記述した場合に警告を出力する
-Wno_used_static_function ※ (-WNUSF)	コード生成が不要な static 関数(ファイル内のどこからも参照されていない関数など)を表示する
-Wno_used_function (-WNUF)	未使用のグローバル関数をリンク時に表示する

Code : R8Cコース

Date : Rev. 2. 20

Page:9 of 11

オプション “-Wnon_prototype”, “-Wunknown_pragma”は、常に指定してコンパイルすることを推奨。

※ “-Wno_used_static_function” で表示された関数は、“-ferase_static_function”を使用してコード生成を抑止できる。

起動オプション(10)

警告 (2/2)

オプション	機 能
-Wccom_max_warnings=ワーニング回数 (-WCMW)	コンパイラ本体の出力するワーニング回数の上限を指定する
-Wall	検出可能な警告を全て表示する(-WLTS,-WNUA で出力される警告は除く)
-Wuninitialize_variable (-WUV)	初期化されていない auto 変数に対してワーニングを出力する
-Wlarge_to_small (-WLTS)	大きいサイズから小さいサイズへの暗黙の転送に対してワーニングを出力する
-Wno_warning_stlib (-WNWS)	プロトタイプ宣言されていない標準ライブラリに対する警告を抑止する(-WNP、-Wall 指定時に有効)
-Wno_used_argument (-WNUA)	引数を持つ関数を定義した場合、使用していない引数に対してワーニングを出力する
-Wundefined_macro (-WUM)	未定義のマクロ名を#if の中で使用した場合に警告する
-Wstop_at_link (-WSAL)	リンク時に全てのインフォメーションおよびウォーニングメッセージをエラーレベルに変更し、リンク処理を中断する

Code : R8Cコース

Date : Rev. 2. 20

Page: 10 of 11

“-Wall” オプションは、“-Wnon_prototype”, “-Wunknown_pragma”, “-Wnesting_comment”, “-Wuninitialize_variable” オプションと同等の警告を出力すると同時に、以下の場合にも警告を出力します。

(例1) 制御文の式で関係演算子 “==” ではなく代入演算子 “=” を記述した場合

```
if( i = 0 ){
    j = 0xf5;
    func( j );
}
```

(例2) 代入演算子 “=” を間違って関係演算子 “==” で記述してしまった場合

```
if( i == 0 ){
    j == 0xf5;
```

(例3) K&R形式の関数定義を行った場合

```
func( i )
int i;
{
    :
}
```

ライブラリ指定

オプション	機 能
-l ライブラリファイル名	リンク時に使用するユーザーライブラリを指定する

アセンブル・リンク

オプション	機 能
-as30 “opt1 opt2 ..”	アセンブラ (as30) のオプションを指定する (一部のオプションは指定不可)
-lnkcmd =コマンド ファイル名	最適化リンカ(optlnk)にコマンドファイルを指定する

<ライブラリ指定オプション注意点>

標準関数ライブラリは、ビルド時にライブラリジェネレータにより自動的に生成され、最適化リンカ(optlnk)でそのライブラリ (r8clib.lib／r8celib.lib／r8cs16.lib／r8ces16.lib／nc30lib.lib／nc30s16.lib) がリンクされる。

<アセンブル・リンクオプション注意点>

“-as30オプション”で、as30の『-.、-C、-M、-O、-P、-T、-V、-X』オプションは指定できない。

付録B.1 引数の引渡し規則

引数の渡され方

nc30では、引数渡しの方法として「レジスタ渡し」と「スタック渡し」の2通りを使用し、関数呼び出し時の条件によりどちらかが選択される。

レジスタ経由で渡される条件

- ① 関数のプロトタイプ宣言を行い、関数呼び出し時に引数の型が確定している
- ② プロトタイプ宣言に可変引数“...”を使用していない
- ③ 関数の引数の型が以下と一致している場合

第1引数	第2引数
_Bool 型, char 型 short 型, int 型 near ポインタ型	short 型, int 型 near ポインタ型

※ 上記の条件を満たさない場合は、全てスタック渡しとなる。

Code : R8Cコース

Date : Rev.2.20

Page:12 of 11

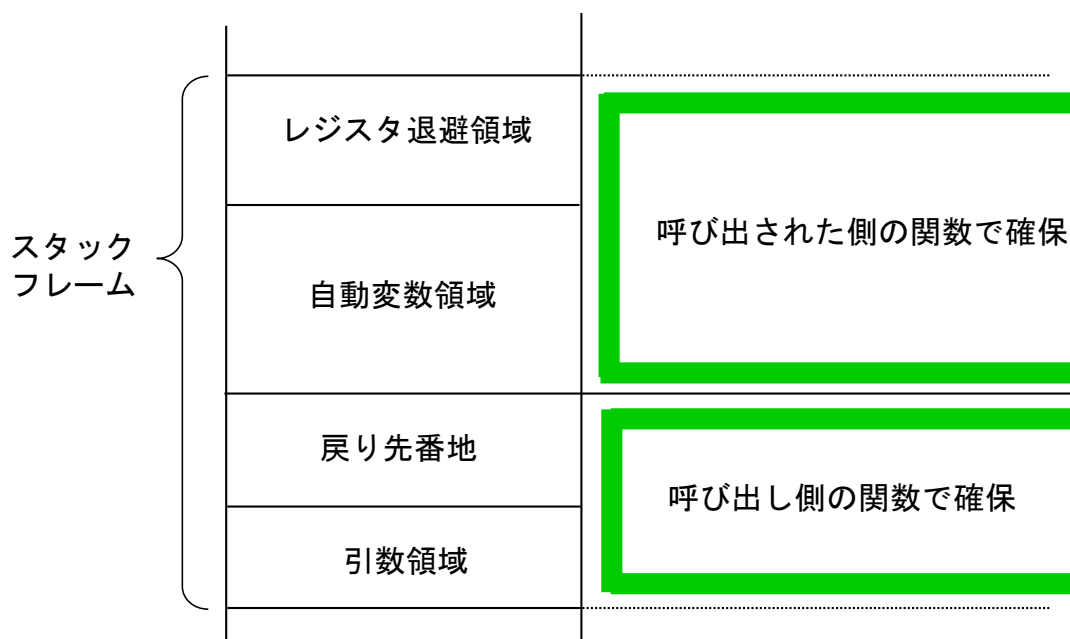
<レジスタ渡しになった場合のレジスタへの割り当て規則>

引数の型	第1引数	第2引数	第3引数以降
_Bool 型, char 型	R1L	スタック	スタック
short 型, int 型 near ポインタ型	R1	R2	スタック
その他の型	スタック	スタック	スタック

付録B.2 スタックフレームの構築と解放

スタックフレームの構成

一般的に関数呼び出しを行うたびに以下のような領域がスタック上に構成される。
この領域はスタックフレームと呼ばれ、関数からリターンする際に解放される。



スタックフレームの構築例(呼び出し側)

【プログラム例】

```
void func(int, char);
```

```
void main ( void )
```

```
{
```

```
    int i = 1 ;
```

```
    char c = 2 ;
```

```
    :
```

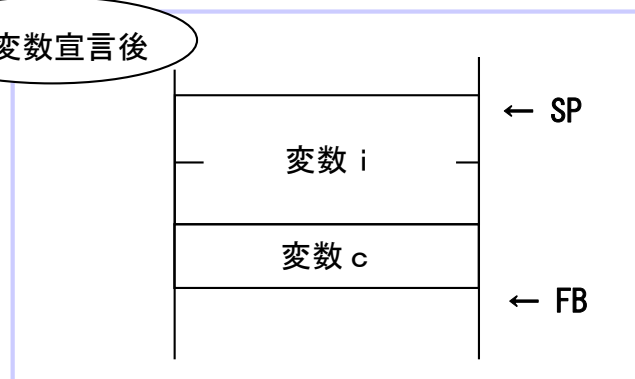
```
    :
```

```
    func( i, c ) ;
```

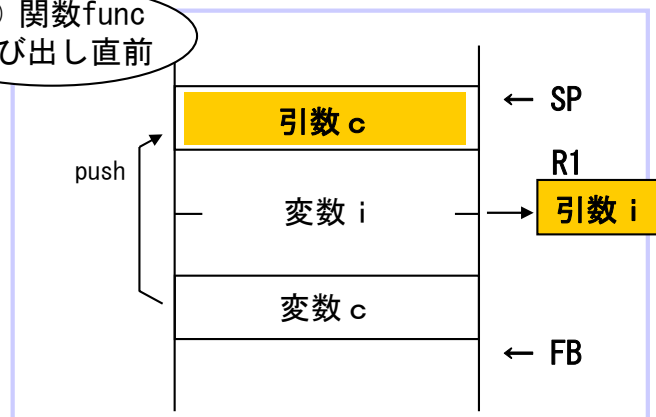
```
    :
```

```
}
```

① 変数宣言後



② 関数func呼び出し直前



Code : R8Cコース

Date : Rev. 2.20

Page:14 of 11

NC30ではauto変数の領域をスタック上に確保する際、複数の変数が同時に使用されない場合は最適化を行い、1つの変数領域のみ確保し、その領域を複数の変数で共有します。

例えば以下のような場合、スタック上にはauto領域として2バイトのみ確保します。

```
void func(void)
```

```
{
```

```
    int i, j, k;
```

```
    for( i=0 ; i<=0 ; i++) {
```

```
        処理
```

iの有効範囲

```
    }
```

```
    :
```

```
    for( j=0 ; j<=0 ; j++) {
```

```
        処理
```

jの有効範囲

```
    }
```

```
    :
```

```
    for( k=0 ; k<=0 ; k++) {
```

```
        処理
```

kの有効範囲

```
    }
```

```
}
```

スタックフレームの構築例 (呼び出され側 1)

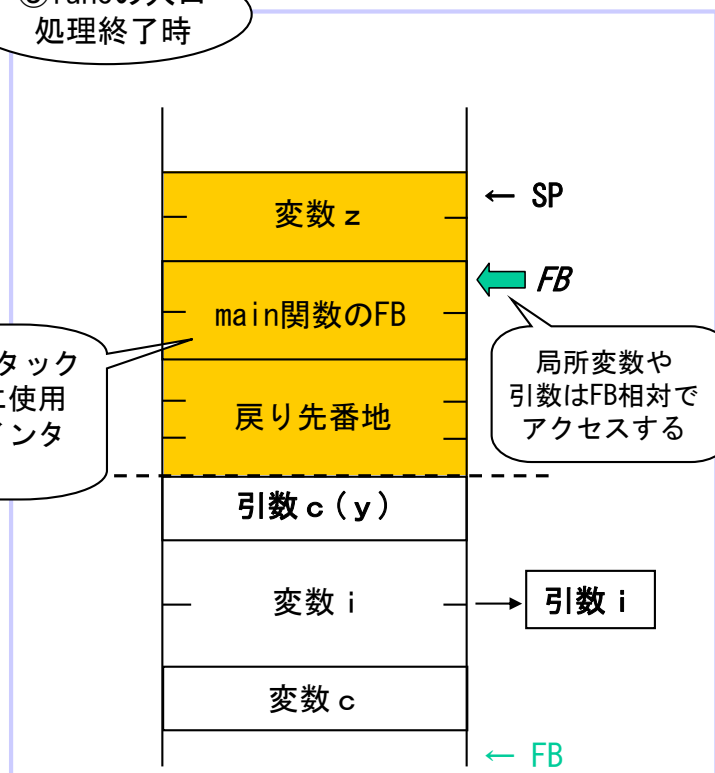
～ 自動変数のサイズが128バイト以下のとき ～

【プログラム例】

```
void func ( int x , char y )
{
    int z ;
    z = x + y ;
    :
    :
}
```

呼び出し元 (main) のスタック
フレームアクセス用に使用
していたフレームポインタ
(FB) を格納する

③funcの入口
処理終了時



Code : R8Cコース

Date : Rev. 2. 20

Page: 15 of 11

関数の入り口で、スタックフレームを構築する “ ENTER 命令 ” を出力する。

<ENTER命令の機能>

記述例 : ENTER src (srcは自動変数のバイト数)

- ① FBレジスタの値をスタックに退避する
- ② スタックポインタ (SP) の値をFBレジスタにコピーする
- ③スタックポインタ (SP) からsrc分を減算し、スタックポインタ (SP) 値を変更する (自動変数領域の確保)

<アセンブリ言語展開例 >

```
void func(int x, char y)
{
    int z;
    :
    z = x + y;
    :
}
```

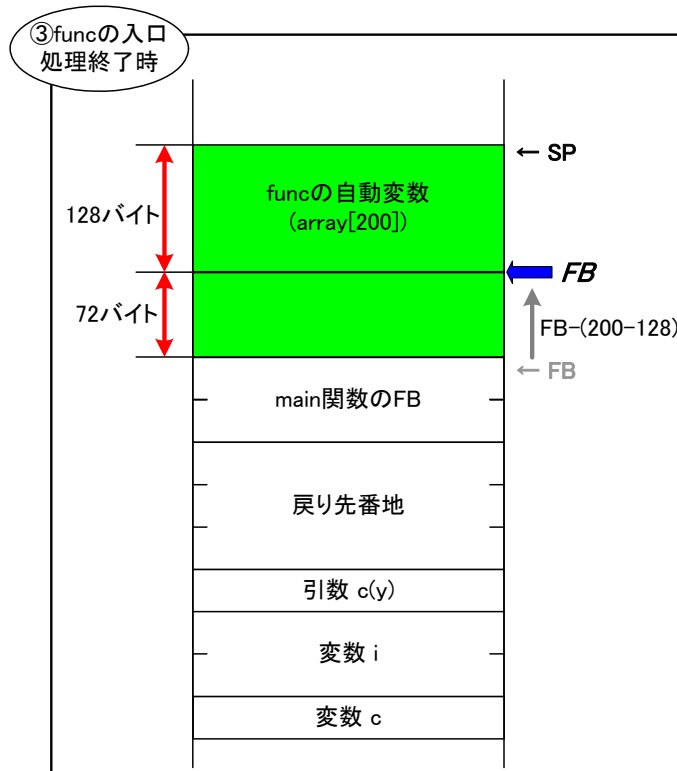
```
$func:
    enter    #4
    :
    :
    :
    :
    exitd
```

スタックフレームの構築例 (呼び出され側 2)

～ 自動変数のサイズが128バイトを超えるとき ～

【プログラム例】

```
void func ( int x , char y )
{
    char array[200] ;
    :
    :
    funcの処理
    :
    :
}
```



Code : R8Cコース

Date : Rev. 2. 20

Page: 16 of 11

自動変数のサイズが128バイトを超える場合、FBレジスタの値を減算することで、スタックフレームの拡張を行う。

よって、NC30で扱える自動変数の最大バイト数はデフォルトでは255バイトとなる。

※生成コード変更オプション

-fauto_128 (-fA1) :

使用するスタックフレームを最大128バイトに制限する。

-fauto_over_255 (-fA02) :

1つの関数で確保可能なスタックフレームサイズを最大64Kバイトに変更する。
(SBレジスタを使ってauto変数をアクセスする)

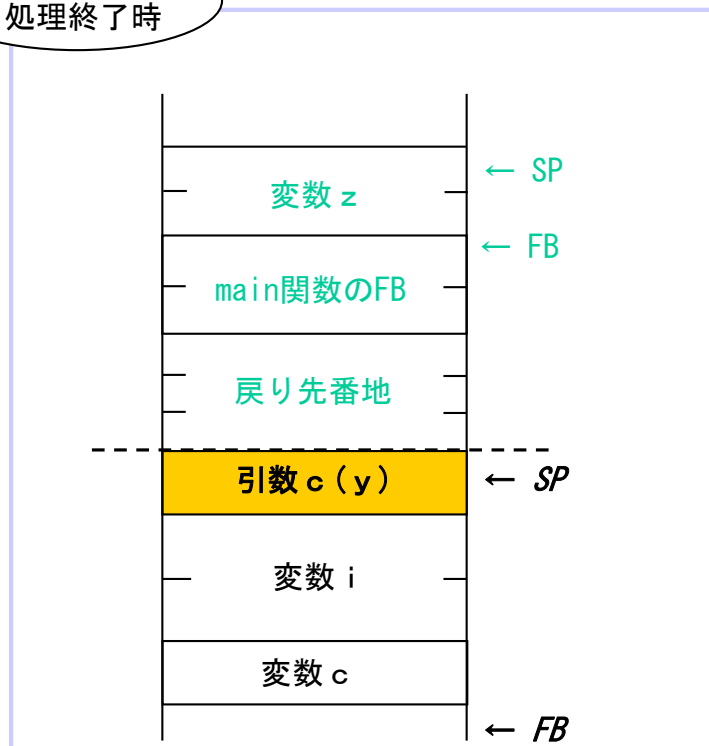
スタックフレームの解放

【プログラム例】

```
void func ( int x ,char y )
{
    int z ;

    z = x + y ;
    :
    :
}
← ④
```

④funcの出口
処理終了時



Code : R8Cコース

Date : Rev.2.20

Page:17 of 11

関数の出口でスタックフレームを解放する “ EXITD 命令” を出力する。

<EXITD命令の機能>

記述形式 : EXITD

- ① FBレジスタの内容をスタックポインタ (SP) に転送する (自動変数領域の解放)
- ② 旧FBレジスタの内容をスタックからFBレジスタに復帰する
- ③ サブルーチン (関数) からリターンする (RTS命令と同じ動作をする)

```
void main(void)
{
    int i = 1;
    char c = 2;
    func(i, c);
}
void func(int x, char y)
{
    int z;
}
```



```
_main:
    jsr $func
    add #1, SP
    :
    :
    :
$func:
    enter #4
    :
    :
    exitd
```

スタック補正コード

スタック補正コードをまとめる最適化(1/2)

スタック補正コード削除オプション “-Osp_ajust” の効果

スタック補正コード (add #n, SP) を削除し、ROM効率を上げる

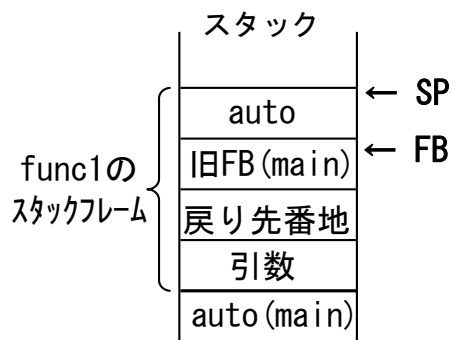
記述例

```
void main( void )
{
    auto変数の確保 ;

    func1(引数) ; ①
    func1(引数) ; ②
    func2(引数) ; ③
    func2(引数) ; ④
    func2(引数) ; ⑤
}
```

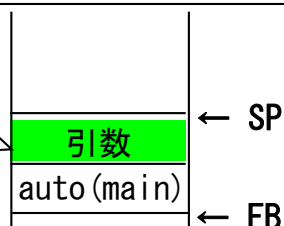
NC30では関数に引数を与えた場合、呼び出し命令(JSR命令)の直後に引数領域を解放するための“スタック補正コード(スタックポインタの加算処理)”が出力される。

① func1入口処理終了時(ENTER命令実行)



② func1リターン時(EXITD命令実行)

スタック補正コードを出力しないため、解放されなかったfunc1への引数領域

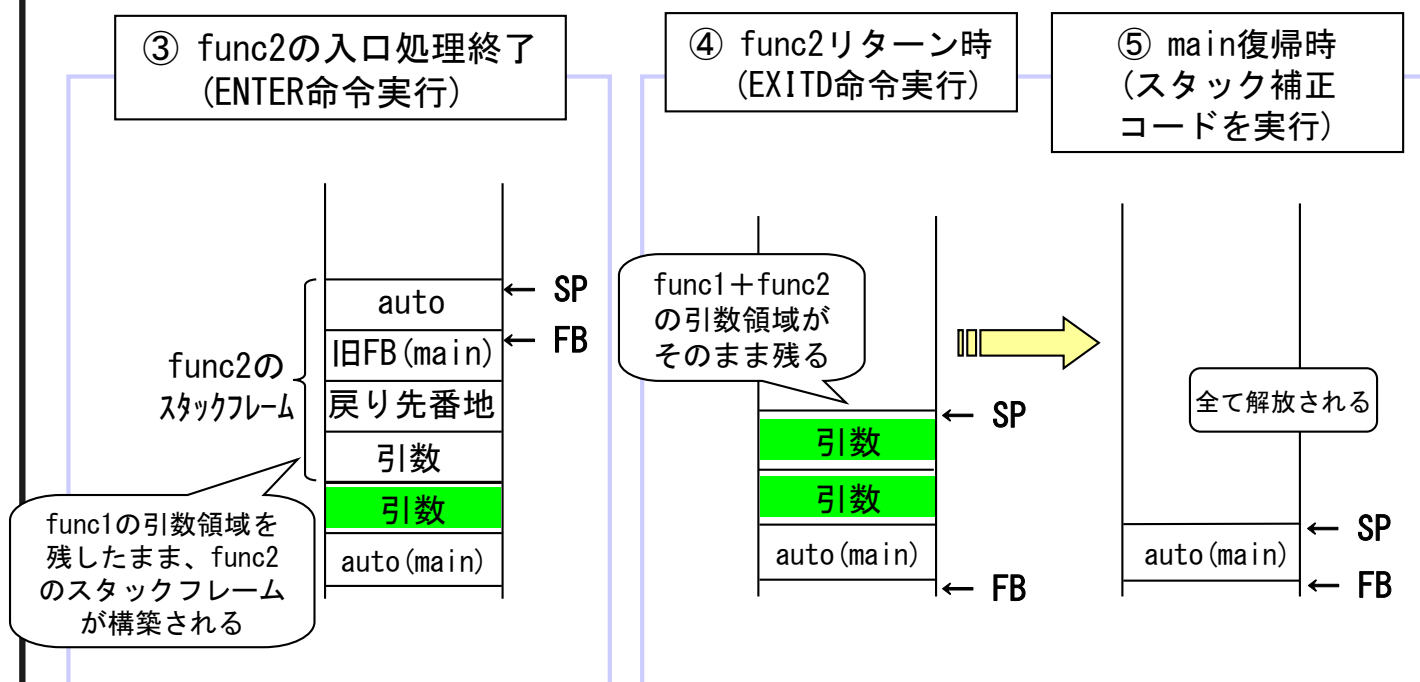


Code : R8Cコース

Date : Rev. 2. 20

Page: 18 of 11

スタック補正コードをまとめる最適化(2/2)



注意点

ROM容量は削減されるが、スタック使用量は多くなる可能性がある。

付録B.3 インラインアセンブル機能

インラインアセンブル機能

C言語プログラム中にアセンブリ言語を記述することができる機能

記述形式

```
asm( " 文字列 " );
```

” ”で囲まれた文字列はそのままアセンブラに渡されるため、アセンブラの記述形式に従うこと

使用例

```
void init(void);  
void main(void)  
{  
    init();  
    asm( " FSET I" ); /* 割り込みの許可 */  
    :  
    :  
}
```

Code : R8Cコース

Date : Rev. 2. 20

Page: 20 of 11

インラインアセンブル機能を使用する場合の注意事項

コンパイラはasm関数内のチェックを行っていないため、asm関数内で内部レジスタの値を不用意に変更しない。特にコンパイラが管理するレジスタを操作する場合は、push命令、pop命令を使用してレジスタの退避 / 復帰を行うこと。

アセンブリ言語の記述のほかに、以下の拡張機能を有する

- C言語プログラム中のauto変数のFBオフセット値の取得
- C言語プログラム中のregister変数のレジスタ名の取得
- C言語プログラム中のstatic／extern変数のシンボル名の取得

#pragmaによるインラインアセンブル記述

インラインアセンブルの指定

#pragma ASM／ENDASM

記述形式

#pragma ASM

アセンブリ言語記述

#pragma ENDASM

注意点

- ◆ 内部レジスタを操作する場合は、push・pop命令でレジスタ内容の退避・復帰を行うこと
- ◆ FBレジスタ及びB, Uフラグ等を変更する場合は、元の状態に復帰させてから#pragma ASM－ENDASM処理から抜けること

使用例

```
void func(void)
```

```
{
```

```
    int i, j;
```

```
    func2(i, j);
```

```
    :
```

```
#pragma ASM
```

```
    FCLR    I
```

```
    MOV.W   #OFFH, R0
```

```
    MOV.W   R2, R3
```

```
#pragma ENDASM
```

```
    :
```

#pragma ASMと#pragma ENDASM間の領域を直接アセンブリ言語ファイルに出力する

付録B. 4 アセンブラマクロ関数

アセンブラマクロ関数の利用

アセンブリ言語命令の一部をC言語の関数として記述することができる。
この機能を使用することにより、特定のアセンブリ言語命令を直接的にC言語のプログラム上に記述でき、ROM効率の向上が可能となる。

分類	アセンブラマクロ関数	機能
演算	DADD, DADC	10進加算(キャリーなし、キャリー付き)
	DSUB, DSBB	10進減算(ボローなし、ボロー付き)
	RMPA	積和演算
	DIV, DIVU, DIVX	除算(符号付き、符号なし)
	MOD	剰余算
転送	SMOVF, SMOVB, SSTR	ストリング転送
	MOVdir	4ビット転送
ローテート ／シフト	ROL, ROR, ROT	ローテート
	SHA, SHL	シフト(算術、論理)
その他	ABS, NEG, NOT	絶対値、2の補数、全ビット反転

アセンブルマクロ関数の使用例(1/3)

～ “RMPA命令” を使用した積和演算 ～

機能と書式

[機能]

初期値 : init、回数 : count、乗数の格納されている先頭アドレスをそれぞれ p1、p2 として積和演算を行い、結果を返す。

[書式]

- ① `static int rmpa_b(int init, unsigned int count, signed char *p1, signed char *p2);`
- ② `static long rmpa_w(long init, unsigned int count, int *p1, int *p2);`

～ “SMOVF命令” を使用したストリング転送 ～

機能と書式

[機能]

p1で示される転送元番地から、p2で示される転送先番地に、count回数分アドレスの上位方向へストリング転送を行う。

[書式]

- ① `static void smovf_b(char *p1, char *p2, unsigned int count);`
- ② `static void smovf_w(unsigned int *p1, unsigned int *p2, unsigned int count);`

アセンブラマクロ関数の使用例 (2/3)

～ “rmpa_w” を使用した積和演算 ～

記述例

```
#include <asmmacro.h>

long  ans;
int   str1[20];
int   str2[20];
void  func(void)
{
    ans = rmpa_w(0, 20, str1, str2);
}
```

アセンブラマクロ
関数の呼び出し

RMPA命令のアセンブラ
マクロの呼び出し

必ず “asmmacro.h”
をインクルードする

展開例

```
## ##ASM START
_rmpa_b .macro
    pushm    R3, A1, A0
    rmpa.b
    popm     R3, A1, A0
    .endm
## ##ASM END

.section program
.glob _func
_func:

    mov.w    #_str2, A1
    mov.w    #_str1, A0
    mov.w    #0014H, R3
    mov.w    #0000H, R0
    mov.w    #0000H, R2
    _rmpa_w
    mov.w    R0, _ans
    :
    :

.section bss_NE, DATA
.glob      _ans, _str1, _str2
_ans:      .blkb    2
_str1:     .blkb    40
_str2:     .blkb    40

.end
```

RMPA命令の
マクロ定義

アセンブラマクロ関数の使用例 (3/3)

～ “smovf_b” を使用したstring転送～

記述例

```
#include <asmmacro.h>
char src_str[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
char dest_str[8];
void func( void )
{
    smovf_b(src_str, dest_str, sizeof(src_str));
}
```

配列src_strの内容を
配列dest_strに転送

アセンブラマクロ
関数の呼び出し

展開例

```
## ##ASM START
_smovf_b .macro
    pushm    R1
    mov.w    #0, R1
    smovf.b
    popm     R1
    .endm
## ##ASM END

.section program
.glob _func
_func:
    mov.w    #0008H, R3
    mov.w    #_dest_str, A1
    mov.w    #_src_str, A0
    _smovf_b
    .
    .
    .

.section data_NE, DATA
.glob _src_str
_src_str: .blkb 8

.section data_NE1, ROMDATA
.byte 01H
.byte 02H
.byte 03H
.byte 04H
.byte 05H
.byte 06H
.byte 07H
.byte 08H
    .
    .
    .

.section bss_NE, DATA
.glob _dest_str
_dest_str: .blkb 8

.end
```

SMOVF命令の
マクロ定義

SMOVF命令の
アセンブラ
マクロ呼び出し

R8C マイコンコース コースノート

発行年月日 2005 年 3 月 28 日 Rev. 1. 00
2012 年 3 月 21 日 Rev. 2. 20

発行 ルネサスエレクトロニクス(株)
〒100-0004 東京都千代田区大手町 2-6-2

R8C マイコンコース テキスト



ルネサスエレクトロニクス株式会社
東京都千代田区大手町2-6-2 日本ビル 〒100-0004