お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

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ものですが、誤りがな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て、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。

RENESAS

アプリケーション・ノート

78K0/Kx2-L

低消費電力動作のための設定

この資料は，78K0/Kx2-Lマイコンの低消費電力モード，スタンバイ機能を使うことで，低消費電力でのマイコンの動作を実現するための設定内容を説明したものです。

対象デバイス

78K0/KY2-Lマイクロコントローラ
78K0/KA2-Lマイクロコントローラ
78K0/KB2-Lマイクロコントローラ
78K0/KC2-Lマイクロコントローラ

目　次

資料番号　U19612JJ1V0AN00（第1版）
発行年月　July 2009 NS

EEPROM**は，**NEC**エレクトロニクス株式会社の登録商標です。**

- **本資料に記載されている内容は**2009**年**7**月現在のもので，今後，予告なく変更することがあります。**
  量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品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，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエ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，ソフトウエ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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。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。当社製品をお客様の機器にご使用の際には，当社製品の不具合の結果として，生命，身体および財産に対する損害や社会的損害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を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。

  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

  特別水準：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

  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，生命維持のための装置またはシステム等

  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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E0710J

第1章　概　　要

このアプリケーション・ノートは，78K0/Kx2-L**マイクロコントローラの低消費電力動作のための設定について説明し，マイコンの消費電力を削減する方法を理解していただくことを目的としています。**

第1章では，主に低消費電力動作が求められる背景と，78K0/Kx2-L**マイクロコントローラの低消費電力のための機能の概要を説明します。**

対応デバイス一覧

78K0/KY2-L：μPD78F0550, μPD78F0551, μPD78F0552, μPD78F0555, μPD78F0556, μPD78F0557

78K0/KA2-L：μPD78F0560, μPD78F0561, μPD78F0562, μPD78F0565, μPD78F0566, μPD78F0567

78K0/KB2-L：μPD78F0571, μPD78F0572, μPD78F0573, μPD78F0576, μPD78F0577, μPD78F0578

78K0/KC2-L：μPD78F0581, μPD78F0582, μPD78F0583, μPD78F0586, μPD78F0587, μPD78F0588

1. 1　低消費電力の求められる背景と78K0/Kx2-Lコントローラの特長

携帯端末機器の普及やセキュリティ機器市場の拡大により，電池の長寿命化が求められるようになってきています。また環境保全への意識の高まりを受けて，半導体をはじめ，各種部品の低消費電力化が求められるようになりました。

78K0/Kx2-Lマイクロコントローラは，長期間にわたる電池動作やメンテナンスフリーが要求される分野の製品，省エネ製品のニーズに応えて開発されたものです。

以下に78K0/Kx2-Lマイクロコントローラで低消費電力による動作を実現する機能を紹介します。

(1)　スタンバイ機能

マイコンは，クロック信号を入力することで内部の回路が動作し，マイコンに電流が流れます。クロックを止めると内部の回路も止まり電流も流れなくなります。

スタンバイ機能とは，プログラムでスタンバイ命令を実行することでクロックの信号の流れを止めたり，発振そのものを止めたりすることのできる機能です。

外部イベント待ちなど，処理をしていない期間はスタンバイ命令を実行して，スタンバイ状態（クロックの信号の流れが止まっている，または発振そのものが止まっていてマイコンが停止している状態）にしておくことで，効率よく低消費電力化を図ることができます。

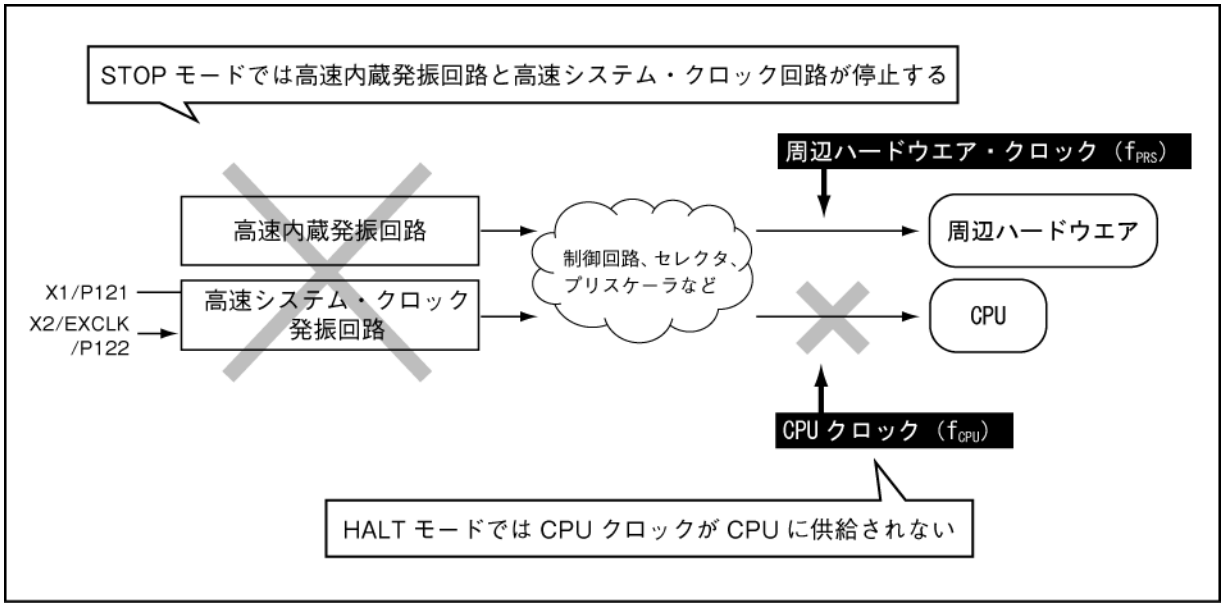
78K0/Kx2-Lマイクロコントローラのスタンバイ機能には，HALTモード，STOPモードの2つのスタンバイ状態があります。

STOPモード ： STOP命令の実行でSTOPモードになります。
クロックの発振回路そのものを停止させます。

HALTモード ： HALT命令の実行でHALTモードになります。
クロックの発振回路は動作させたままで，CPUへの供給を止めます。

消費電力は，通常の動作モード ＞ HALTモード ＞ STOPモードの順で小さくなります。

図1－1　スタンバイ機能とクロック発生回路

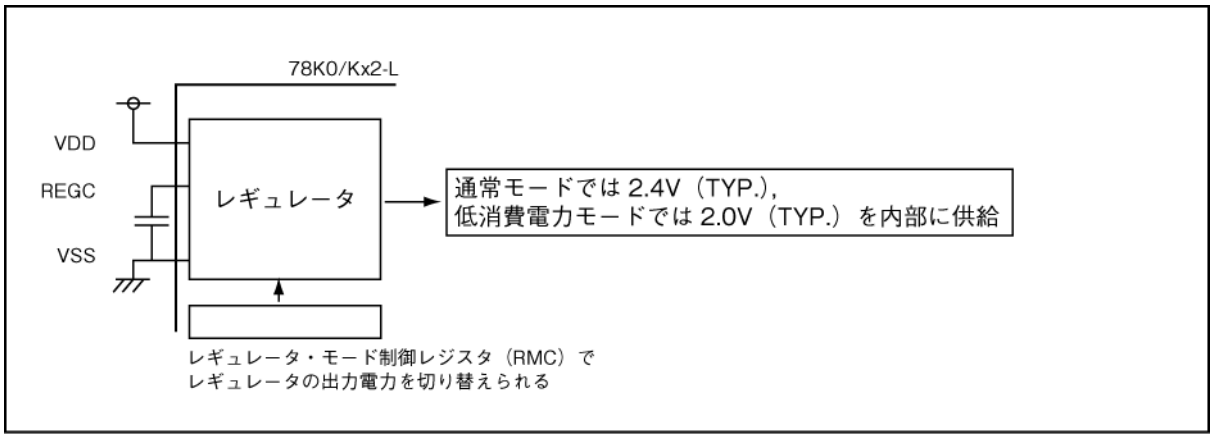


(2) レギュレータ設定による低消費電力モード

78K0/Kx2-Lマイクロコントローラは，外部からの電源電圧からマイコン内部に安定した電力を供給するためのレギュレータ回路を内蔵しています。このレギュレータ回路を制御するレギュレータ・モード・レジスタ（RMC）を設定することで，レギュレータがマイコン内部に出力する電圧を制御することができます。

レギュレータは通常モードでは2.4 V（TYP.），低消費電力モードでは2.0 V（TYP.）を供給します。レギュレータが供給する電圧よりも電源電圧が高い場合でも，消費電力は同じになります。

図1－2　レギュレータ



【コラム】その他の低消費電力のための対策-未使用のポート端子の処置

未使用の入出力ポートを入力状態，未接続にしておくと貫通電流が発生し，消費電力が増加する要因となります。ポート・モード・レジスタを出力に設定し，オープンにすることでこの問題を回避することができます。入力専用ポートは，プルアップかプルダウンすることにより貫通電流の発生を抑えることができます。

第2章　クロック発生回路とスタンバイ機能

スタンバイ機能は，クロック発生回路を制御する機能です。また，マイコンの消費電力は，クロックの動作／停止に加えて，周波数も大きく影響します。

クロック発生回路からのクロックの流れについてよく理解し，処理に最適な周波数クロックを選択し，必要のないクロックは停止させることで消費電力を抑えることができます。

この章では，78K0/Kx2-Lマイクロコントローラのクロック発生回路とスタンバイ機能について説明します。

2. 1　クロック発生回路

(1)　リセット解除後のクロック発生回路

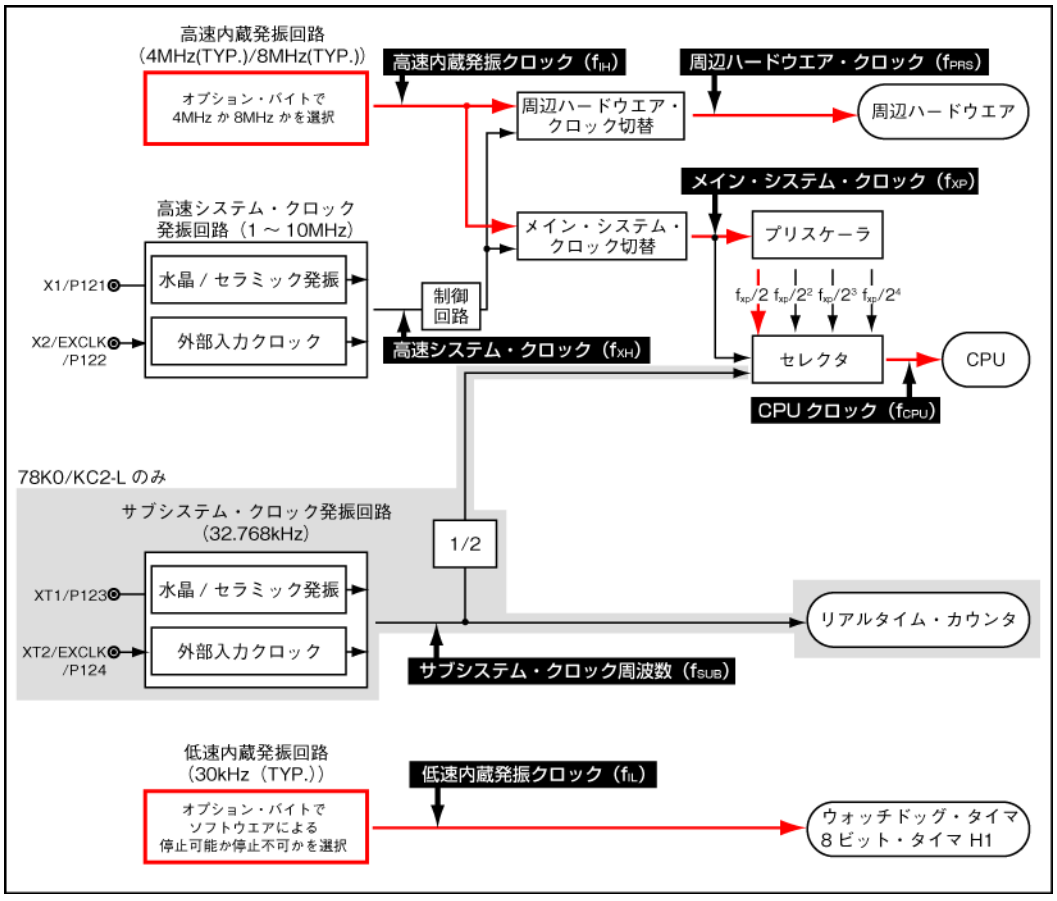
リセット解除後は，高速内蔵発振回路（8 MHz（TYP.）/4 MHz（TYP.）かをオプション・バイトの設定によって選択できます），低速内蔵発振回路（30 kHz（TYP.））が動作します。

高速内蔵発振回路はメイン・システム・クロックに供給され，プリスケーラによって2分周されたメイン・システム・クロック（$f_{XP}/2$）がCPUクロック（f_{CPU}）に供給されます。

また高速内蔵発振回路は，周辺ハードウエアのクロック・ソースとなります。

低速内蔵発振回路（30 kHz（TYP.））はウォッチドッグ・タイマとタイマH1のクロック・ソースとなります。

図2－1　リセット解除後のクロック発生回路（赤枠，線の箇所が動作）



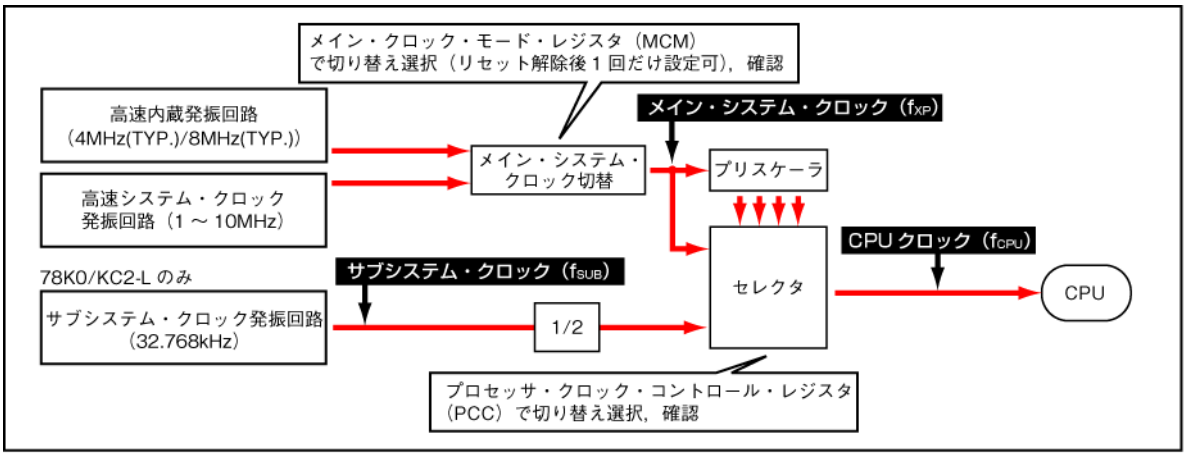
(2)　CPUクロック（f_{CPU}）の流れとクロック・ソースの切り替え

CPUクロック（f_{CPU}）は，CPUへ供給されて命令の読み出し／実行，内部レジスタやメモリの読み出し／書き込みのためのクロック信号となります。

CPUクロック（f_{CPU}）の供給元は，メイン・システム・クロック（分周なし，プリスケーラによる1/2, 1/4, 1/8, 1/16分周），サブシステム・クロックの2分周（f_{SUB}/2）から選択できます。プロセッサ・クロック・コントロール・レジスタ（PCC）で切り替えます。

メイン・システム・クロック（f_{XP}）の供給元は，高速内蔵発振回路から高速システム・クロック発振回路に切り替えることができます。切り替えはリセット解除後1回だけ可能です。メイン・クロック・モード・レジスタ（MCM）の設定で切り替えます。

図2－2　各クロック発生回路からCPUクロック（f_{CPU}）への流れ（赤枠部分）と切り替えのためのレジスタ



メイン・クロック・モード・レジスタ（MCM）　リセット時：0000 0000B

0	0	0	0	0	XSEL	MCS	MCM0

XSEL	MCM0	メイン・システム・クロック（f_{XP}）のクロック・ソース	周辺ハードウエア・クロック（f_{PRS}）のクロック・ソース
0	0	高速内蔵発振クロック（f_{IH}）（デフォルト）	高速内蔵発振クロック（f_{IH}）（デフォルト）
0	1		
1	0		高速システム・クロック（f_{XH}）
1	1	高速システム・クロック（f_{XH}）	

MCS	メイン・システム・クロック（f_{XP}）のステータス（クロック・ソース）
0	高速内蔵発振クロック（f_{IH}）（デフォルト）
1	高速システム・クロック（f_{XH}）

プロセッサ・クロック・コントロール・レジスタ（PCC）　リセット時：0000 0001B

0	★XSTART	★CLS	★CSS	0	PCC2	PCC1	PCC0

★ 78K0/KC2-L のみ

CSS	PCC2	PCC1	PCC0	CPU クロック（f_{CPU}）のクロック・ソース
1	0	0	0	メイン・システム・クロック分周なし（f_{XP}）
	0	0	1	メイン・システム・クロック２分周（f_{XP}/2）（デフォルト）
	0	1	0	メイン・システム・クロック 2^2 分周（$f_{XP}/2^2$）
	0	1	1	メイン・システム・クロック 2^3 分周（$f_{XP}/2^3$）
	1	0	0	メイン・システム・クロック 2^4 分周（$f_{XP}/2^4$）

上記以外の設定禁止

CSS	PCC2	PCC1	PCC0	CPU クロック（f_{CPU}）のクロック・ソース
0	0	0	0	サブシステム・クロック２分周（f_{SUB}/2）
	0	0	1	
	0	1	0	
	0	1	1	
	1	0	0	

上記以外の設定禁止

CLS	CPU クロック（f_{CPU}）のステータス（クロック・ソース）
0	メイン・システム・クロック（f_{XP}）（デフォルト）
1	サブシステム・クロック２分周（f_{SUB}/2）

メイン・システム・クロック（f_{XP}），CPUクロック（f_{CPU}）の供給元を切り替えた場合，数クロックは切り替え前のクロックで動作します。

メイン・システム・クロックの場合は，メイン・クロック・モード・レジスタ（MCM）のMCSビットで完全に切り替わったことを確認できます。

CPUクロック（f_{CPU}）をメイン・システム・クロック（f_{XP}）からサブシステム・クロックの2分周（$f_{SUB}/2$）に切り替えた場合，またはその逆の場合は，プロセッサ・クロック・コントロール・レジスタ（PCC）のCLSビットで完全に切り替わったことを確認できます。

プロセッサ・クロック・コントロール・レジスタ（PCC）の設定を変更してメイン・システム・クロックの分周比を切り替えた場合は，以下の時間をソフトウエアでウエイトする必要があります。

表2－1　メイン・システム・クロックの分周比を切り替えた場合のCPUクロックの切り替え最大時間

切り替え前の分周比	切り替え後の分周比				
	分周なし	1/2	$1/2^2$	$1/2^3$	$1/2^4$
分周なし		16クロック	16クロック	16クロック	16クロック
1/2	8クロック		8クロック	8クロック	8クロック
$1/2^2$	4クロック	4クロック		4クロック	4クロック
$1/2^3$	2クロック	2クロック	2クロック		2クロック
$1/2^4$	1クロック	1クロック	1クロック	1クロック	

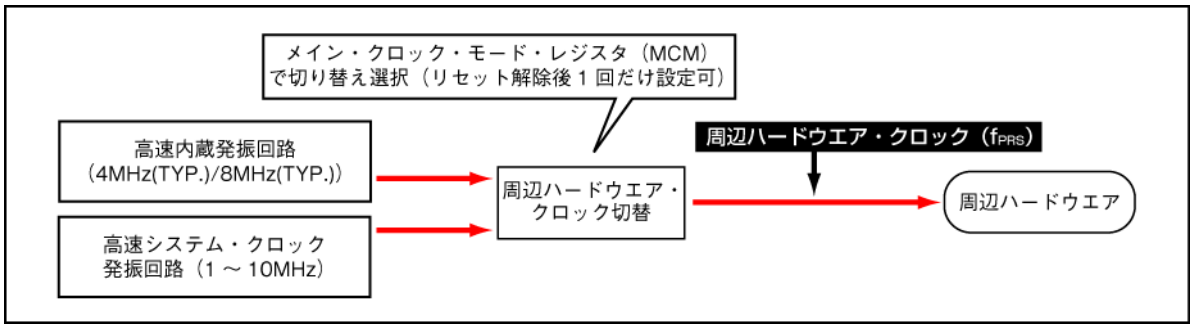
備考　クロック数は，切り替え前のCPUクロック（f_{CPU}）のクロック数です

(3)　周辺ハードウエア・クロックの流れとクロック・ソースの切り替え

周辺ハードウエア・クロックは，周辺ハードウエアへ供給され，各周辺ハードウエアの動作クロックとなります。

周辺ハードウエア・クロックの供給元は，高速内蔵発振回路から高速システム・クロック発振回路に切り替えることができます。切り替えはリセット解除後1回だけ可能です。メイン・クロック・モード・レジスタ（MCM）の設定で切り替えます。（メイン・クロック・モード・レジスタ（MCM）の詳細については図2－2を参照してください。）

図2－3　各クロック発生回路から周辺ハードウエア・クロック（f_{PRS}）への流れ（赤線部分）



(4)　各クロック発生回路の停止と動作開始

どこにも供給していないクロック発生回路，または供給先の機能を使用しない場合はクロック発生回路を停止することができます（低速内蔵発振回路はオプション・バイトで「ソフトウエアによる停止可能」に設定している場合に停止できます）。各クロック発生回路を停止する前に，動作中の機能（CPU，周辺ハードウエア）へ供給されていないか確認する必要があります。

停止しているクロック発生回路の動作を開始したあとは，CPUクロック，周辺ハードウエア・クロック，リアルタイム・カウンタ・クロックへ供給する前に，発振安定時間をウエイトしてから供給する必要があります。

① 高速内蔵発振回路，低速内蔵発振回路

各内蔵発振回路の発振／停止は，内蔵発振モード・レジスタ（RCM）で設定します。高速内蔵発振回路の発振安定は，RSTSビットで確認することができます。

図2－4　内蔵発振モード・レジスタ

内蔵発振モード・レジスタ（RCM）　リセット時：1000 0000B

RSTS	0	0	0	0	0	LSRSTOP	RSTOP

LSRSTOP	低速内蔵発振器の発振 / 停止
0	低速内蔵発振器の発振（デフォルト）
1	低速内蔵発振器の停止

RSTOP	高速内蔵発振器の発振 / 動作
0	高速内蔵発振器の発振（デフォルト）
1	高速内蔵発振器の停止

RSTS	高速内蔵発振器のステータス
0	高速内蔵発振器の発振精度安定待ち中
1	高速内蔵発振器安定動作中（デフォルト）

注意　リセット直後はRSTS = 0ですが，高速内蔵発振器の発振精度安定待ち後に1に切り替わります。

② 高速システム・クロック発振回路

高速システム・クロック発振回路の停止，動作は，メインOSCコントロール・レジスタ（MOC）のMSTOPビットで設定します。

X1発振モードの場合，発振安定時間のウエイトは，X1発振安定時間カウンタで計測することができます。安定時間を発振安定時間選択レジスタ（OSTS）に設定し，発振安定時間カウンタ状態レジスタ（OSTC）で設定した時間が経過したかどうかを確認できます。設定時間はお使いの発振子にあわせて十分な時間を設定してください（X1発振安定時間カウンタは，設定された時間を計測するだけで，X1発振の安定発振を確認しているわけではありません）。

発振安定時間選択レジスタ（OSTS）はリセット解除後，最も長い計測時間（2^{16}/fx）の設定となります。

図2－5　メインOSCコントロール・レジスタ

メイン OSC コントロール・レジスタ（MOC）　リセット時：1000 0000B

MSTOP	0	0	0	0	0	0	0

MSTOP	高速システム・クロックの動作制御	
	X1 発振モード時	外部クロック入力モード時
0	X1 発振回路動作	EXCLK 端子からの外部クロック有効
1	X1 発振回路停止（デフォルト）	EXCLK 端子からの外部クロック無効（デフォルト）

図2－6　X1発振安定時間カウンタの各レジスタ

発振安定時間選択レジスタ（OSTS）　リセット時：0000 0101B

0	0	0	0	0	OSTS2	OSTS1	OSTS0

OSTS2	OSTS1	OSTS0	発振安定時間の選択	f_X = 10MHz 時
0	0	1	$2^{11}/f_X$	204.8 μs
0	1	0	$2^{13}/f_X$	819.2 μs
0	1	1	$2^{14}/f_X$	1.64ms
1	0	0	$2^{15}/f_X$	3.27ms
1	0	1	$2^{16}/f_X$（デフォルト）	6.55ms

上記以外の設定禁止

発振安定時間カウンタ状態レジスタ（OSTC）　リセット時：0000 0000B

0	0	0	MOST11	MOST13	MOST14	MOST15	MOST16

MOST11	MOST13	MOST14	MOST15	MOST16	発振安定時間の選択	f_X = 10MHz 時
1	0	0	0	0	$2^{11}/f_X$ 以上	204.8 μs 以上
1	1	0	0	0	$2^{13}/f_X$ 以上	819.2 μs 以上
1	1	1	0	0	$2^{14}/f_X$ 以上	1.64ms 以上
1	1	1	1	0	$2^{15}/f_X$ 以上	3.27ms 以上
1	1	1	1	1	$2^{16}/f_X$ 以上	6.55ms 以上

→ 時間経過にしたがって 1 になる

③ サブシステム・クロック発振回路（78K0/KC2-Lのみ**）**

サブシステム・クロック発振回路の停止，動作はプロセッサ・クロック・コントロール・レジスタ（PCC），クロック動作モード選択レジスタ（OSCCTL）の各ビットで設定します。

XT1発振モードで動作開始後は,お使いの発振子にあわせて安定時間をソフトウエアでウエイトする必要があります。

図2－7　サブシステム・クロック発振回路の各レジスタ

プロセッサ・クロック・コントロール・レジスタ（PCC）　リセット時：0000 0001B

0	★XSTART	★CLS	★CSS	0	PCC2	PCC1	PCC0

★ 78K0/KC2-L のみ

クロック動作モード選択レジスタ（OSCCTL）　リセット時：0000 0000B

EXCLK	OSCSEL	★EXCLKS	★OSCSELS	0	★RSWOSC	★AMPHXT	0

★ 78K0/KC2-L のみ

XSTART	EXCLKS	OSCSELS	サブシステム・クロックの動作モード
0	0	1	XT1 発振モード（P123/XT1、P124/XT2 に水晶 / セラミック発振子が接続されていること）
1	×	×	
0	1	1	外部クロック入力モード（P124/EXCLKS に外部クロックが入力されていること）

サブシステム・クロックの停止は OSCSELS を 0 に設定してください（デフォルトは停止）　×：don't care

RSWOSC	AMPHXT	XT1 発振回路の発振モード選択
0	0	低消費発振（デフォルト）
0	1	通常費発振
1	×	超低消費発振

×：don't care

CLS	CPU クロック（f_{CPU}）のステータス（クロック・ソース）
0	メイン・システム・クロック（f_{XP}）で動作
1	サブシステム・クロック（f_{SUB}）で動作

2. 2　スタンバイ機能

スタンバイ機能では，各モードへ移行する命令（HALT命令，STOP命令）を実行することで，プログラムの任意のタイミングでクロックを制御して消費電力を抑えることができます。各モードから通常の動作モードへは，マスクされていない割り込み要求の信号かリセット信号の発生で復帰します。

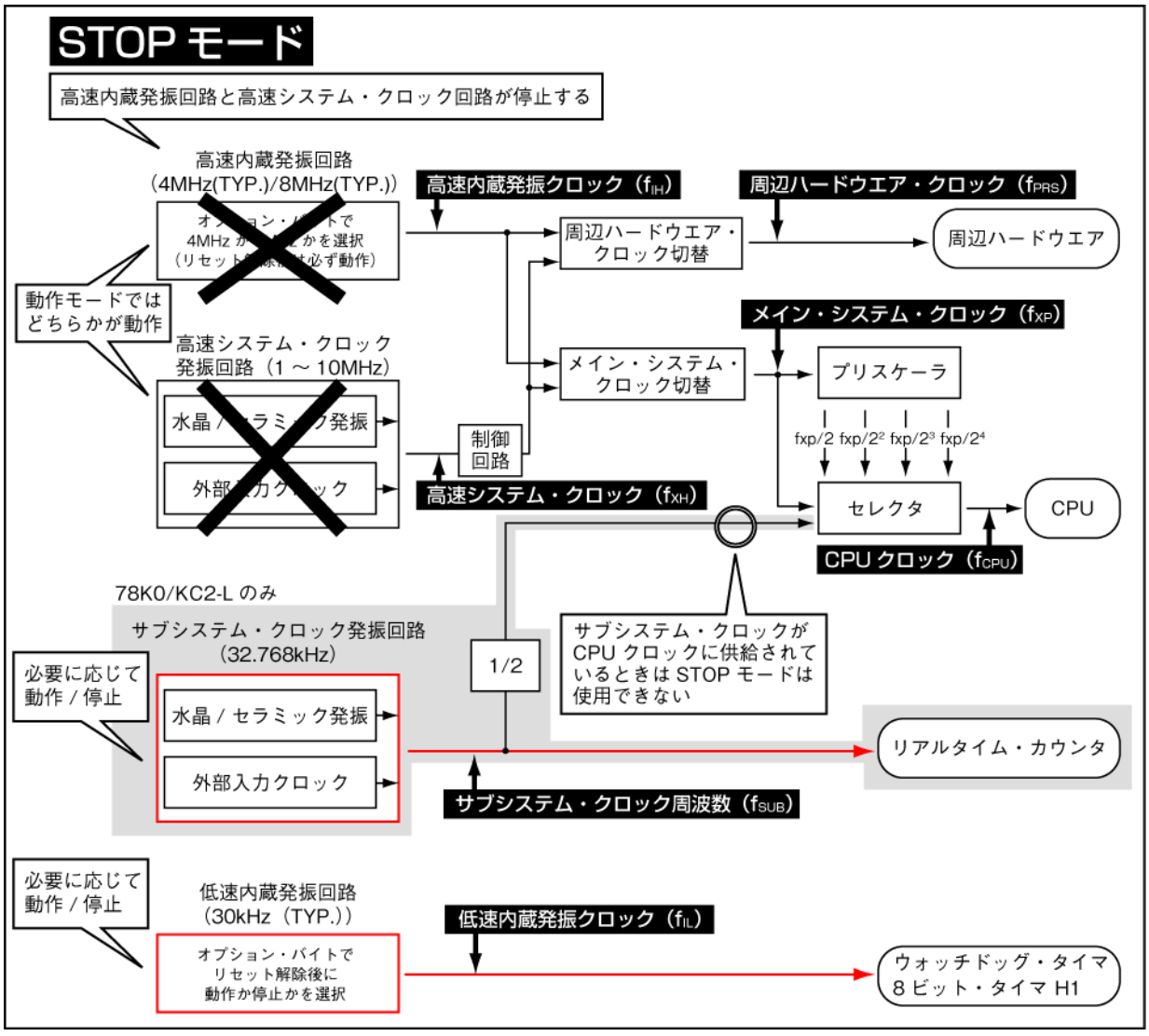
HALTモードはSTOPモードほどの動作電流の低減はできませんが，クロックが発振しているので，復帰にかかる時間は短くなります。STOPモードはHALTモードに比べると動作電流は低減できますが，クロックの発振回路が停止するため，復帰にかかる時間は長くなります。

どちらのモードでスタンバイするかは，プログラムでの間欠動作の頻度に伴う遅延や消費電力の要求仕様で決めます。動作電流の比較については「2. 3　スタンバイ機能のトータル電流の比較」，復帰時間の比較については「2. 4 スタンバイ機能の割り込みによる復帰時間の比較」を参照してください。

(1)　STOPモード

STOP命令実行時は高速内蔵発振回路，高速システム・クロック発振回路が停止します。サブシステム・クロック発振回路，低速内蔵発振回路は動作をしている場合は動作し続けます。

図2－8　STOPモード時のクロック発生回路



① CPUクロック（f_{CPU}）の条件

STOPモードはメイン・システム・クロック（f_{XP}）がCPUクロック（f_{CPU}）に選択されている場合のみ移行できます。サブシステム・クロック（f_{SUB}：78K0/KC2-Lのみ）はSTOP命令を実行しても停止しないので，CPUクロックに選択されている場合はSTOPモードへ移行できません。

② CPUと周辺ハードウエアの状態

- 高速内蔵発振回路（f_{IH}），高速システム・クロック発振回路（f_{XH}）が停止するのでCPUは停止します
- 周辺ハードウエア・クロック（f_{PRS}）をソースとする周辺ハードウエアの各機能は停止します。ただし，周辺ハードウエアで端子からの外部クロック入力，低速内蔵発振クロック（f_{IL}），サブシステム・クロック（f_{SUB}）をソースとする機能はSTOP命令の実行直前の状態を維持します（動作している場合は継続して動作します）。STOP命令を実行する前に，周辺ハードウエア・クロック（f_{PRS}）をソースとする周辺ハードウエアが動作している場合は必ず停止させる必要があります。
- CPUの各レジスタ，RAM，周辺ハードウエアの各レジスタはSTOP命令実行前の状態を保持します。
- オプション・バイトで「低速内蔵発振器　ソフトウエアにより停止可能」に設定している場合は，ウォッチドッグ・タイマへの低速内蔵発振回路からのクロック供給が停止します。

表2－2　STOPモード時の周辺ハードウエア

周辺ハードウエア		STOPモード時の状況
16ビット・タイマ／イベント・カウンタ00		動作停止
8ビット・タイマ／イベント・カウンタ	50	カウント・クロックにTI50選択時のみ動作可能
	51	カウント・クロックにTI51選択時のみ動作可能
8ビット・タイマ	H0	8ビット・タイマ／イベント・カウンタ50動作時，カウント・クロックにTM50出力選択時のみ動作可能
	H1	カウント・クロックにf_{IL}，$f_{IL}/2^6$，$f_{IL}/2^{15}$選択時のみ動作可能
リアルタイム・カウンタ		動作可能
ウォッチドッグ・タイマ		動作可能。ただしオプション・バイトで「低速内蔵発振器　ソフトウエアにより停止可能」に設定した場合はウォッチドッグ・タイマへのクロック供給停止
クロック出力		カウント・クロックにサブシステム・クロック選択時のみ動作可能
A/Dコンバータ		動作停止
オペアンプ0，1		動作停止
シリアル・インターフェース	UART6	8ビット・タイマ／イベント・カウンタ50動作時，シリアル・クロックにTM50出力選択時のみ動作可能
	CSI10，CSI11	シリアル・クロックに外部クロック選択時のみ動作可能
	IICA	アドレス一致によるウエイク・アップ（割り込み信号発生）可能
キー割り込み		動作可能
パワーオン・クリア機能		
低電圧検出機能		
外部割り込み		

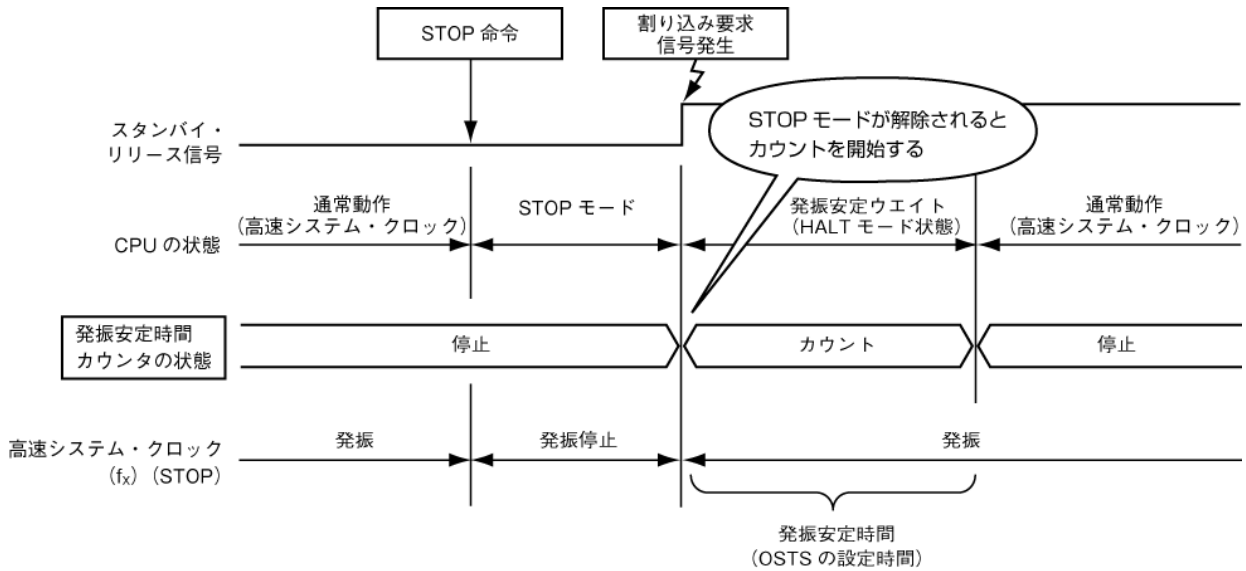
③ STOPモードからの復帰時の発振安定時間カウンタ動作について

クロック発生回路には，X1発振子の安定時間を計測するカウンタがあります。高速システム・クロックのX1発振子を発振させる場合，発振安定時間選択レジスタ（OSTS）で時間を設定して，発振安定時間カウンタ状態レジスタ（OSTC）を読み出すことで発振安定時間を計測することができます。

OSTCレジスタは，CPUクロック（f_{CPU}）が高速内蔵発振回路かサブシステム・クロック発振回路から供給されている状態で，X1発振回路の発振を開始した（MSTOP = 0）タイミングでカウントを開始します。

ただし，CPUクロック（f_{CPU}）が高速システム・クロックのX1発振子から供給されている状態で，STOP命令が実行され，マスクされていない割り込み要求信号の発生で復帰した場合のみ，復帰直後からカウントを開始します。カウントの間はCPUクロック（f_{CPU}）にクロック・ソースが供給されないHALT状態となり，OSTSレジスタに設定された時間を計測したあと動作モード（通常の状態）となります。したがってSTOP命令実行より前に発振安定時間選択レジスタ（OSTS）に適切な値が設定されているようにしてください。

図2－9　STOP命令解除時のX1発振安定時間カウンタの動作



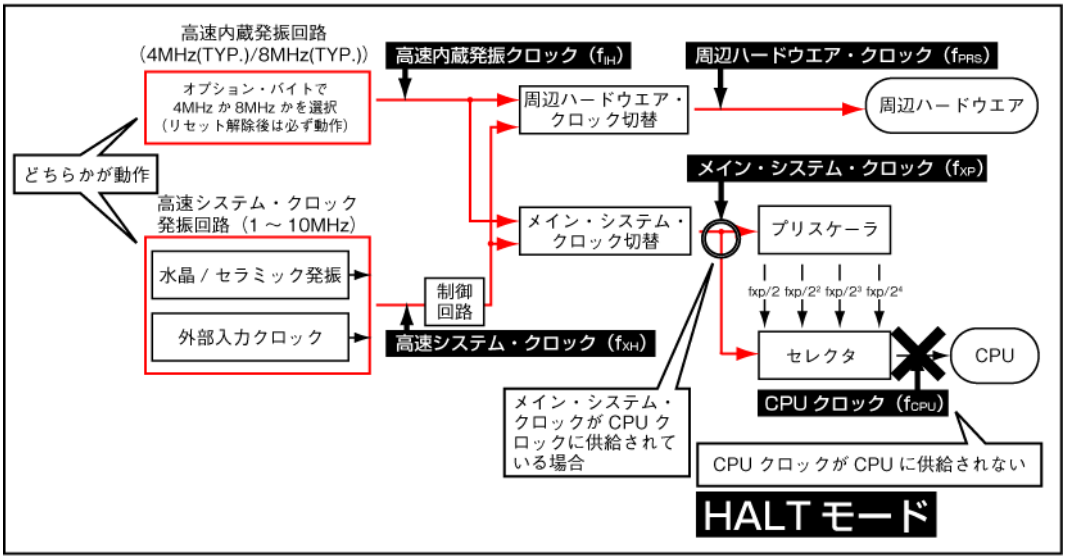
(2)　HALTモード

HALT命令実行時は，各発振回路が動作をしている場合は動作を継続し，CPUへのCPUクロック（f_{CPU}）の供給が止まります。

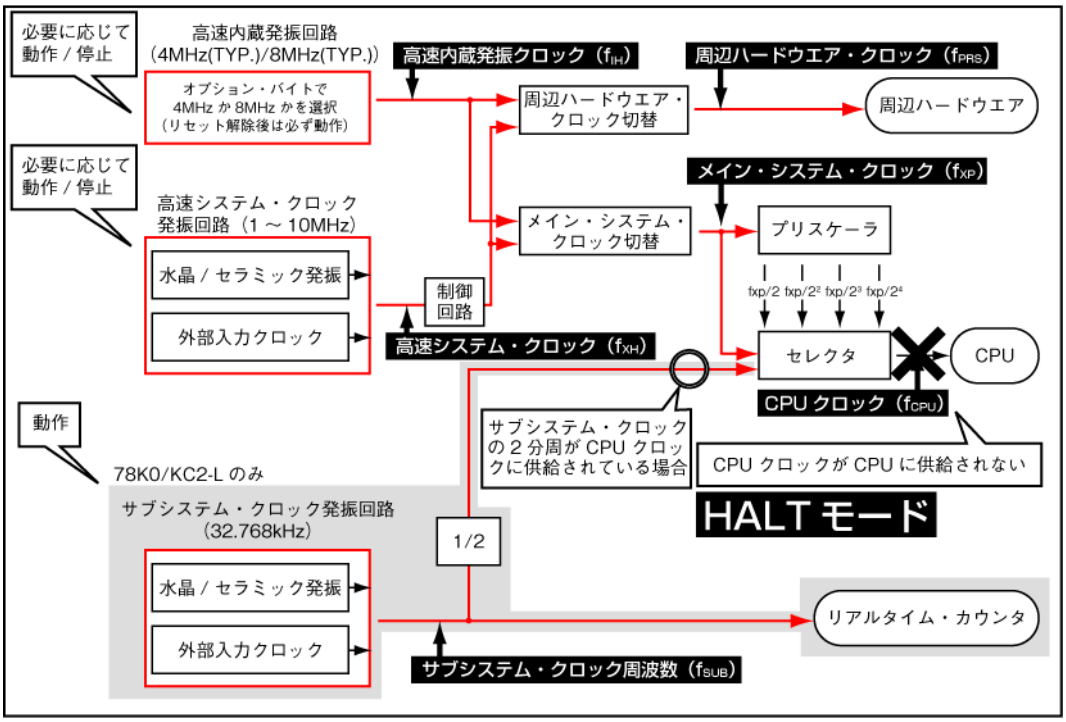
HALTモードは，メイン・システム・クロック（f_{XP}），サブシステム・クロックの2分周（$f_{SUB}/2$）のいずれがCPUクロック（f_{CPU}）に選択されていても移行できます。

図2－10　HALTモード時のクロック発生回路（低速内蔵発振回路は省略しています）

メイン・システム・クロックが CPU クロックの場合



サブシステム・クロックの 2 分周が CPU クロックの場合（78K0/KC2-L のみ）



① CPUクロックの条件

HLATモードはメイン・システム・クロック（f_{XP}），サブシステム・クロックの2分周（$f_{SUB}/2$）のどちらがCPUクロック（f_{CPU}）に選択されていても移行できます。

サブシステム・クロックの2分周（$f_{SUB}/2$）がCPUクロック（f_{CPU}）に選択されている場合は，高速内蔵発振回路，高速システム発振回路ともに停止させることで，HALTモード中の消費電力をさらに抑えることができます（表2－5参照）。

② CPUと周辺ハードウエアの状態

- CPUへCPUクロック（f_{CPU}）が供給されないのでCPUは停止します。
- 周辺ハードウエアへは周辺ハードウエア・クロック（f_{PRS}）が供給されるのでHALT命令実行前の状態をそのまま維持します（動作している場合は継続して動作します）。
- CPUの各レジスタ，RAM，周辺ハードウエアの各レジスタはHALT命令実行前の状態を保持します。
- オプション・バイトで「低速内蔵発振器　ソフトウエアにより停止可能」に設定している場合は，ウォッチドッグ・タイマへの低速内蔵発振回路からのクロック供給が停止します。

(3) マスクされていない割り込みによる復帰について（STOPモード，HALTモード共通）

マスクされていない割り込みの要求信号による復帰時，割り込み許可フラグ（IE）が許可の場合（IE = 1）は，割り込み処理を行い，禁止の場合（IE = 0）割り込み処理は行わないで次の命令を実行します。この場合，割り込み要求信号は自動的にクリアされずに保持されるので割り込みが許可された時点（IE = 1）で割り込み処理を実行します（割り込みベクタに該当するアドレスにプログラムが存在しない場合は暴走します）。

マスクされていない割り込みによるSTOPモード，HALTモードの復帰時に割り込み処理を行わない場合，必ず復帰後に割り込み許可（IE = 1）に設定する前に割り込み要求フラグをクリアする必要があります。

表2－3　割り込み要求に対する復帰時の動作（STOP, HALTモード）

解除ソース	MKXX （マスク・フラグ）	PRXXX （優先順位指定フラグ）	IE	ISP	動作
マスカブル割り込み要求	0（許可）	0（高）	0	×	次アドレス実行 （割り込み要求フラグは保持）
			1	×	割り込み処理実行
		1（低）	0	1	次アドレス実行 （割り込み要求フラグは保持）
			×	0	
			1	1	割り込み処理実行
	1（禁止）	×	×	×	STOP, HALTモード保持
リセット	-	-	×	×	リセット処理

【コラム】 周辺ハードウエアのイベント待ちのHALTモード使用について

A/Dコンバータの変換待ちなど，周辺ハードウエアのイベント発生をソフトウエアのポーリングで検知している場合は，代わりにHALTモードの使用を検討するとよいでしょう。割り込みによるHALTモードからの復帰時間の方が，ポーリングでの検知よりも処理が早い場合があります。また，ポーリングはマイコンが命令を実行し続けるので，HALTモードで待機した方が消費電力は小さくなります。

2. 3　スタンバイ機能のトータル電流の比較

以下に各条件での通常動作，HALTモード，STOPモードでの内部電源（V_{DD}，AV_{REF}）に流れるトータル電流（レギュレータ設定別）を示します。

表2－4　CPUクロック（f_{CPU}）が高速内蔵発振クロック（f_{IH}），高速システム・クロック（f_{XH}）から供給

動作条件	動作モード（通常動作）	HALTモード	STOPモード
f_{IH} = 4 MHz（高速内蔵発振クロック）， V_{DD} = 3.0 V RMC = 56H（低消費電力モード）	0.5 mA（TYP.）， 1.4 mA（MAX.）	0.2 mA（TYP.）， 0.5 mA（MAX.）	0.3 μA（TYP.）， 5.5 μA（MAX.） （クロック発生回路は停止しているので周波数は条件に含まれません）
f_{IH} = 8 MHz（高速内蔵発振クロック）， V_{DD} = 5.0 V RMC = 00H	1.3 mA（TYP.）， 2.5 mA（MAX.）	0.3 mA（TYP.）， 1.2 mA（MAX.）	-
f_{XH} = 10 MHz（外部入力クロック）， V_{DD} = 5.0 V RMC = 00H	1.6 mA（TYP.）， 2.8 mA（MAX.）	0.4 mA（TYP.）， 1.3 mA（MAX.）	-
f_{XH} = 10 MHz（水晶／セラミック発振）， V_{DD} = 5.0 V RMC = 00H	2.3 mA（TYP.）， 3.9 mA（MAX.）	1.0 mA（TYP.）， 2.4 mA（MAX.）	-

表2－5　CPUクロック（f_{CPU}）がサブシステム・クロック2分周（f_{SUB}/2）から供給（78K0/KC2-Lのみ）

動作条件	動作モード（通常動作）	HALTモード	STOPモード
f_{SUB} = 32.768 MHz（水晶／セラミック発振）， V_{DD} = 3.0 V RMC = 56H（低消費電力モード）	3 μA（TYP.）， 9.7 μA（MAX.）	0.8 μA（TYP.）， 6.7 μA（MAX.）	0.3 μA（TYP.）， 5.5 μA（MAX.） （クロック発生回路は停止しているので周波数は条件に含まれません）

注意1. 表2－4, 2－5は「78K0/Kx2-L ユーザーズ・マニュアル」の「第28章　電気的特性」より抜粋したものです。

2. 表2－4, 2－5の電流値は，入力端子をV_{DD}またはV_{SS}に固定した状態での入力リーク電流を含みます。ただしポートのプルアップ抵抗，プルダウン抵抗と出力電流は含みません。

3. 通常モード，HALTモード時の電流は，CPUに供給されるクロック発振以外の発振回路に流れる電流は含みません。また，LVI回路，A/Dコンバータ，オペアンプ，ウォッチドッグ・タイマ，リアルタイム・カウンタ，8ビットタイマH1（カウント・クロックに30 kHz低速内蔵発振クロックを使用）に流れる電流は含みません。

4. STOPモード時の電流は，LVI回路，A/Dコンバータ，オペアンプ，ウォッチドッグ・タイマ，リアルタイム・カウンタ，8ビットタイマH1（カウント・クロックに30 kHz低速内蔵発振クロックを使用）に流れる電流は含みません。

5. STOPモードの電流値は，V_{DD} = 3.0 V，RMC = 56Hの条件のものです。クロック発生回路は停止しますので周波数は条件に含まれません。

2. 4　スタンバイ機能の割り込みによる復帰時間の比較

マスクされていないマスカブル割り込みの要求信号による復帰の場合は，HALTモードからとSTOPモードからでは復帰時間が異なります。これはHALT状態では高速内蔵発振回路，高速システム・クロック発振回路は動作しており，STOP状態では停止しているためです。

割り込みの要求信号が発生すると，スタンバイ・リリース信号（内部信号：H = リリース状態）がリリースとなり，各モードから復帰します。

STOPモードからの復帰は，CPUクロック（f_{CPU}）にどのクロック（高速内蔵発振回路（f_{IH}），高速システム・クロック発振回路（f_X，f_{EXCLK}とも），サブクロック・システム発振回路の2分周（$f_{SUB}/2$））が供給されているかで復帰時間が異なります（高速システム・クロック発振回路のX1発振子（f_X）が供給されている場合が一番長くなります）。HALTモードからの復帰は，CPUクロック（f_{CPU}）にどのクロックが供給されている場合も同じ復帰時間となります。

表中の各数値は，特に指定のない場合はRMC = 00H（2.4 V供給固定）の場合となります。

(1)　CPUクロック（f_{CPU}）が高速内蔵発振クロック（f_{IH}）から供給されている場合

図2－11　割り込み要求信号による復帰（f_{CPU} ← f_{IH}）

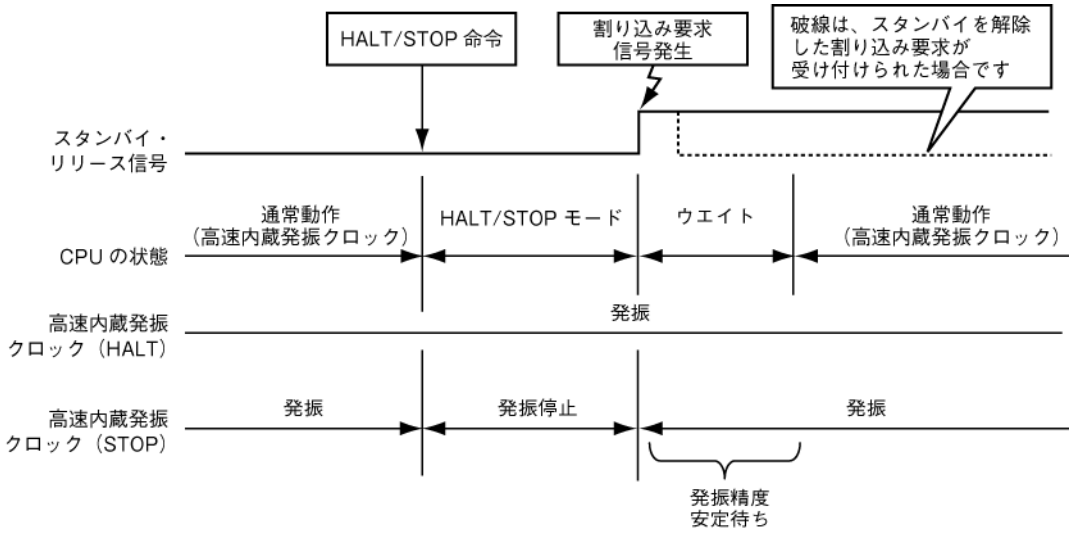


表2－6　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{IH}）

要因		HALTモード（f_{CPU} = 8 MHzの場合）	STOPモード（f_{CPU} = 8 MHzの場合）
ウエイト	ベクタ割り込み処理を行う場合	11～12クロック（1.375～1.5 μs）	17～18クロック（2.125～2.25 μs）
	ベクタ割り込み処理を行わない場合	4～5クロック（1.375～1.5 μs）	11～12クロック（1.375～1.5 μs）
発振精度安定待ち		-	102～407 μs（RMC = 00H） 120～481 μs（RMC = 56H）

(2)　CPUクロック（f_{CPU}）が高速システム・クロックのX1発振子（f_X）から供給されている場合

CPUクロック（f_{CPU}）をX1の水晶／セラミック発振子から供給していて，STOPモードから復帰する場合は，発振子にあわせてユーザがOSTSレジスタに設定した発振安定時間をウエイトします。この間はCPUクロック（f_{CPU}）にクロックが供給されないHALTモード状態となります。

図2－12　割り込み要求信号による復帰（f_{CPU} ← f_{XH} ← f_X）

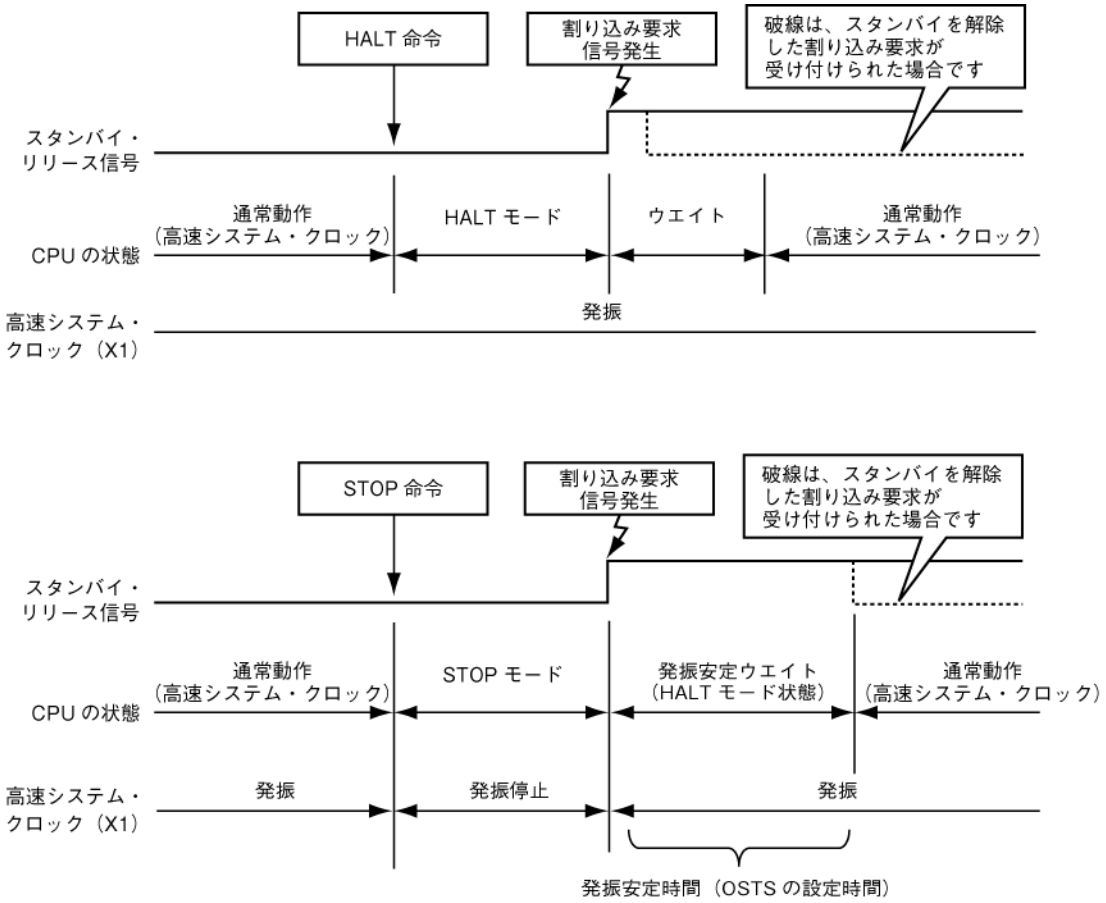


表2－7　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{XH} ← f_X）

要因		HALTモード（f_{CPU} = 10 MHzの場合）	STOPモード（f_{CPU} = 10 MHzの場合）
ウエイト	ベクタ割り込み処理を行う場合	11～12クロック（1.1 μs～1.2 μs）	-
	ベクタ割り込み処理を行わない場合	4～5クロック（0.4 μs～0.5 μs）	-
発振安定時間（OSTSの設定時間）		-	OSTSレジスタに設定したウエイト時間 $2^{11}/f_X$（204.8 μs） $2^{13}/f_X$（819.2 μs） $2^{14}/f_X$（1.64 ms） $2^{15}/f_X$（3.27 ms） $2^{16}/f_X$（6.55 ms）

(3)　CPUクロック（f_{CPU}）が高速システム・クロックのX2外部クロック（f_{EXCLK}）から供給されている場合

図2－13　割り込み要求信号による復帰タイミング図（f_{CPU} ← f_{XH} ← f_{EXCLK}）

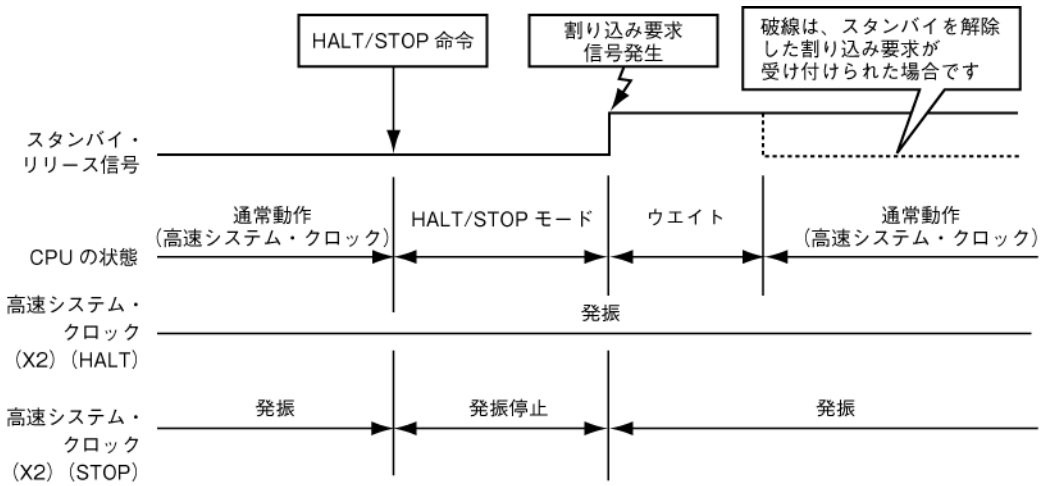


表2－8　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{XH} ← f_{EXCLK}）

要因		HALTモード（f_{CPU} = 8 MHzの場合）	STOPモード（f_{CPU} = 8 MHzの場合）
ウエイト	ベクタ割り込み処理を行う場合	11～12クロック（1.375～1.5 μs）	17～18クロック（2.125～2.25 μs）
	ベクタ割り込み処理を行わない場合	4～5クロック（1.375～1.5 μs）	11～12クロック（1.375～1.5 μs）

(4)　CPUクロック（f_{CPU}）がサブシステム・クロック（f_{SUB}/2 ← f_{XT}, f_{EXCLKS}）から供給されている場合

XT1入力（f_{XT}：水晶発振子／セラミック発振子），XT2入力（f_{EXCLKS}：外部クロック）とも，HALT命令のみ有効となります（STOP命令の実行は禁止です）。復帰タイミング図，復帰時間は，(1)～(3)のHALT命令からの復帰の場合と同じになります。

2. 5　スタンバイ機能のリセットによる復帰

リセット信号発生による復帰の場合は，すべてのクロック発生回路はリセット期間中停止しますので，STOP, HALT状態とも復帰時間は同じです。

(1)　リセット信号による復帰

リセット信号による復帰の場合，リセット期間中はすべてのクロック発生回路が停止しますので，HALT, STOPモードとも復帰時間は同じになります。

① CPUクロック（f_{CPU}）が高速内蔵発振クロック（f_{IH}）から供給されている場合

図2－14　リセット信号による復帰タイミング図（f_{CPU} ← f_{IH}）

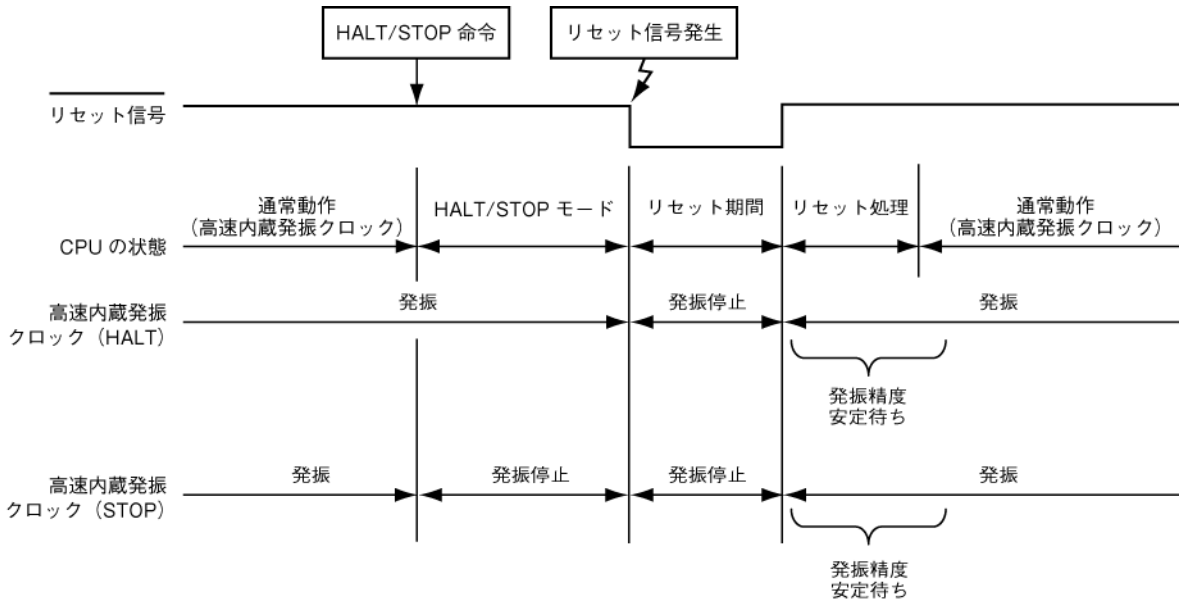


表2－9　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{IH}）

要因	HALTモード/STOPモード
リセット処理	12～51 μs
発振精度安定待ち	102～407 μs

② CPUクロック（f_{CPU}）が高速システム・クロックのX1発振子（f_X）から供給されている場合

図2－15　リセット信号による復帰（f_{CPU} ← f_{XH} ← f_X）

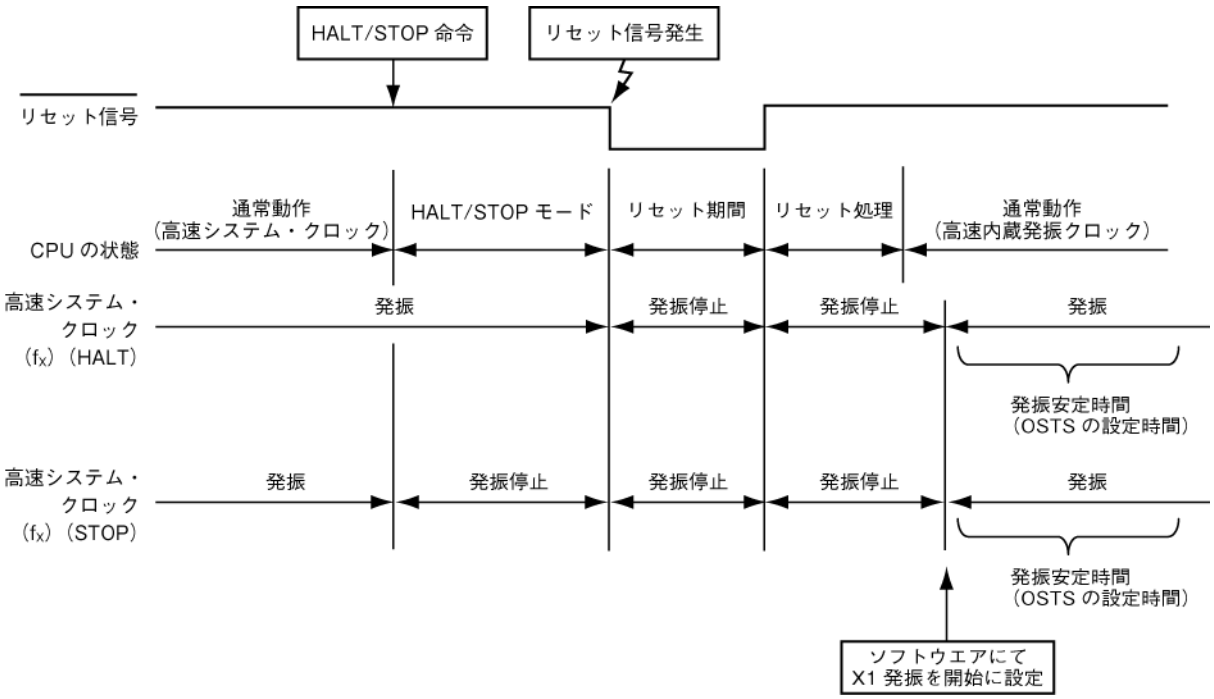


表2－10　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{XH} ← f_X）

要因	HALTモード/STOPモード
リセット処理	12～51 μs
発振精度安定時間（OSTSの設定時間）	OSTSレジスタに設定したウエイト時間 （f_{CPU} = 10 MHzの場合） $2^{11}/f_X$（204.8 μs） $2^{13}/f_X$（819.2 μs） $2^{14}/f_X$（1.64 ms） $2^{15}/f_X$（3.27 ms） $2^{16}/f_X$（6.55 ms）

③ CPUクロック（f_{CPU}）が高速システム・クロックのX2外部クロック（f_{EXCLK}）から供給されている場合

図2－16　割り込み要求信号による復帰タイミング図（f_{CPU} ← f_{XH} ← f_{EXCLK}）

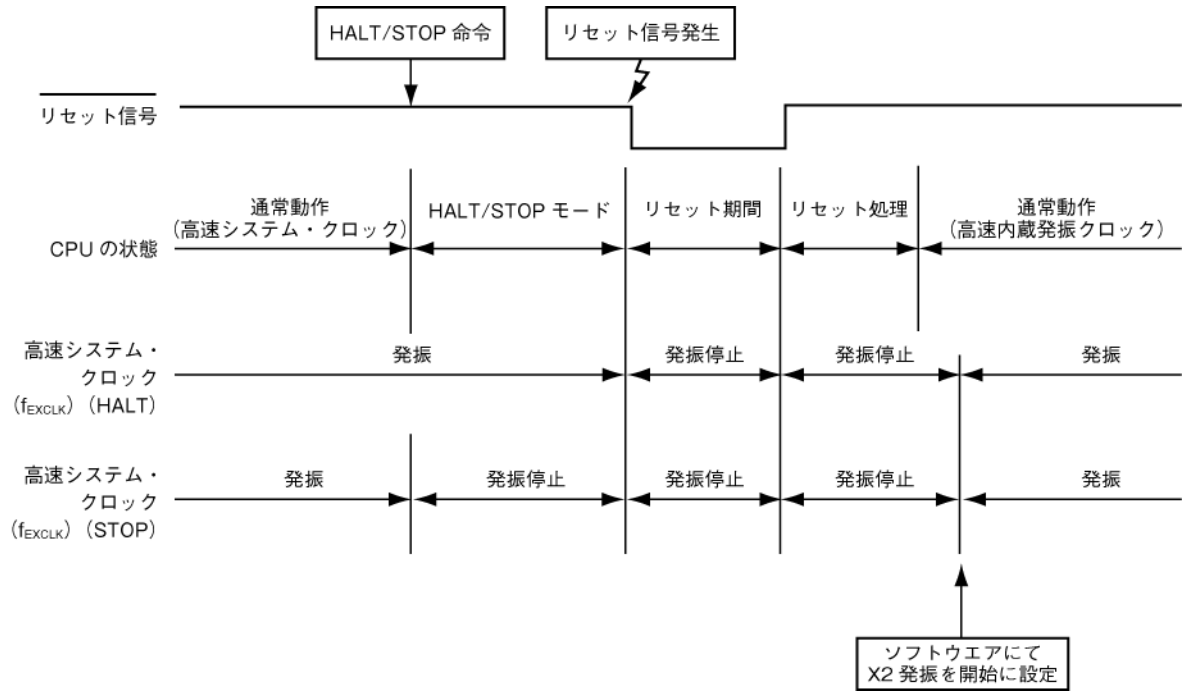


表2－11　HALT, STOPモードからの復帰時間（f_{CPU} ← f_{XH} ← f_{EXCLK}）

要因	HALTモード/STOPモード
リセット処理	12～51 μs

④ CPUクロック（f_{CPU}）がサブシステム・クロックの2分周（$f_{SUB}/2$）から供給されている場合

XT1入力（f_{XT}：水晶発振子／セラミック発振子），XT2入力（f_{EXCLKS}：外部クロック）とも，HALT命令のみ有効となります（STOP命令の実行は禁止です）。

XT1入力の発振安定時間は，使用する発振子にあわせてユーザがタイマなどで計測するウエイト時間です。

図2－17　割り込み要求信号による復帰タイミング図（f_{CPU} ← $f_{SUB}/2$）

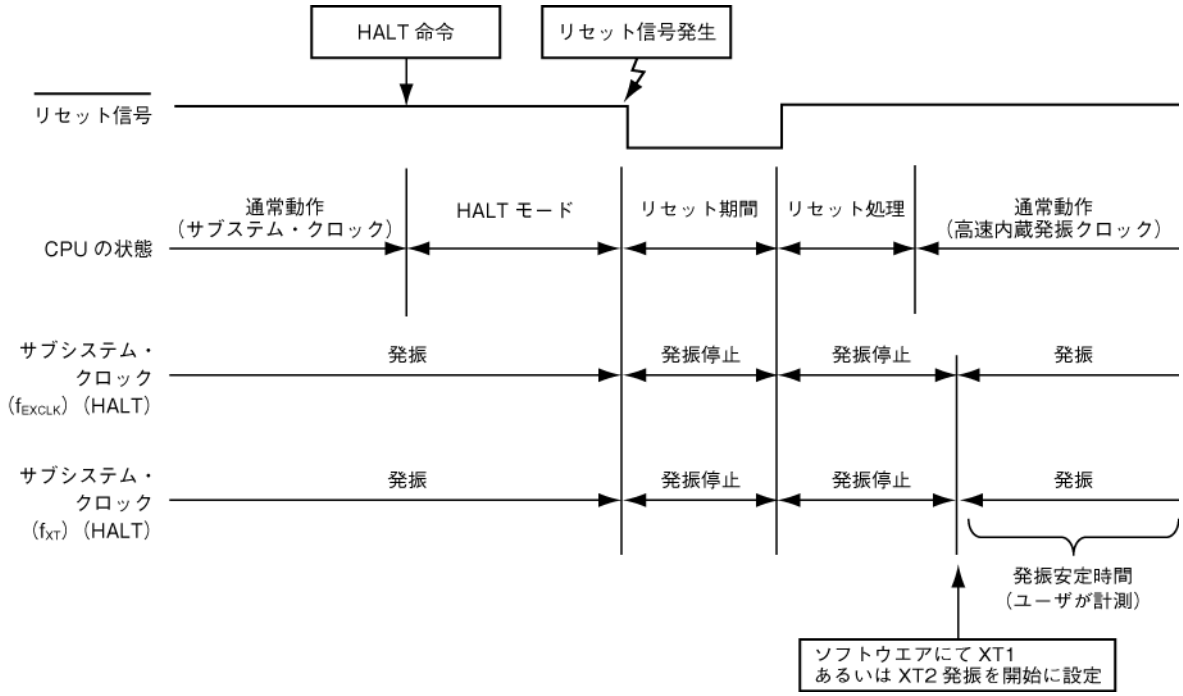


表2－12　HALTモードからの復帰時間（f_{CPU} ← $f_{SUB}/2$ ← f_{XT}, f_{EXCLKS}）

要因	HALTモード/STOPモード
リセット処理	12～51 μs
発振安定時間（ユーザが計測） （f_{CPU} ← $f_{SUB}/2$ ← f_{XT}のみ）	使用する発振子に適応したウエイト時間を，タイマなどで計測してウエイトします

2. 6　クロック発生回路，スタンバイ機能の注意事項

- サブシステム・クロックの2分周（$f_{SUB}/2$）がCPUクロック（f_{CPU}）に供給されている場合は，フラッシュ・メモリのセルフ・プログラミング機能は使用できません。セルフ・プログラミング機能は，高速内蔵発振クロックか高速システム・クロックがCPUクロック（f_{CPU}）に供給されている場合に使用できます。

- STOP命令の実行か，プログラムで任意に周辺ハードウエア・クロック（f_{PRS}）を停止させる場合は，必ず周辺ハードウエア・クロックをクロック・ソースとする周辺ハードウエアを停止させてください。

【コラム】 セルフ・プログラミング機能について

78K0/Kx2-Lシリーズのマイクロコントローラには，ユーザ・プログラムでフラッシュ・メモリを書き換えることのできる，セルフ・プログラミング機能があります。セルフ・プログラミング機能を使う場合，当社が提供するセルフ・プログラミング・ライブラリを使用します。セルフ・プログラミング・ライブラリについては，「第4章 4.2 初期設定とワークエリア」のコラム「EEPROM®エミュレーション・ライブラリ，セルフ・プログラミング・ライブラリについて 1」を参照してください。

第3章　レギュレータ

3. 1　レギュレータの概要

78K0/Kx2-Lマイクロコントローラには，デバイス内部を定電圧動作させるための回路を内蔵しています。このときレギュレータ出力電圧を安定させるために，REGC端子にコンデンサ（0.47～1 μF）を介し，Vssに接続してください。ただし，高速内蔵発振クロック，外部メイン・システム・クロック動作時からのSTOPモードを使用する場合は，0.47 μFを推奨します。また，内部電圧の安定のために使用するため，特性のよいコンデンサを使用してください。

レギュレータ出力電圧は，通常は2.4 V（TYP.），低消費電力モードでは2.0 V（TYP.）です。

3. 2　レギュレータを制御するレジスタ

（1）レギュレータ・モード制御レジスタ（RMC）

レギュレータの出力電圧を設定するレジスタです。

RMCは，8ビット・メモリ操作命令で設定します。

リセット信号の発生により，00Hになります。

図3－1　レギュレータ・モード制御レジスタ（RMC）のフォーマット

アドレス：FF3DH　リセット時：00H　R/W

略号	7	6	5	4	3	2	1	0
RMC								

RMC[7:0]	レギュレータの出力電圧の制御
56H	低消費電力モード（2.0 V）固定
00H	条件によって通常電力モード（2.4 V）と低消費電力モード（2.0 V）を切り替える（表3－1参照）
上記以外	設定禁止

注意1.　RMCレジスタの設定値を56Hから00Hに変更して，5 MHz以上のCPU動作周波数を使用する場合，RMCレジスタ設定後10 μs以上経過してから，PCCレジスタとRCMレジスタを変更してください。

注意2.　**低消費電力モード固定の設定で使用する場合は，以下の場合にかぎり使用可能です。**

＜CPU**クロックに**X1**クロック選択時＞**

f_X**≦**5 MHz**かつ**f_{CPU}**≦**5 MHz

＜CPU**クロックに高速内蔵発振クロック，外部入力クロック，サブシステム・クロック選択時＞**

f_{CPU}**≦**5 MHz

表3－1　**レギュレータ出力電圧条件**

モード	出力電圧	条　件
低消費電力モード	2.0 V	STOPモード時
		サブシステム・クロック（f_{XT}）でCPU動作中で，高速システム・クロック（f_{XH}）と高速内蔵発振クロック（f_{IH}）が共に停止
		サブシステム・クロック（f_{XT}）でCPU動作設定時のHALTモード中で，高速システム・クロック（f_{XH}）と高速内蔵発振クロック（f_{IH}）が共に停止
通常電力モード	2.4 V	上記以外

3. 3　セルフ・プログラミングに関する注意事項

1. セルフ・プログラミングまたはEEPROMエミュレーション実行時は，レギュレータ出力電圧のモードを固定にしてください。
2. 通常電力モードでフラッシュ・メモリを書き換える場合，書き換え可能な電源電圧範囲は，V_{DD}≧2.5Vです。また，通常電力モードの場合，セルフ・プログラミング・ライブラリによるプログラム領域の書き換えは可能です。
3. 低消費電力モードでのフラッシュ・メモリの書き換えは，次の点に注意してください。
   - データ領域の書き換えのみ可能です。プログラム領域の書き換えはできません。
   - 電源電圧がレギュレータの出力電圧（2.0 V）以下の場合，書き換えできません。
   - 低消費電力モードにて書き込み，消去を行ったフラッシュ・メモリは，通常電力モードでは読み出せません。使用する場合は，低消費電力モードに切り替えて，フラッシュ・メモリの内容をRAMに移してください。
   - EEPROMエミュレーションで書き換えを行う場合，同一ブロックへの上書きは可能です。しかし，セルフ・プログラミング・ライブラリで書き換えを行う場合，同一ブロックへの上書きはできませんので，必ず書き換えの前にブロック消去を実行してください。
   - 通常電力モードから低消費電力モードに切り替えたあとにセルフ・プログラミングを実行する場合，モードの切り替え後，2 msのウエイト時間が必要です。

備考　セルフ・プログラミング機能の詳細およびセルフ・プログラミング・ライブラリの詳細については，「78K0**マイクロコントローラ　ユーザーズ・マニュアル　セルフ・プログラミング・ライブラリ** Type01（U18274J）」と「78K0 **セルフ・プログラミング・ライブラリ** Type01　**ユーザーズ・マニュアル** 78K0/Kx2-L, 78K0/Ix2**仕様差分の件**（ZBB-CB-09-0019）」を参照してください。

EEPROMエミュレーションの詳細については，「78K0**マイクロコントローラ　ユーザーズ・マニュアル**　EEPROM**エミュレーション・ライブラリ**　Type01（U18275J）」と「78K0 EEPROM**エミュレーション・ライブラリ**　Type01　**ユーザーズ・マニュアル**　78K0/Kx2-L，78K0/Ix2**仕様差分の件**（ZBB-CB-09-0020）」を参照してください。

第4章　低消費電力プログラム例

この章では，78K0/KC2-Lマイクロコントローラの低消費電力の各機能を使ったサンプル・プログラムについて紹介します。

掲載している各プログラム・リストは，主にスタンバイ機能，レギュレータに関連するプログラムです。すべてのプログラムについては，Cソース・コードを参照してください。

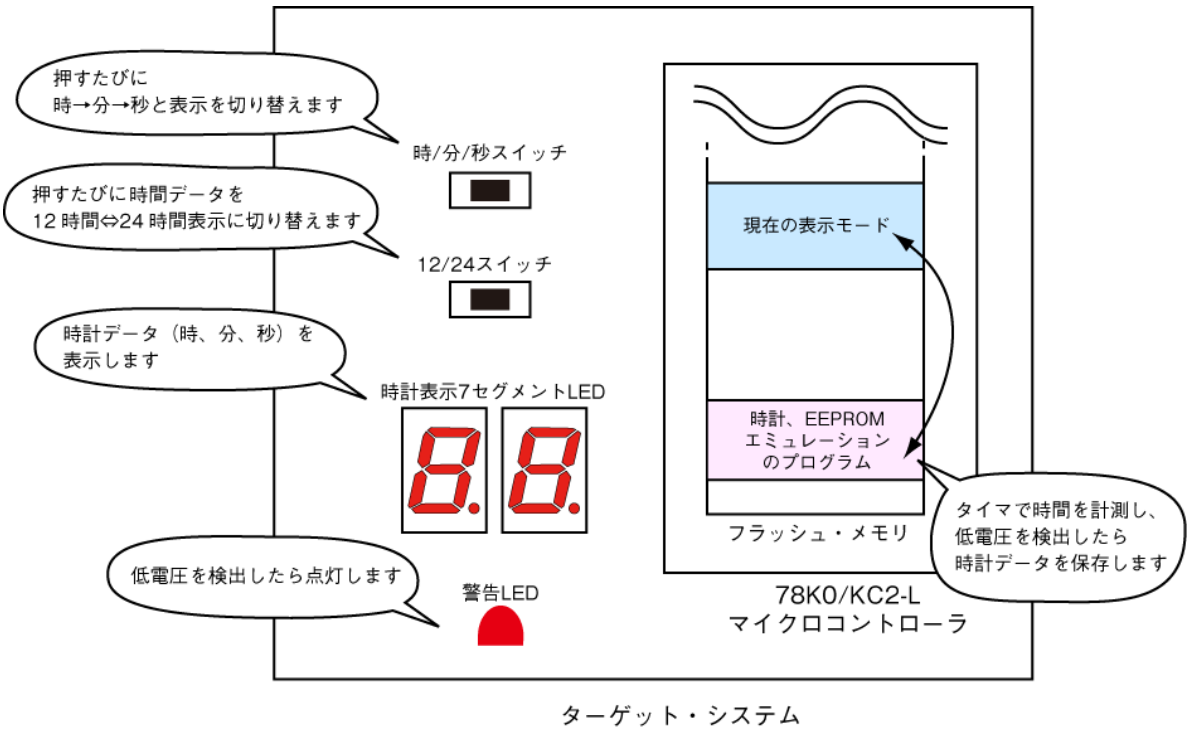
4.1　仕様と全体の流れ

リアルタイム・カウンタを使って時計機能を実現します。スイッチで表示モード（時，分，秒と12/24時間表示）を切り替えられます。

低電圧検出回路によって，低電圧（2.53 V±0.1 V）を検知すると，警告LEDを点灯し，現在の表示モード（時，分，秒と12/24時間表示）をEEPROMエミュレーション・ライブラリでフラッシュ・メモリにセルフ・プログラミングします。

低電圧を検知したあとにいったん電源を遮断して再び電源投入すると，表示モードが再現され，フラッシュ・メモリに書き込まれていることを確認することができます。

図4－1　プログラムの動作（イメージ図）



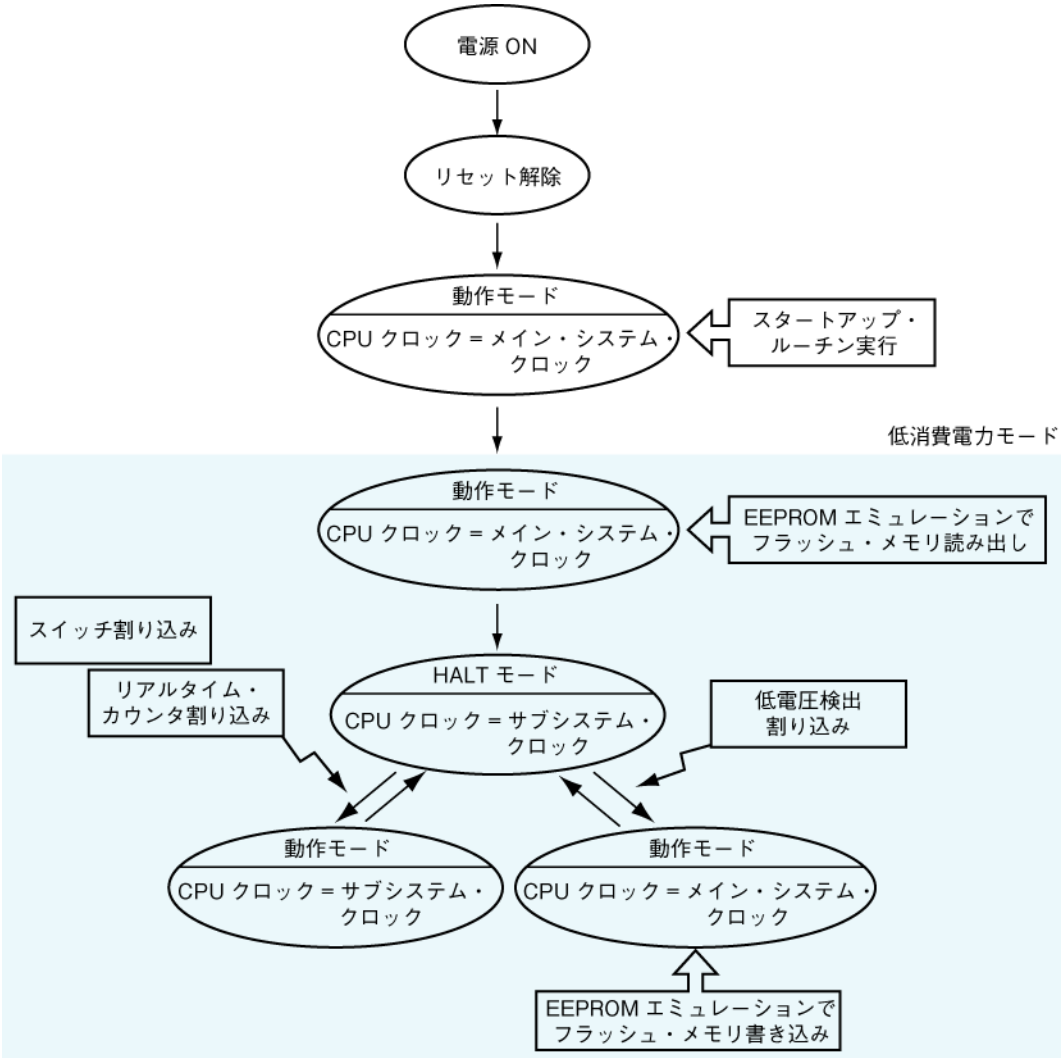
以下にサンプル・プログラムの状態遷移図を示します。

スタートアップ・ルーチン実行後，低消費電力モードに設定してEEPROMエミュレーション・ライブラリで表示モードを読み出します。その後CPUクロックをサブシステム・クロックからの供給に切り替え，HALTモードになります。

リアルタイム・カウンタによる割り込み，スイッチ入力による割り込み，低電圧検出割り込み発生時はHALTモードから復帰して動作モード（通常の動作している状態）になります。

低電圧検出割り込み処理では，CPUクロックをメイン・システム・クロックからの供給に切り替え，EEPROMエミュレーション・ライブラリで表示モードをフラッシュ・メモリに書き込みます。

図4－2　プログラムの状態遷移

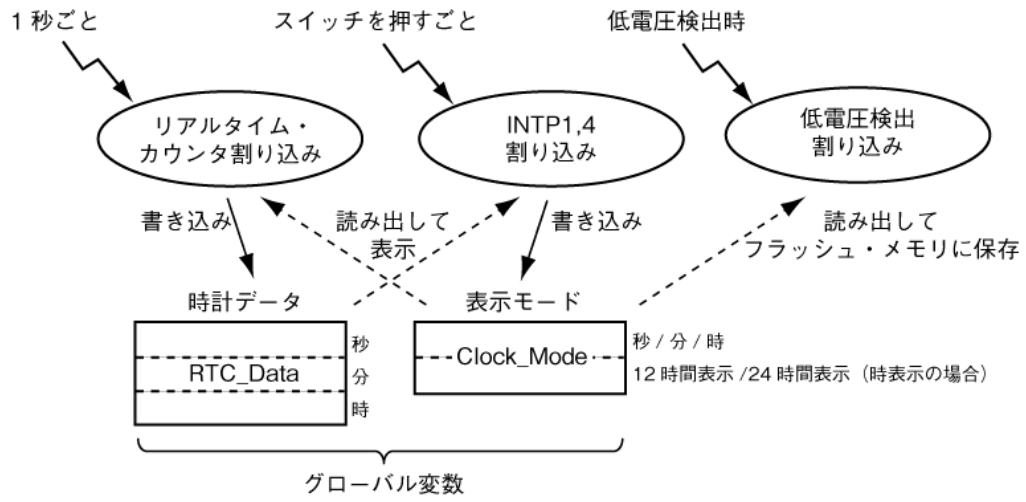


以下にサンプル・プログラムの各割り込み処理とデータのフローを示します。

リアルタイム・カウンタ割り込み処理では，リアルタイム・カウンタの時，分，秒のデータを変数「RTC_Data」に保存します。INTP1, INTP4割り込み処理では，表示モードの変数「Clock_Mode」を書き換えます。

低電圧検出割り込み処理では，表示モードを読み出して，フラッシュ・メモリに保存します。

図4－3　各割り込み処理とデータ



サンプル・プログラムでは，処理に応じてCPUクロック（f_{CPU}）の供給元を切り替えています。

以下にプログラムの流れとクロックの状態について示します。図中の赤字はプログラムの設定による変更を示します。

図4－4　プログラムの処理とクロックの状態

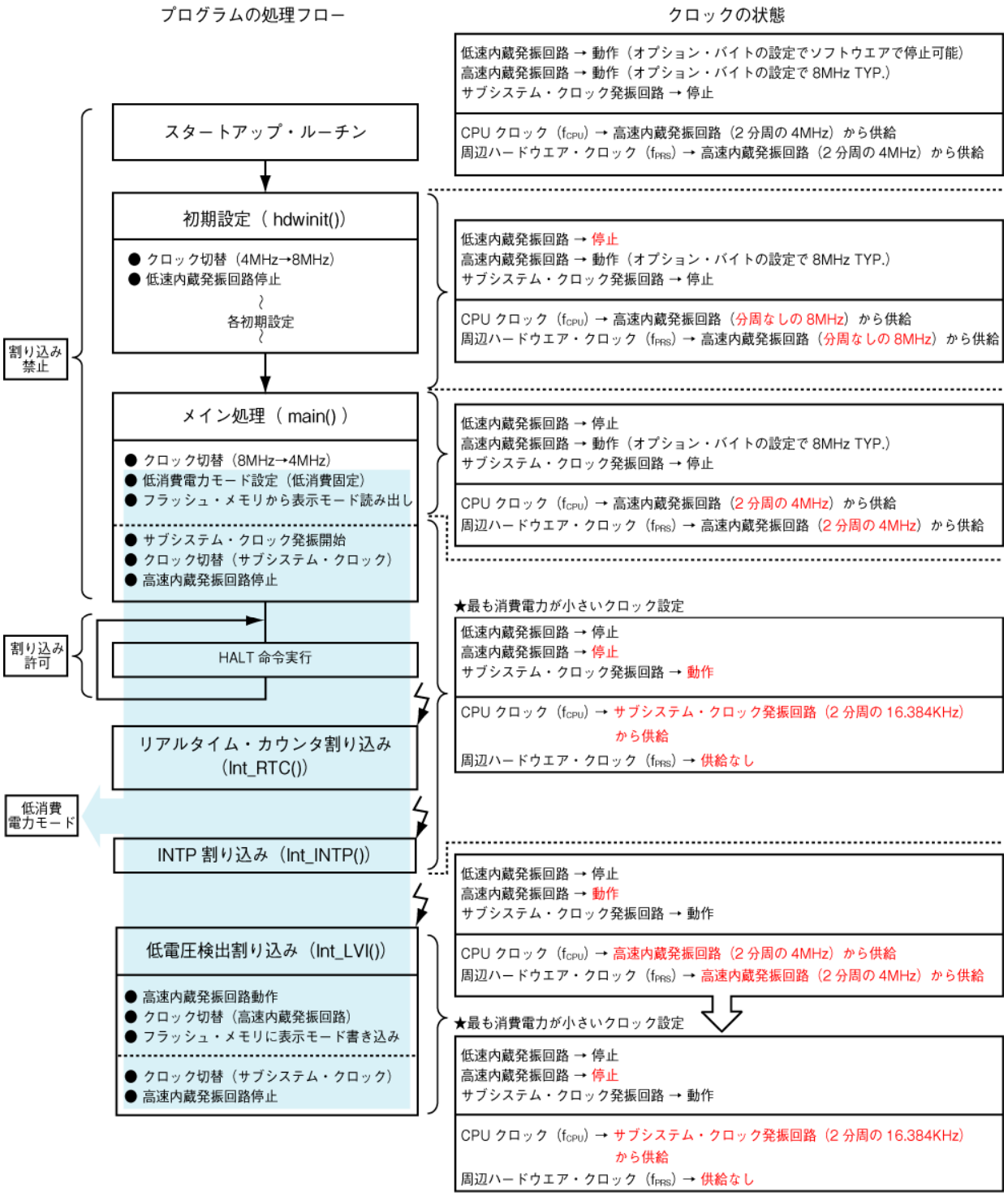


表4－1　プログラムのファイル構成（フォルダ名：U19612_Kx2L_LowPower_01_AN）

フォルダ名		ファイル名	内容
U19612_Kx2L_LowPower_01_AN		Kx2L_LowPower.lmf	ロード・モジュール・ファイル
		Kx2L_LowPower.hex	HEX形式オブジェクト・モジュール・ファイル
		Kx2L_LowPower.prw	プロジェクト・ファイル
		-	その他生成されたファイル，プロジェクト関連ファイル等
	¥src	initial.c	初期設定等
		main.c	メイン・プログラム
		Int_RTC.c	リアルタイム・カウンタ割り込み関数
		Int_Intp.c	INTPエッジ検出割り込み関数
		Int_LVI.c	低電圧検出割り込み関数
		option.asm	オプション・バイト設定ファイル
		LowPower.h	ヘッダ・ファイル
		Kx2_eee.h	ヘッダ・ファイル（EEPROMエミュレーション用）

このサンプル・プログラムは，テセラ・テクノロジー社製の評価ボード「TK-78K0/KC2L」（μPD78F0588GA-GAM-AX搭載）をターゲットとしています。TK-78K0/KC2Lの操作方法はTK-78K0/KC2Lのドキュメントを参照してください。

以下にプログラムで使用しているI/Oとマイコンの SFRを示します。

表4－2　プログラムで使用しているTK-78K0/KC2LのI/O

I/O	用途	接続端子
7セグメントLED	・ 電源投入後は"-"を表示する ・ 時計データの時，分，秒を表示する ・ 時／分／秒スイッチを押すごとに表示を時→分→秒と切り替える ・ 12/24スイッチを押すごとに時間の表記を12時間⇔24時間と切り替える ・ デフォルトの表示は秒，12時間表記だが，低電圧割り込みが入ってからの起動の場合はそれまでの表示を再現する	＜セグメント＞ ポート2 P20～P27端子 ＜桁切り替え＞ ポート0 P00, P01端子
時／分／秒スイッチ（SW5）	押すごとに時計表示を時→分→秒と切り替える	P30/INTP1端子
AM/PMスイッチ（SW6）	押すごとに時間の表示を12時間⇔24時間と切り替える	P33/TI51/TO51/INTP4端子
警告LED	低電圧検出割り込みが入ると点灯する （このLEDは，TK-78K0/KC2Lに拡張したものです）	ポート0 P02端子

表4－3　プログラムで使用しているマイコンのSFR

SFR	用途
リアルタイム・カウンタ	時計機能を実現する（割り込み）
低電圧検出回路	電源電圧の低電圧を検出する（割り込み）
割り込み端子	INTP1：時／分／秒スイッチの切り替えを検知する INTP4：12/24スイッチの切り替えを検知する
ポート	ポート2：7セグメントLEDに出力する ポート0_0，0_1：7セグメントLEDの表示桁を切り替える （各ラッチにクロック信号を出力する）

4. 2　初期設定と各ワークエリア

サンプル・プログラムではEEPROMエミュレーション・ライブラリを使用してフラッシュ・メモリに書き込んでいるため，EEPROMエミュレーションのヘッダのインクルード，各ライブラリ用のワークエリアを設定する必要があります。

また，コンパイラでヘッダ・ファイルのパスの設定，リンカでEEPROMエミュレーション・ライブラリとセルフ・プログラミング・ライブラリのライブラリ・ファイルを設定する必要があります。

リスト4－1　スタートアップ・ルーチン初期設定関数（main.c）

```
/*-----------------------------------------------------------------------
*       前処理指令（#pragma 指令）
*-----------------------------------------------------------------------*/

#pragma  sfr                /* 特殊機能レジスタ（SFR）名を記述可能にする */
#pragma  di                 /* DI 命令を記述可能にする */
#pragma  ei                 /* EI 命令を記述可能にする */
#pragma  nop                /* NOP 命令を記述可能にする */
#pragma  halt               /* HALT 命令を記述可能にする */
#pragma  stop               /* STOP 命令を記述可能にする */

/*-----------------------------------------------------------------------
*       ヘッダ・ファイル
*-----------------------------------------------------------------------*/
#include "eeelib.h"         /* EEPROM エミュレーション・ライブラリヘッダ */
#include "Kx2_eee.h"        /* サンプル・プログラム・ヘッダ */
#include "LowPower.h"       /* サンプル・プログラム・ヘッダ */

/*-----------------------------------------------------------------------
*       マクロ定義
*-----------------------------------------------------------------------*/
#define NOP_4  NOP();NOP();NOP();NOP();         /* NOP を 4 回実行する  */
#define NOP_8  NOP_4 NOP_4                      /* NOP を 4 回実行するマクロを 2 回実行する（4×2=8）*/

/*-----------------------------------------------------------------------
*       プロトタイプ宣言
*-----------------------------------------------------------------------*/

        省　略

/*-----------------------------------------------------------------------
*       グローバル変数
*-----------------------------------------------------------------------*/

sreg    UCHAR EntryRAM[100];                    /* セルフ・プログラミング・ライブラリ用エントリ RAM */
UCHAR   EEPROM_DataBuf[EEPROM_BUFFLEN];         /* EEPROM エミュレーション・ライブラリ用データバッファ */
UCHAR   Clock_Mode[EEPROM_DATALEN];
UCHAR   RTC_Data[3];
UCHAR   const LED_Pattern[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x83, 0xf8, 0x80,
                               0x98, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e, 0xbf};
/***********************************************************************
        リセット解除後の初期化処理

***********************************************************************/
void hdwinit(void)
{
        /* 割り込み禁止 */
        DI();

        /* 周辺ハードウエア初期設定 */
        Init_sfr();
}
```


【コラム】　EEPROMエミュレーション・ライブラリ，セルフ・プログラミング・ライブラリについて 1

当社が提供するEEPROMエミュレーション・ライブラリとセルフ・プログラミング・ライブラリは，ユーザ・プログラムからフラッシュ・メモリに書き込むためのライブラリです。

セルフ・プログラミング・ライブラリは，フラッシュ・メモリへの書き込み，消去を行うライブラリです。

EEPROMエミュレーション・ライブラリは，書き込みの最小単位が4バイト，消去が1ブロック（= 1024バイト）となるフラッシュ・メモリに，あたかも任意のサイズのデータを書き換えているような操作を実現するためのライブラリです。実際のフラッシュ・メモリへの書き込み，消去などはEEPROMエミュレーション・ライブラリからセルフ・プログラミング・ライブラリを呼び出しています。

未接続ポートはすべて出力ポートに設定しています。サブシステム・クロックを開始したあと，発振安定時間計測のためにタイマTM00を開始します。

リスト4－2　初期設定関数（main.c） 1

```
/************************************************************************************************
;       Init_sfr()
;------------------------------------------------------------------------------------------------
;       周辺ハードウエアの初期設定
;------------------------------------------------------------------------------------------------
;       [I N]     -
;       [OUT]     -
;************************************************************************************************/
void Init_sfr(void)
{
        /* ループカウンタ用 */
        UCHAR i;

        /* uPD78F0583,uPD78F0588 使用時の ROM/RAM サイズの設定 */
        IMS =     0xC8;

        /* リセット解除後は、オプション・バイトで高速内蔵発振回路は 8MHz（TYP.）*/
        /* リセット解除後は、高速内蔵発振回路がメイン・クロックで、メイン・クロックの 2 分周が CPU クロック */
        /* CPU クロックをメイン・クロックの 2 分周から分周なしに切り替える */
        Change_CPUClk(MAINSYSCLK2_1);

        /* 16 ビット・タイマ TM00 設定（サブシステム・クロックの発振安定待ち（約 2 秒）割り込みは使用しない ）*/
        TMC00 = 0b00000000;             /* 16 ビット・タイマ 00 動作禁止。動作クロック供給禁止。カウンタ 00 をクリア */
        TMMK000 = 1;                    /* タイマ 00 割り込みマスクセット（割り込み禁止）*/
        TMIF000 = 0;                    /* タイマ 00 割り込み要求フラグクリア */
        CRC00 = 0b00000000;             /* CR000 をコンペア・レジスタとして動作に設定 */
        TOC00 = 0b00000000;             /* 出力コントロール・レジスタ 00 に TO00 端子の出力禁止を設定 */
        PRM00 = 0b00000010;             /* プリスケーラにカウント・クロック=256 分周（31.25KHz（ソース=8MHz 時））*/
        CR000 = 62500-1;                /* 31.25KHz（1/212=50）* 62500= 約 2 秒 */
        CR010 = 0xFF;

        /* サブシステム・クロックを動作させる */
        Start_SubSystemClk();

        /* タイマ TM00 開始 ↓サブシステム・クロック発振安定のためここから 2 秒を計測 */
        Start_TM00();

        /* 低速内蔵発振回路を停止 */
        LSRSTOP = 1;

        /* ポート P00 ～ P02 */
        P0  = 0b00000100;               /* P00 は 7 セグ LED 左桁、P01 は右桁、P02 は低電圧警告 LED←消灯（拡張 )*/
        PM0 = 0b00000000;               /* 出力ポートに設定 */

        /* ポート P10 ～ P17 */
        ADS = 0b01000000;               /* アナログ入力チャネル指定レジスタを OP アンプに設定 */
        ADPC0 = 0b11111111;             /* 全てデジタル入出力 */
        PU1 = 0b00011000;               /* P13、P14 は内部プルアップを抵抗を接続する */
        P1 = 0b10111111;                /* P13、P14 は TxD、RxD（マイコンと接続）、それ以外は未接続 */
        PM1 = 0b00011000;               /* P13、P14 は入力、他は出力ポートに設定 */

        /* ポート P20 ～ P27 */
        P2 = 0b10111111;                /* P20 ～ P27 は 7 セグメント LED のセグメント出力 "-" を表示 */
        PM2 = 0b00000000;               /* 全て出力ポートに設定 */

        /* ポート P30 ～ P33 */
        PU3 = 0b00001111;               /* P30/INTP1 は SW5、P33/INTP4 は SW6  P31 は SW3-2、P32 は SW3-3  */
        PM3 = 0b11111111;               /* 入力ポートに設定 ↑内部プルアップ抵抗を接続する */
```

サブシステム・クロックの動作を開始

発振安定時間（2 秒）の計測を開始

【コラム】 EEPROM**エミュレーション・ライブラリ，セルフ・プログラミング・ライブラリについて** 2

ユーザ・プログラムでEEPROMエミュレーション・ライブラリを使用する場合は，セルフ・プログラミング・ライブラリも使用することになりますので，リンカでEEPROMエミュレーション・ライブラリ，セルフ・プログラミング・ライブラリの各ライブラリ・ファイルをリンクするように設定してください。

リスト4－2　初期設定関数（main.c）　2

```
        /* ポート P40～P42 */
        P4  = 0b00000000;           /* P40～P42   全て未接続 */
        PM4 = 0b00000000;           /* 全て出力ポートに設定 */

        /* ポート P60～P63 */
        P6  = 0b00000000;           /* P60～P63   全て未接続 */
        PM6 = 0b00000000;           /* 全て出力ポートに設定 */

        /* ポート P70～P75 */
        PU7 = 0b00111111;           /* P70～P75 は SW3-4～SW3-8 内部プルアップ抵抗を接続する */
        PM7 = 0b11111111;           /* 入力ポートに設定 */

        /* ポート P120～P125 */
        PU12 = 0b00000000;          /* P120 は未接続 */
        PM12 = 0b11111110;          /* P120 は出力ポートに設定 P121～P125 は入力端子（X1,X2,XT1,XT2,RESET）*/

        /* 7セグメント LED の D-FF にデータを出力 */
        LED_LEFT = 1;               /* 左桁に表示（D-FF をラッチ）*/
        LED_LEFT = 0;

        LED_RIGHT = 1;              /* 右桁に表示（D-FF をラッチ）*/
        LED_RIGHT = 0;

        /* 低電圧検出回路（2.53V ± 0.1V で割り込み発生）オプション・バイトで LVIM は ALL 0（動作禁止）*/
        LVIS = 0b00001011;          /* 検出電圧を 2.53V±1 に設定 */
        LVION = 1;                  /* 低電圧検出回路の動作を許可に設定 */

        for(i=0; i<2; i++)          /* 約10μs ウエイト 8MHz 動作時 1クロック=125ns */
        {                           /* このループで 90 クロックかかるので、実際は 90×125ns=11.25μs */
            NOP_4                   ← マクロ定義により NOP を 4 回展開する
        }

                                    /* ↑8MHz 動作時 1クロック=125ns NOP は 2 クロック 10μs/(125*2ns)=40 */
        LVIMK = 1;                  /* LVI 割り込みのマスク設定（割り込み禁止）*/
        LVIIF = 0;                  /* LVI 割り込み要求フラグクリア */

        /* INTP0,4 割り込み設定 */
        EGNCTL0 = 0b00010010;       /* INTP1、INTP4 を立下りエッジ検出割り込みに設定 */
        EGPCTL0 = 0b00000000;       /* INTP1、INTP4 を立下りエッジ検出割り込みに設定 */
        PMK1 = 1;                   /* INTP1 割り込みのマスク設定（割り込み禁止）*/
        PIF1 = 0;                   /* INTP1 割り込みフラグクリア */
        PMK4 = 1;                   /* INTP4 割り込みのマスク設定（割り込み禁止）*/
        PIF4 = 0;                   /* INTP4 割り込みフラグクリア */

        /* リアルタイム・カウンタ設定（割り込み使用）*/
        RTCEN    = 1;               /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給 */  ← 制御クロック供給
        RTCE     = 0;               /* リアルタイム・カウンタ動作停止 */
        RTCC0 = RTCC0|0b00100010;   /* カウンタ動作開始 RTCCL 端子 32.768kHz 出力 1 秒定周期割り込み ANPM12 時制 */
                                    /* RTCCL 端子の出力（32.768kHz）許可 */

        /* リアルタイム・カウンタ カウント開始時刻の設定（コメントの時間 / 年月日はヘッダ・ファイルのデフォルト値）*/
        SEC      = SEC_INI;         /* 秒 :50 */
        MIN      = MIN_INI;         /* 分 :59 */
        HOUR     = HOUR_INI;        /* 時 :11 */
        WEEK     = WEEK_INI;        /* 曜日 : 土曜日 */
        DAY      = DAY_INI;         /* 日 :01 */
        MONTH    = MONTH_INI;       /* 月 :01 */
        YEAR     = YEAR_INI;        /* 年 :00 */

        RTCMK    = 1;               /* リアルタイム・カウンタ 定周期 / アラーム割り込みマスク設定（割り込み禁止）*/
        RTCIF    = 0;               /* CT0～CT2 を変更したら割り込み要求フラグクリアする */
        RTCEN    = 0;               /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給停止 */  ← 制御クロック供給停止
    }
```

【コラム】　リアルタイム・カウンタの関連レジスタの設定について

リアルタイム・カウンタは，サブシステム・クロック（f_{SUB}）によって動作（カウント）しますが，関連レジスタの設定はサブシステム・クロック（f_{SUB}）が停止していてもCPUクロック（f_{CPU}）が供給されていれば可能です。関連レジスタへの書き込みには制御クロックを供給する必要があります（レジスタの読み出しには必要ありません）。

周辺イネーブル・レジスタ0（PER0）のRTCENを1に設定することにより，制御クロックが供給されます。関連レジスタの書き込みを行わないときは，停止（RTCEN = 0）しておくと低消費電力化を実現できます。

4.3　メイン処理

main()関数は，スタートアップ・ルーチンでhdwinit()関数の実行，メモリの初期化設定などが行われたあとに呼び出されます。

リアルタイム・カウンタを開始させる前に，フラッシュ・メモリから表示モードを読み出します。低電圧検出割り込みによって表示モードが書き込まれている場合は以前の表示を再現します。

hdwinit()関数内でサブシステム・クロックの動作を開始し，2秒計測のためにタイマTM00を開始しているので，サブシステム・クロックをCPUクロックに供給する前に2秒計測かどうかを検知します。2秒経過していない場合はタイマTM00の割り込みを許可し，HALTモードで待機します。割り込み許可フラグは禁止としているので（IE = 0），割り込みベクタには分岐しません。したがって，割り込み要求が発生した場合は，割り込み要求が発生したあとに手動で割り込み要求フラグをクリアする必要があります。

リスト4－3　メイン関数（main.c）

```
/************************************************************************************
スタートアップ・ルーチンから実行される
************************************************
void main(void)
{
        /* CPU クロックをメイン・クロックの分周なしから 2 分周に
        Change_CPUClk(MAINSYSCLK1_2);

        /* 低消費電力モードに設定 */
        Set_LowPowerMode();

        /* フラッシュ・メモリから表示モードを読み出す */
        Read_ClockMode();

        /* EEPROM エミュレーション・ライブラリ実行後は DI 実行 */
        DI();

        /* 表示モードより 12/24 時間表示をリアルタイム・カウンタに設定する */
        Set_ampm();

        /* タイマ TM00 要求フラグセット（2 秒経過） ? */
        if(Wait_TM00() == FALSE)
        {

                /* タイマ 00 割り込みマスク・フラグクリア（割り込み許可）*/
                Clear_Tm00IntMsk();

                /* タイマ TM00 割り込みが発生するまで HALT モード */
                 HALT();
        }

        /* タイマ 00 割り込み要求フラグクリア */
        Clear_Tm00IntReq();

        /* タイマ TM00 停止 */
        Stop_TM00();

        /* サブシステム・クロックを CPU クロックに設定する */
        Change_CPUClk(MAINSYSCLK_SUBCLK);

        /* 高速内蔵発振回路を止める */
        Stop_HispeedIntClk();

        /* リアルタイム・カウンタ開始 */
        Start_RTC();

        /* 各割り込みマスク・フラグを解除する        */
        Clear_IntMask();

        /* 割り込みを許可する */
        EI();

        /* 無限ループ */
        while(1)
        {
                /* HALT モード */
                HALT();

        }
}
```

低消費モードに設定するため CPU クロックをメイン・クロックの 2 分周に切り替える（メイン・クロックは内蔵高速発振回路 8MHz（TYP.）なので 4MHz（TYP.）となる）（低消費モード固定の設定は CPU クロックが 5MHz 以下の場合のみ）

EEPROM エミュレーション・ライブラリの実行を含む期間中割り込み禁止にしたい場合は、割り込みマスク・フラグをセットしてから EEPROM エミュレーション・ライブラリを実行し、実行後に DI() を行ってから割り込みマスク・フラグをクリアする必要があります。この時点では割り込みマスク・フラグはセットされているので DI() のみ行います。（詳細は「4.6 低電圧割り込み処理」を参照してください）

HALT 命令実行

割り込み処理を行わない場合はかならず手動でクリア

HALT 命令実行

(1)　クロック発生回路の動作／停止の設定

各クロック発生を動作させる場合は，動作を開始してからクロックの発振安定時間をウエイトする必要があります。

高速内蔵発振回路は，ステータス（RSTS = 1：安定動作）を参照することで発振の安定を確認することができます。

XT1発振子によるサブシステム・クロックは，タイマなどでウエイトする必要があります。プログラムではタイマTM00で2秒を計測しています（発振安定時間はお使いの発振子のメーカにお問い合わせください）。

高速内蔵発振回路を停止させる場合は，CPUクロック（f_{CPU}）に他の発振クロック（高速システム・クロックかサブシステム・クロック）から供給されていることを確認してください。

リスト4－4　クロック回路動作／停止（main.c）

```
/************************************************************************************************
;         Set_HispeedIntClk()
;-----------------------------------------------------------------------------------------------
;         高速内蔵発振器を動作させる
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Start_HispeedIntClk(void)
{
          /* 高速内蔵発振器を動作させる */
          RSTOP = 0;

          /* 高速内蔵発振器の発振安定待ち */
          while (RSTS == 0)
          {
                    NOP();
          }
}
/************************************************************************************************
;          Stop_HispeedIntClk()
;-----------------------------------------------------------------------------------------------
;         内蔵高速発振回路を止める
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Stop_HispeedIntClk(void)
{
          /* CPU クロックが高速システム・クロックかサブシステム・クロックで動作している場合のみ停止 */
          if((CLS == 1)||(MCS == 1))
          {
                    RSTOP = 1;
          }
}

/************************************************************************************************
;         Set_SubSystemClk()
;-----------------------------------------------------------------------------------------------
;         サブシステム・クロックを動作させる
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Start_SubSystemClk(void)
{
          /* サブシステム・クロックを XT1 発振モード（水晶発振子接続）に設定する */
          XTSTART = 1;
}
```

停止する前に高速内蔵システム・クロック以外が CPU クロックに供給されていることを確認

【コラム】　スタートアップ・ルーチンについて

CC78K0コンパイラ標準のスタートアップ・ルーチンでは，hdwinit()関数を呼び出したあとに変数（初期値あり，なしの両方）の初期設定を行ってから，main()関数を呼び出します。hdwinit()関数内で変数に値を設定すると，そのあとに初期化されるので注意が必要です。

(2)　CPUクロック（f_{CPU}）の切り替え

CPUクロックの供給元あるいは分周比を切り替えた場合は，完全に切り替わるまで時間を要します。

メイン・システム・クロックからサブシステム・クロックへの切り替え，あるいはその反対の場合は，CLSビットで確認できますが，メイン・システム・クロックから供給されていてプリスケーラの分周比を切り替える場合は，クロックの切り替えに必要な時間をウエイトします。

リスト4－5　CPUクロック切り替え（main.c）

```
/************************************************************************************************************
;       Change_Clock()
;------------------------------------------------------------------------------------------------------------
;       CPU クロックの供給元を切り替える
;------------------------------------------------------------------------------------------------------------
;       [I N]   UCHAR clk
;       [OUT]   -
;************************************************************************************************************/
void Change_CPUClk(UCHAR clk)
{
        switch(clk)
        {
                /* CPU クロックをメイン・システム・クロックの 2 分周から分周なしとする */
                case MAINSYSCLK2_1:
                {
                        /* PCC2,PCC1,PCC0=0（分周なし）*/
                        PCC = PCC & 0b11111000;

                        /* クロック切り替え（fXP/2→fXP）に必要な時間（8 クロック）ウエイトする */
                        /* NOP は 2 クロック必要とするので 8/2=4  */
                        NOP_4

                        break;
                }

                /* CPU クロックをメイン・システム・クロックの分周なしから 2 分周とする */
                case MAINSYSCLK1_2:
                {
                        /* PCC2,PCC1=0,PCC0=1（2 分周）*/
                        PCC = PCC | 0b00000001;

                        /* クロック切り替え（fXP→fXP/2）に必要な時間（16 クロック）ウエイトする */
                        /* NOP は 2 クロック必要とするので 16/2=8  */
                        NOP_8

                        break;
                }

                /* CPU クロックをメイン・システム・クロックからサブシステム・クロック（32.768KHz）の 2 分周とする */
                case MAINSYSCLK_SUBCLK:
                {
                        /* CPU クロックをサブシステム・クロックから供給する */
                        CSS = 1;

                        /* CLS ビット（1=サブシステム・クロック）で CPU クロックのステータスを確認する）  */
                        while(CLS == 0)
                        {
                                NOP();
                        }

                        break;
                }

                /* CPU クロックをサブシステム・クロック（32.768KHz）の 2 分周からメイン・システム・クロックとする */
                case SUBCLK_MAINSYSCLK:
                {
                        /* PCC2,PCC10=0 PCC0=1（2 分周）*/
                        PCC = PCC | 0b00000001;

                        /* CPU クロックをメイン・システム・クロックから供給する */
                        CSS = 0;

                        /* CLS ビット（0=メインシステム・クロック）で CPU クロックのステータスを確認する）  */
                        while(CLS == 1)
                        {
                                NOP();
                        }
                }
        }
}
```

マクロ定義により NOP を 4 回展開する

マクロ定義により NOP を 8 回展開する

メイン・システム・クロックの分周比切り替えの場合は必要な時間ウエイトする

メイン・システム・クロック⇔サブシステム・クロックの切り替えの場合は CLS で確認する

(3)　低消費電力モードの設定

レギュレータ・モード制御レジスタ（RMC）に56Hを設定します。

リスト4－6　低消費電力モードの設定（main.c）

```
/************************************************************************************************
;       Set_LowPowerMode()
;------------------------------------------------------------------------------------------------
;       低消費電力モードに設定する
;------------------------------------------------------------------------------------------------
;       [I N]    -
;       [OUT]    -
;************************************************************************************************/
void Set_LowPowerMode(void)
{
        /* レギュレータ設定（低消費電力モード（2.0V）固定）*/
        RMC = 0x56;
}
```

(4)　EEPROMエミュレーションによるフラッシュ・メモリの読み出し

フラッシュ・メモリに表示モードが書き込まれているかどうか読み出します。表示モードが書き込まれている場合（低電圧検出割り込みが入った場合）は，その内容を変数に設定します。表示モードが書き込まれていない場合は初期値（秒を表示，12時間表記）を変数に設定します。

リスト4－7　フラッシュ・メモリの読み出し（main.c）

```
/************************************************************************************************
;       Read_ClockMode()
;------------------------------------------------------------------------------------------------
;       表示モード（12/24 表記）を EERPOM エミュレーション・ライブラリで読み出す
;------------------------------------------------------------------------------------------------
;       [I N]    -
;       [OUT]    -
;************************************************************************************************/
void Read_ClockMode(void)
{
        UCHAR Result;

        /* FLMDPUP 設定 */
        FLMDPUP = 1;

        /* EEPROM エミュレーションのための初期設定 */
        Init_EEPROM();

        /* EEPROM エミュレーションで表示モードを読み出して初期値として設定 */
        Result = ucEEPROMReadEx_A( EEPROMDATA_NO, Clock_Mode);

        /* FLMDPUP クリア */
        FLMDPUP = 0;

        /* 読み出しエラーの場合は初期値（秒、12 時間表示）を設定 */
        if (Result != EEE_NORMAL)
        {
                Clock_Mode[ MODE_HHMMSS] = SS;
                Clock_Mode[ MODE_AMPM] = MODE_12;
        }
}
```

FLMDPUP の設定

【コラム】　FLMDPUPビットについて

78K0/Kx2-Lシリーズのマイクロコントローラでは，セルフ・プログラミングを行う場合FLMDPUPビットを1に設定する必要があります。EEPROMエミュレーション・ライブラリを使用する場合は，ライブラリの実行前に1に設定し，ライブラリ実行後に0に戻してください。

(5)　リアルタイム・カウンタのAMPMビットの設定

main()関数では，フラッシュ・メモリから表示モードを読み出したあとに，リアルタイム・カウンタのAMPMビットを設定しています。

サンプル・プログラムでは初期設定で12時間表示の値を時カウント・レジスタHOURに設定しているため，表示モードを12時間表示から24時間表示に変更する場合は，設定値を修正しています。また，AMPMビットを12時間表示から24時間表示に変更すると時カウント・レジスタの値が0になるので，必ずその前に退避させる必要があります。

設定値の修正については「4.5　INTP1, INTP4割り込み処理」を参照してください。

リスト4－8　AMPMビットの設定（main.c）

```
/************************************************************************************************************
;         Set_ampm()
;------------------------------------------------------------------------------------------------------------
;         リアルタイム・カウンタの AMPM ビットを変更する
;------------------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************************/
void Set_ampm(void)
{

          UCHAR tmp;

          /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給 */
          RTCEN     = 1;                                                    ← 制御クロック供給

          /* 12 時間表示の場合 */
          if(Clock_Mode[MODE_AMPM] == 0)
          {
                    /* AMPM (3 ビット目) 0 (12 時間表記) に設定 */
                    RTCC0 = RTCC0 &  0b11110111;

          }

          /* 24 時間表示の場合 */
          else
          {
                    /* 時間レジスタを読み出す (AMPM ビットを変更する前に退避する) */
                    tmp = HOUR;

                    /* AMPM (3 ビット目) を 1 (24 時間表記) に設定 */
                    RTCC0 = RTCC0 |  0b00001000;

                    /* 12 時間表示を 24 時間表示に修正して設定 */
                    /* 12H なら 00H を設定 */
                    if(tmp == 0x12)
                    {
                              HOUR = 0x00;
                    }
                    /* 32H なら 12H を設定 */
                    else if(tmp == 0x32)
                    {
                              HOUR = 0x12;
                    }
                    /* 21H ～ 27H,30H,31H なら 0EH を引く */
                    else if((0x21 <= tmp && tmp <= 0x27)||(tmp==0x30)||(tmp==0x31))
                    {
                              HOUR = tmp - 0x0e;
                    }
                    /* 28H,29H なら 08H を引く */
                    else if((tmp==0x28)||(tmp==0x29))
                    {
                              HOUR = tmp - 0x08;
                    }

          }

          /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給停止 */
          RTCEN     = 0;                                                    ← 制御クロック供給停止

}
```

4. 4　リアルタイム・カウンタ割り込み処理

サンプル・プログラムではリアルタイム・カウンタのアラーム割り込みを使って1秒ごとに表示を更新しています。

時，分，秒の各カウント・レジスタを読み出すときは，リアルタイム・カウンタ・コントロール・レジスタ1（RTCC1）のRWAITビットを1（カウンタ停止設定，読み出し／書き込みモード）に設定していったんカウンタを停止し，RWSTフラグでカウンタが停止しているかを確認する必要があります。ただし，リアルタイム・カウンタが停止している状態（RTCE = 0）で各レジスタを設定する場合はRWAITビットの設定は必要ありません。リアルタイム・カウンタが停止している状態で，RWATIビットを設定してもRWSTフラグは変化しません。

リスト4－9　リアルタイム・カウンタ割り込み　（Int_RTC.c）　1

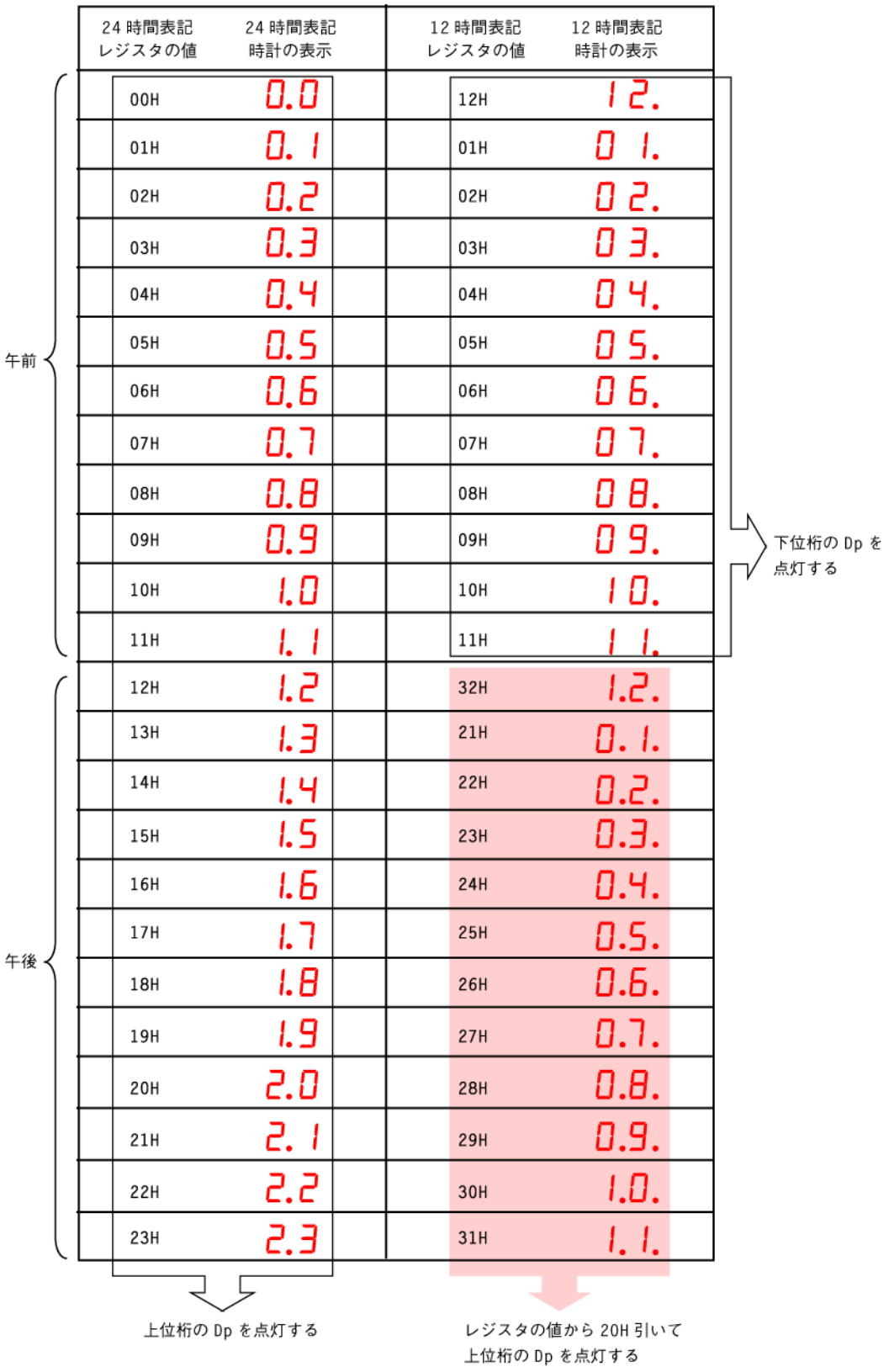
```
/*-----------------------------------------------------------------------
*         前処理指令（#pragma 指令）
*-----------------------------------------------------------------------*/
#pragma   sfr
#pragma   nop
#pragma interrupt INTRTC Int_RTC

/*-----------------------------------------------------------------------
*         ヘッダ・ファイル インクルード
*-----------------------------------------------------------------------*/
#include  "Kx2_eee.h"                 /* サンプル・プログラム・ヘッダ */
#include  "LowPower.h"                /* サンプル・プログラム・ヘッダ */
/*-----------------------------------------------------------------------
*         プロトタイプ宣言
*-----------------------------------------------------------------------*/
void Set_DisplayData_by_ssmmddMode( UCHAR *);
void Set_DisplayData_by_12_24Mode( UCHAR *);
void Display_LED( UCHAR *);
/*-----------------------------------------------------------------------
*         外部参照（変数）
*-----------------------------------------------------------------------*/
extern UCHAR Clock_Mode[EEPROM_DATALEN];   ← 表示モード
extern UCHAR RTC_Data[3];                  ← 時計データ
extern UCHAR const LED_Pattern[];

/*****************************************************************************************************************
;         RTC_1sec()  リアルタイム・カウンタ割り込み関数
;-----------------------------------------------------------------------------------------------------------------
;         1 秒ごとに呼び出されます
;-----------------------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;*****************************************************************************************************************/
void Int_RTC(void)
{
          UCHAR display_data[2];

          /* カウンタ停止、カウンタ値読み出し / 書き込みモード */
          RWAIT = 1;                       ← カウンタ値読み出し / 書き込みモードに設定

          /* カウンタ読み出し / 書き込みモードになるまでウエイト */
          while(RWST == 0)
          {
                    NOP();
          }

          /* カウンタ・レジスタの値を読む */
          RTC_Data[SS] = SEC;
          RTC_Data[MM] = MIN;              ← レジスタ読み出し
          RTC_Data[HH] = HOUR;

          /* カウンタ動作モード */
          RWAIT = 0;                       ← カウンタ動作モードに戻す

          /* 表示モード（時 / 分 / 秒）にしたがって表示データを設定する */
          Set_DisplayData_by_ssmmddMode( display_data );

          /* 7 セグメント LED に表示する */
          Display_LED( display_data);

          /* 割り込み要求フラグをクリアしておく */
          RTCIF = 0;

}
```

時カウント・レジスタの値と，7セグメントLEDの表示パターンの対応を以下に示します。

24時間表記の場合は上位桁に，12時間表記の場合は下位桁にDpを点灯させます。また，12時間表記の午後時間の場合は上位桁にもDpを点灯させます。

12時間表記の場合，21H以上の値は実際の時計表示と異なりますので，表示する前に加工する必要があります。

図4－5　時カウント・レジスタと時計表示パターンの対応



リスト4－9　リアルタイム・カウンタ割り込み　（Int_RTC.c）　2

```
void Set_DisplayData_by_ssmmddMode( UCHAR *display_data)
{
        UCHAR tmp;

    switch (Clock_Mode[MODE_HHMMSS])
    {
            /* 分を表示する */
        case MM:
        {
                tmp = RTC_Data[MM];
                display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
                display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
                break;
        }

            /* 時を表示する */
        case HH:
        {
                /* 表示モード（12/24 時間表示）にしたがって時の表示データを設定する */
                Set_DisplayData_by_12_24Mode( display_data );
                break;
        }

            /* 0 かその他の値なら秒を表示する */
        default:
        {
                tmp = RTC_Data[SS];
                display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
                display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
        }
    }
}
```

リスト4－9　リアルタイム・カウンタ割り込み　（Int_RTC.c）　3

```
void Set_DisplayData_by_12_24Mode( UCHAR *display_data)
{

        UCHAR tmp;

        tmp = RTC_Data[HH];

        /* 12/24 時間表示の区別 */
    if (Clock_Mode[MODE_AMPM] == 0)
    {
            /* PM12 時～ PM11 時の表示データを修正して上位桁の DP を点灯 */
        if (tmp >= 0x21)
        {
                tmp = tmp - 0x20;
                display_data[HIGH] = (LED_Pattern[((tmp & 0xf0)>>4)])& 0b01111111;
        }
        else
        {
                display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
        }
            /* 12 時間表示なら下位桁の DP を点灯 */
            display_data[LOW] = (LED_Pattern[(tmp & 0x0f)])& 0b01111111;
    }
    else
    {
            /* 24 時間表示なら上位桁の DP を点灯 */
            display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
            display_data[HIGH] = (LED_Pattern[((tmp & 0xf0)>>4)]) & 0b01111111;
    }
}
```

レジスタの値が 21H 以上の場合は 20H を引いて上位桁に Dp を点灯する

12 時間表記なら下位桁の Dp を表示

24 時間表記なら上位桁の Dp を表示

【コラム】　周辺ハードウエア・クロック供給停止中のポート・レジスタの書き換えについて

リアルタイム・カウンタ，INTP1，INTP4の各割り込み中は，高速内蔵発振回路は停止しているので周辺ハードウエアへはクロックが供給されていませんが，ポート・レジスタの書き換えはCPUクロック（f_{CPU}）が供給されていれば可能です。

4. 5　INTP1，INTP4割り込み処理

スイッチによる時／分／秒の表示切り替えをINTP1, 12/24時間表示切り替えをINTP4割り込みで行っています。

(1)　INTP1割り込み処理

割り込みが入るたびに表示モードを切り替えます。表示モードにしたがって，7セグメントLEDの表示も切り替えます。

リスト4－10　INTP1割り込み処理（Int_Intp.c）

```
/*---------------------------------------------------------------------
*       前処理指令（#pragma 指令）
*---------------------------------------------------------------------*/
#pragma  sfr
#pragma  nop
#pragma interrupt INTP1 SW_hhmmss
#pragma interrupt INTP4 SW_12_24

/*---------------------------------------------------------------------
*       ヘッダ・ファイル インクルード
*---------------------------------------------------------------------*/
#include "Kx2_eee.h"                /* サンプル・プログラム・ヘッダ */
#include "LowPower.h"               /* サンプル・プログラム・ヘッダ */

/*---------------------------------------------------------------------
*       外部参照（関数）
*---------------------------------------------------------------------*/
extern void Set_DisplayData_by_ssmmddMode( UCHAR *);
extern void Set_DisplayData_by_12_24Mode( UCHAR *);
extern void Display_LED( UCHAR *);
/*---------------------------------------------------------------------
*       外部参照（変数）
*---------------------------------------------------------------------*/
extern UCHAR      Clock_Mode[EEPROM_DATALEN];   ← 表示モード
extern UCHAR      RTC_Data[3];                  ← 時計データ

/*****************************************************************************************************************
;       SW_ttmmss()  INTP1 割り込み関数
;-----------------------------------------------------------------------------------------------------------------
;       スイッチ SW5 を押し下げるたびに呼び出されます
;-----------------------------------------------------------------------------------------------------------------
;       [I N]    -
;       [OUT]    -
;*****************************************************************************************************************/
void SW_hhmmss(void)
{
        UCHAR display_data[2];      /* 表示データ */

        /* ポートの状態を見る（SW チャタリング対策）*/
        if(P3.0 == 0)
        {

                switch (Clock_Mode[MODE_HHMMSS])
                {
                        /* 秒なら分を表示 */
                        case SS:
                        {
                                Clock_Mode[MODE_HHMMSS] = MM;
                                break;
                        }

                        /* 分なら時間を表示 */
                        case MM:
                        {
                                Clock_Mode[MODE_HHMMSS] = HH;
                                break;
                        }

                        /* 時間あるいはそれ以外なら秒を表示 */
                        default:
                        {
                                Clock_Mode[MODE_HHMMSS] = SS;
                        }
                }

                /* 表示モード（時 / 分 / 秒）にしたがって表示データを設定する */
                Set_DisplayData_by_ssmmddMode( display_data );

                /* 7 セグメント LED に表示する */
                Display_LED( display_data);

                /* 割り込み要求フラグをクリアしておく */
                PIF1 = 0;
        }
}
```

(2)　INTP4割り込み

12時間表記から24時間表示に切り替える場合（AMPMビットを0から1に設定）は，時カウント・レジスタHOURが0になるのでレジスタの内容を退避させる必要があります。また，12時間表示の値と24時間表記の値は異なりますので，変換してから時カウント・レジスタHOURに復帰させる必要があります。

リスト4－11　INTP4割り込み処理（Int_Intp.c）　1

```
void SW_12_24(void)
{
        UCHAR tmp;                      /* 退避用テンポラリ */
        UCHAR display_data[2];          /* 表示データ */

        /* ポートの状態を見る（SW チャタリング対策）*/
        if(P3.3 == 0)
        {
                /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給 */
                RTCEN    = 1;                    ← 制御クロック供給

                /* カウンタ停止、カウンタ値読み出し / 書き込みモード */
                RWAIT = 1;                       ← カウンタ値読み出し / 書き込みモードに設定

                /* カウンタ読み出し / 書き込みモードになるまでウエイト */
                while(RWST == 0)
                {
                        NOP();
                }

                /* 時カウント・レジスタを退避 */
                tmp = HOUR;                      ← レジスタ退避

                /* 24 なら 12 時間表示で表示 */
                if(Clock_Mode[MODE_AMPM] == MODE_12)
                {
                        /* 12←24 表示切り替え */
                        Clock_Mode[MODE_AMPM] = MODE_24;

                        /* AMPM（3 ビット目）を 1（24 時間表示）に設定
                        RTCC0 = RTCC0 |  0b00001000;

                        /* 12 時間表記を 24 時間表示に修正して復帰 */
                        if(tmp == 0x12)
                        {
                                HOUR = 0x00;
                        }
                        else if(tmp == 0x32)
                        {
                                HOUR = 0x12;
                        }
                        else if((0x21 <= tmp && tmp <= 0x27)
                                                        ||(tmp==0x
                                                        ||(tmp==0x
                        {
                                HOUR = tmp - 0x0e;
                        }
                        else if((tmp==0x28)||(tmp==0x29))
                        {
                                HOUR = tmp - 0x08;
                        }
                        else
                        {
                                HOUR = tmp;
                        }
                        RTC_Data[HH] = HOUR;
                }
```

24 時間表記 レジスタの値	12 時間表記 レジスタの値	
00H	12H	00H⇔12H 変換する
01H	01H	加工しない
02H	02H	
03H	03H	
04H	04H	
05H	05H	
06H	06H	
07H	07H	
08H	08H	
09H	09H	
10H	10H	
11H	11H	
12H	32H	12H⇔32H 変換する
13H	21H	0EH 差をとる
14H	22H	
15H	23H	
16H	24H	
17H	25H	
18H	26H	
19H	27H	
20H	28H	08H 差をとる
21H	29H	
22H	30H	0EH 差をとる
23H	31H	

リスト4－11　INTP4割り込み処理（Int_Intp.c）　2

```
        /* 24 あるいはそれ以外なら 12 時間表示で表示 */
        else
        {
                /* 12→24 表示切り替え */
                Clock_Mode[MODE_AMPM] = MODE_12;

                /* AMPM (3 ビット目) 0 (12 時間表示) に設定 */
                RTCC0 = RTCC0 &  0b11110111;

                /* 24 時間表記を 12 時間表示に修正して復帰 */
                if(tmp == 0x00)
                {
                        HOUR = 0x12;
                }
                else if(tmp == 0x12)
                {
                        HOUR = 0x32;
                }
                else if((0x13 <= tmp && tmp <= 0x19)||(tmp==0x22)||(tmp==0x23))
                {
                        HOUR = tmp + 0x0e;
                }
                else if((tmp==0x20)||(tmp==0x21))
                {
                        HOUR = tmp + 0x08;
                }
                else
                {
                        HOUR = tmp;
                }
                RTC_Data[HH] = HOUR;
        }

        /* カウンタ動作モード */
        RWAIT = 0;          ← カウンタ動作モードに戻す

        /* リアルタイム・カウンタへのライト・アクセスの制御クロック供給停止 */
        RTCEN     = 0;      ← 制御クロック供給停止

        /* 表示モード (時 / 分 / 秒) にしたがって時の表示データを設定する */
        Set_DisplayData_by_ssmmddMode( display_data );

        /* 7 セグメント LED に表示する */
        Display_LED( display_data);
    }

    /* 割り込み要求フラグをクリアしておく */
    PIF4 = 0;

}
```

4. 6　低電圧検出割り込み処理

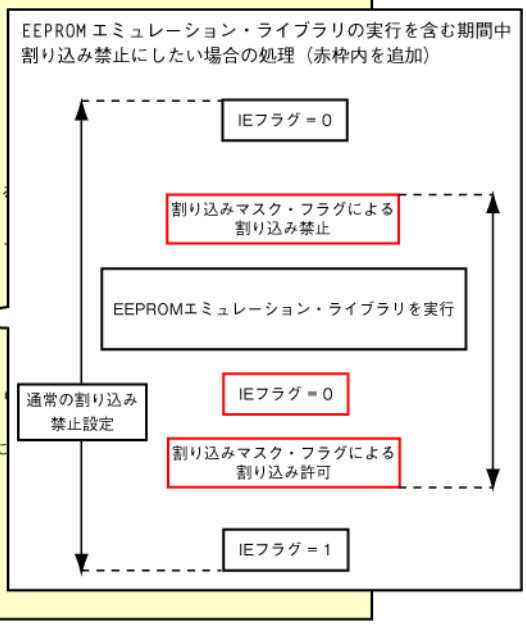
2.53 V±0.1 Vの低電圧を検出すると低電圧検出割り込みが発生します。

割り込み処理では，EEPROMエミュレーション・ライブラリでフラッシュ・メモリに表示モードを書き込むため高速内蔵発振回路を動作させ，CPUクロック（f_{CPU}）をいったん高速内蔵発振クロック（4 MHz（TYP.））からの供給に切り替えます（フラッシュ・メモリにセルフ・プログラミングする場合，サブシステム・クロック以外のクロックがCPUクロック（f_{CPU}）に供給されている必要があります）。書き込んだあとは，CPUクロック（f_{CPU}）を再びサブシステム・クロック（f_{SUB}）からの供給に戻して，高速内蔵発振回路を停止させます。

EEPROMエミュレーション・ライブラリ実行時に割り込み禁止にしたい場合は，他の割り込みマスク・フラグをセットする必要があります。

リスト4－12　LVI割り込み処理（Int_LVI.c） 1

```
/*-----------------------------------------------------------------------
*         前処理指令（#pragma 指令）
*-----------------------------------------------------------------------*/
#pragma sfr
#pragma interrupt INTLVI LVI_vdd

/*-----------------------------------------------------------------------
*         ヘッダ・ファイル インクルード                EEPROM エミュレーション・ライブラリ ヘッダ・ファイル
*-----------------------------------------------------------------------*/
#include  "eeelib.h"                  /* EEPROM エミュレーション・ライブラリヘッダ */
#include  "Kx2_eee.h"                 /* サンプル・プログラム・ヘッダ */
#include  "LowPower_01.h"             /* サンプル・プログラム・ヘッダ */

/*-----------------------------------------------------------------------
*         プロトタイプ宣言
*-----------------------------------------------------------------------*/
UCHAR EEPROM_Write(const UCHAR, const UCHAR *, UCHAR *);

/*-----------------------------------------------------------------------
*         外部参照（関数）
*-----------------------------------------------------------------------*/
extern void Write_ClockMode(void);
extern void Change_CPUClk(UCHAR clk);
extern void Set_HispeedIntClk(void);
extern void Stop_HispeedIntClk(void);

/*-----------------------------------------------------------------------
*         外部参照（変数）
*-----------------------------------------------------------------------*/
extern UCHAR        Clock_Mode[EEPROM_DATALEN];   表示モード

/*****************************************************************************************************
;          LVI_vdd()
;-----------------------------------------------------------------------------------------------------
;         2.53V ± 0.1V で割り込みが発生します
;-----------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;*****************************  電源電圧＜LVI 検出電圧なら電圧が降下しているとみなす  ******************/
void LVI_vdd(void)
{
        /* 電源電圧＜LVI 検出電圧 */
        if(LVIF == 1)
        {
                /* 警告 LED 点灯 */
                LED_LVI = 0;

                /* 高速内蔵発振回路を動作させる */
                Start_HispeedIntClk();

                /* CPU クロックを高速内蔵発振回路 4MHz TYP. に切り替え
                Change_CPUClk(SUBCLK_MAINSYSCLK);

                /* 他の割り込み（RTC,INTPO,INTP4）マスク・フラグをセ
                Set_IntMsk_RTC_INTP();

                /* フラッシュ・メモリに表示モードを書き込む */
                Write_ClockMode();

                /* 割り込み禁止 */
                DI();

                /* 他の割り込み（RTC,INTPO,INTP4）マスク・フラグをク
                Clear_IntMsk_RTC_INTP();

                /* CPU クロックをサブシステム・クロック（32.768KHz）に
                Change_CPUClk(MAINSYSCLK_SUBCLK);

                /* 高速内蔵発振回路を止める */
                Stop_HispeedIntClk();
        }

        /* 割り込み要求フラグをクリアしておく */
        LVIIF = 0;
}
```



EEPROMエミュレーション・ライブラリによる書き込みを実行した際に書き込むブロックが一杯の場合は，戻り値としてデータ・フルを返します。その場合は，書き込みブロックを変更します。

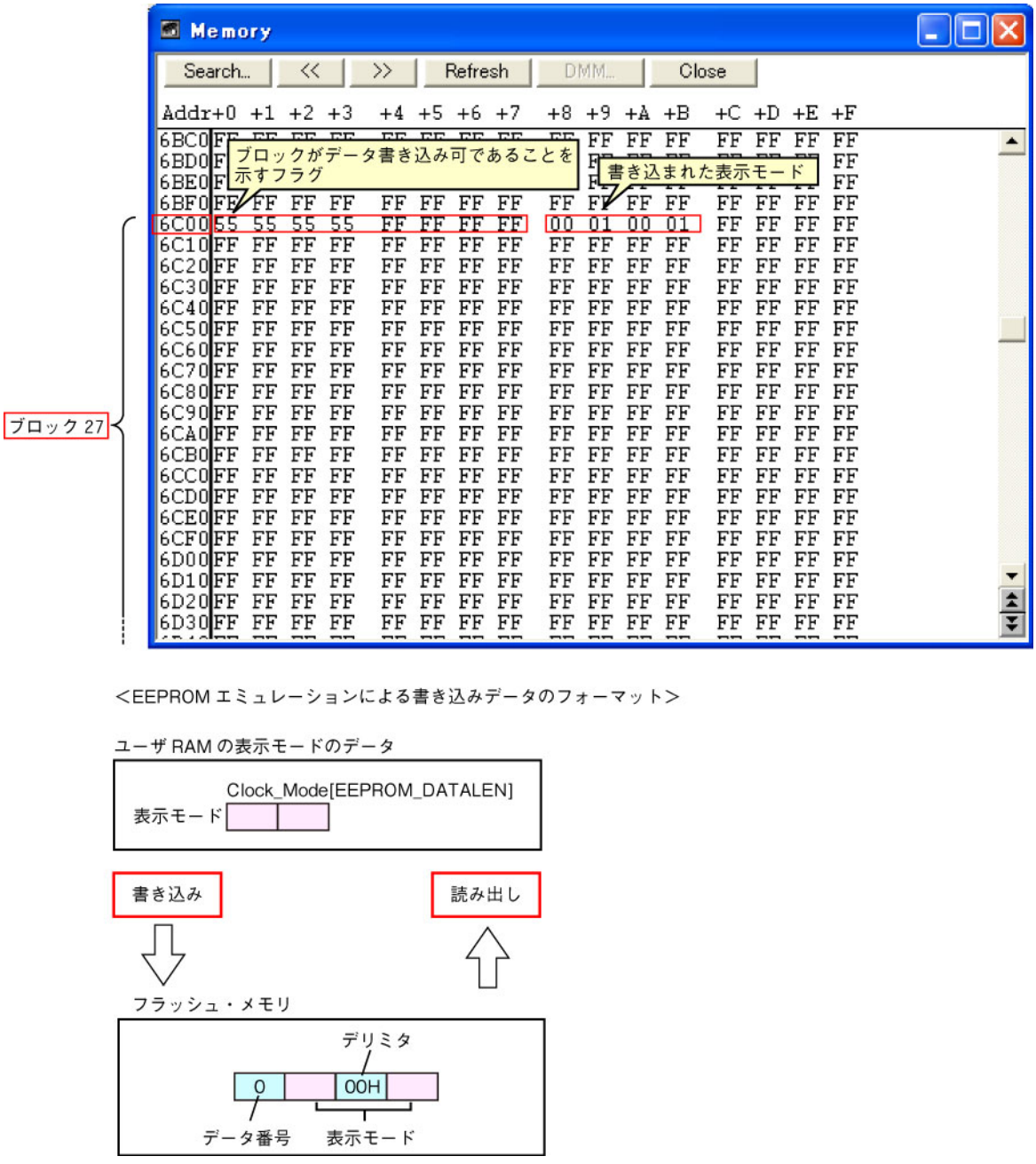
リスト4－12　LVI**割り込み処理**（Int_LVI.c）　2

```
/************************************************************************************************************
;         Write_ClockMode()
;-----------------------------------------------------------------------------------------------------------
;         現在の表示モードをフラッシュ・メモリに書き込む
;-----------------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************************/
void Write_ClockMode(void)
{
          UCHAR temp;

          FLMDPUP = 1;                                                    FLMDPUP の設定
          EEPROM_Write(EEPROMData_No,Clock_Mode,&temp);
          FLMDPUP = 0;

}

/************************************************************************************************************
;         EEPROM_Write()
;-----------------------------------------------------------------------------------------------------------
;         EEPROM エミュレーション・ライブラリでデータを保存する
;-----------------------------------------------------------------------------------------------------------
;         [I N]     const UCHAR ucEEPROMNo, const UCHAR *ucEEPROMBuf, UCHAR *ChangeBlock_Count
;         [OUT]     UCHAR Result
;************************************************************************************************************/
UCHAR EEPROM_Write( const UCHAR ucEEPROMNo, const UCHAR *ucEEPROMBuf, UCHAR *ChangeBlock_Count )
{
          /* ローカル変数 */
          UCHAR     i;                  /* ループカウンタ */
          UCHAR     Result;             /* EEPROM エミュレーション・ライブラリの戻り値         */

          /* EEPROM エミュレーション・ライブラリでフラッシュ・メモリに書き込む */
          Result = ucEEPROMWriteEx_A( ucEEPROMNo, ucEEPROMBuf );

          /* データフルの場合    */
          if (Result == ERRFULL)                                          ブロックが一杯の場合はブロックを変更する
          {

                    /* EEPROM エミュレーション・ライブラリでブロックを変更 */
                    Result = ucEEPROMChangeEx_A();
                    if (( Result == EEE_NORMAL ) || ( Result == NMLBLK ))
                    {
                              /* ブロック変更カウンタをインクリメントする（0 ～ FF までカウント可能とする）*/
                              (*ChangeBlock_Count) = (*ChangeBlock_Count)  + 1;

                              /* ブロック変更したら新しい有効ブロックに EEPROM エミュレーション・ライブラリで書き込む */
                              Result = ucEEPROMWriteEx_A( ucEEPROMNo, ucEEPROMBuf );

                              /* 書き込みエラーの場合は終了 */
                              if ( Result != EEE_NORMAL )
                              {
                                        return( Result );
                              }

                              /* EEPROM エミュレーション・ライブラリで前の有効ブロックを消去しておく */
                              Result = ucEEPROMEraseEx_A();
                    }
          }
          return( Result );
}
```

以下にEEPROMエミュレーション・ライブラリで書き込まれたフラッシュ・メモリを示します（MINICUBE2による確認）。EEPROMエミュレーション・ライブラリ用のヘッダ・ファイル Kx2_eee.hにより書き込むブロック番号を27（1BH）と指定しているので，6C00H番地より書き込まれます（EEPROMエミュレーション・ライブラリでは，指定したブロック番号から4ブロック分を，データの書き込み領域としています）。

EEPROMエミュレーション・ライブラリでは，書き込むデータにデータ番号，デリミタを追加し，フラッシュ・メモリの書き込みの最小単位である4バイトの倍数長に調整してから書き込みます。2バイトを書き込む場合はデータ番号，デリミタを追加すると4バイトになるので4の倍数長に調整しません。データ番号は先頭に，デリミタ（00H）は最終バイトよりひとつ前に追加されます。

図4－6　表示モードが書き込まれたフラッシュ・メモリ



<EEPROM エミュレーションによる書き込みデータのフォーマット>

第5章　関連資料

資　料　名		PDF/資料番号
RA78K0　Ver.3.80　ユーザーズ・マニュアル[注1]	操作編	PDF
	言語編	PDF
	構造化アセンブリ言語編	PDF
RA78K0　Ver.4.01　使用上の留意点（文書）[注1]		ZUD-CD-07-0181
CC78K0　Ver.3.70　ユーザーズ・マニュアル[注2]	操作編	PDF
	言語編	PDF
CC78K0　Ver.4.00　使用上の留意点（文書）[注2]		ZUD-CD-07-0103
SM+　ユーザーズ・マニュアル	操作編	PDF
	ユーザ・オープン・インタフェース編	PDF
ID78K0-QB　Ver.2.94　ユーザーズ・マニュアル	操作編	PDF
ID78K0-QB　Ver.3.00　ユーザーズ・マニュアル	操作編	PDF
PM plus Ver.5.20[注3]　ユーザーズ・マニュアル		PDF
PM+ Ver.6.30[注4]　ユーザーズ・マニュアル		PDF

注 1. この資料は，RA78K0 Ver.4.01のインストール時に，ツール本体と一緒に，PCにインストールされます。「RA78K0　Ver.4.01 使用上の留意点（文書）」に記載されていない内容に関しては，RA78K0 Ver.3.80のユーザーズ・マニュアルを参照してください。

2. この資料は，CC78K0 Ver.4.00のインストール時に，ツール本体と一緒に，PCにインストールされます。「CC78K0　Ver4.00 使用上の留意点（文書）」に記載されていない内容に関しては，CC78K0 Ver.3.70のユーザーズ・マニュアルを参照してください。

3. PM plus Ver.5.20は，RA78K0 Ver.3.80に同梱されている統合開発環境です。

4. PM+ Ver.6.30は，RA78K0 Ver.4.01に同梱されている統合開発環境です。ソフトウエア・ツール（アセンブラ，Cコンパイラ，デバッガ，シミュレータ）の複数の異なるバージョン製品を管理することができます。

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

【発　行】

NECエレクトロニクス株式会社

〒211-8668　神奈川県川崎市中原区下沼部1753

電話（代表）：(044)435−5111

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。

URL（アドレス）　http://www.necel.co.jp/

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。

お問い合わせ先

【営業関係，デバイスの技術関係お問い合わせ先】

半導体ホットライン

（電話：午前 9:00〜12:00，午後 1:00〜5:00）

電　話　：(044)435−9494

E-mail　：info@necel.com

【マイコン開発ツールの技術関係お問い合わせ先】

開発ツールサポートセンター

E-mail　：toolsupport-micom@ml.necel.com

C0901-A-01