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。

M32C/85 グループ
リモコン受信

1. 要約

インテリジェント I/O の時間計測機能を使用して、リモコン受信を実現します。

2. はじめに

この資料で説明する応用例は、次のマイコン、条件での利用に適用されます。
・マイコン　　：M32C/85 グループ

M32C/85 グループと同様の SFR(周辺機能制御レジスタ)を持つ他の M16C ファミリでも本プログラムを使用することができます。ただし、一部の機能を機能追加等で変更している場合がありますのでマニュアルで確認してください。このアプリケーションノートをご使用に際しては十分な評価を行ってください。

3. リモコン受信概要

リモコンから送信した赤外線信号は、一定の周波数（キャリア周波数）で受信部に送信されます。赤外線信号は、受信部では拡散して弱くなっているため、赤外線受光素子の出力をプリアンプで増幅する必要があります。また、帯域フィルタ（BPF：BandPass Filter）を通すことで、キャリア周波数成分だけを抽出し、検波し、波形整形することで正確なリモコン信号が得られます。さらに、赤外線リモコン用プリアンプからは、負論理（反転）でデータが出力されます。なお、キャリア周波数は 38KHz です。
図 1 に受信概要ブロック図、図 2 に赤外線リモコン受信モジュール内部ブロック図、図 3 にキャリア波形を示します。

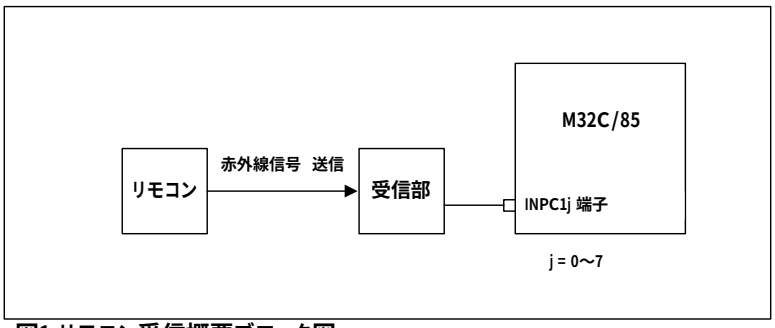

図1 リモコン受信概要ブロック図

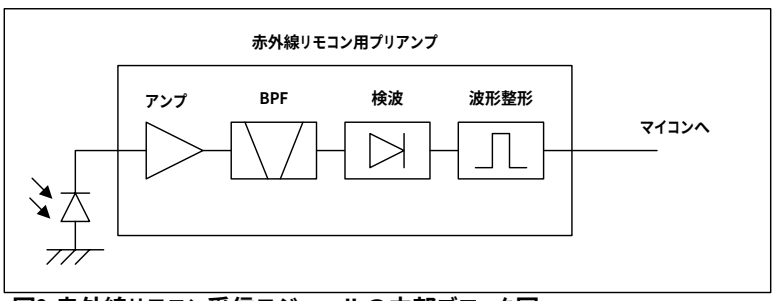

図2 赤外線リモコン受信モジュールの内部ブロック図

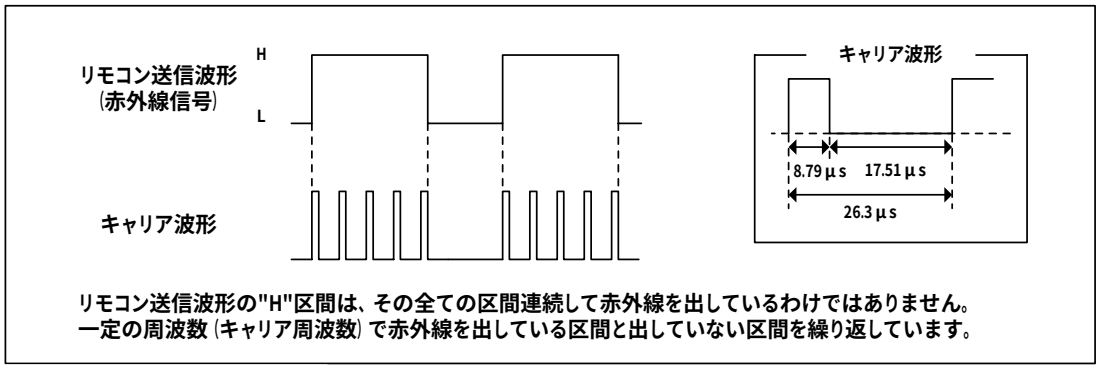

図3 キャリア波形

4. 時間計測機能

インテリジェント I/O の時間計測機能は、外部トリガ入力に同期し、ベースタイマの値を G1TMj レジスタ（j=0～7）に格納します。図 4 に時間計測機能の動作例を示します。

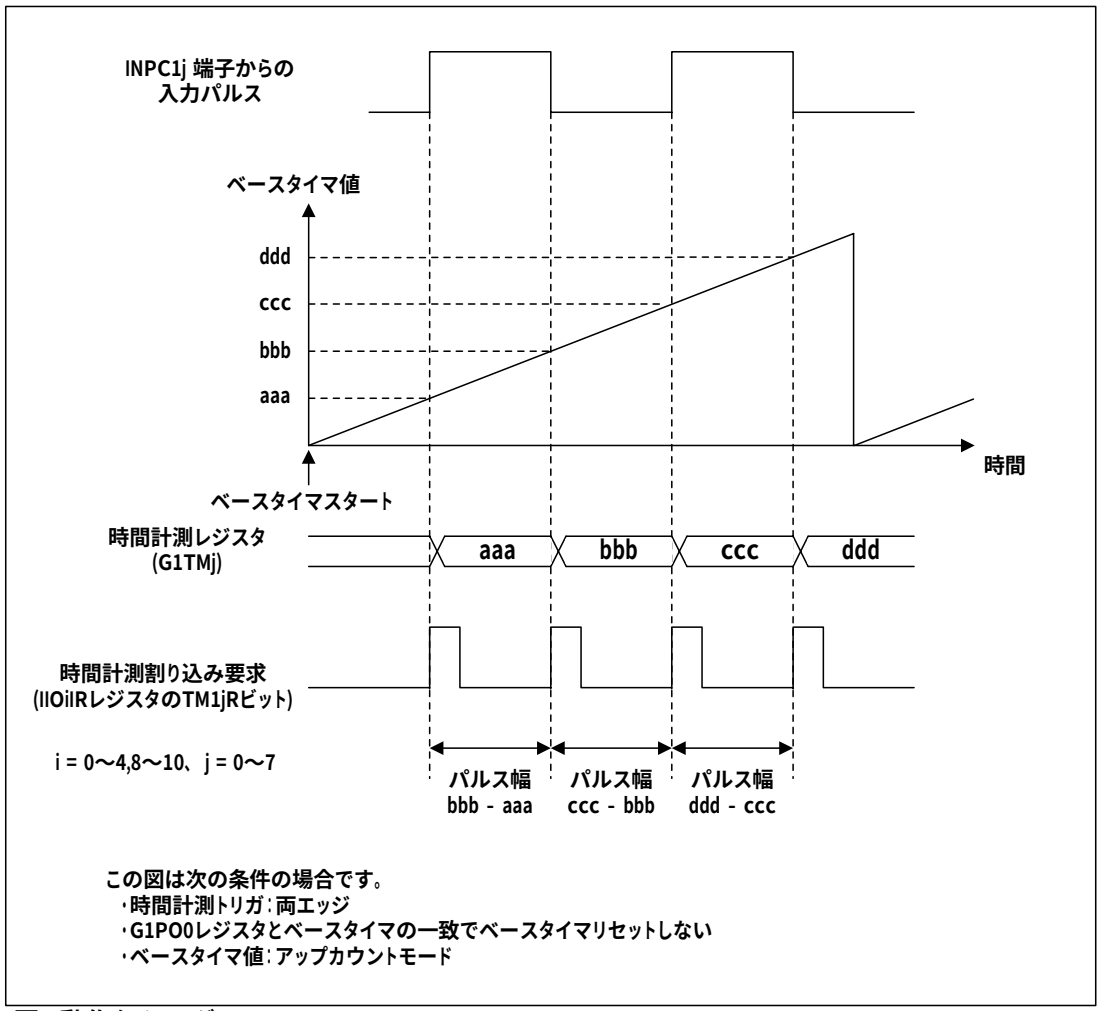

図4 動作タイミング

5. 動作説明

表 1 に、リソース一覧表を示します。

表 1 リソース一覧

機能	チャネル	設定値	用途
インテリジェント I/O	時間計測 1 チャネル	62 分周 カウントソース f1 両エッジトリガ	リモコン送信波形時間計測
タイマ	タイマ A0	1ms	1 フレームタイマ計測 (タイマ A1 のイベントカウント用)
	タイマ A1	140ms	1 フレームタイマ計測

― インテリジェント I/O ―

(1) ベースタイマを 62 分周、カウントソース f1 に設定します。
(2) ベースタイマの時間計測トリガを両エッジに設定します。
(3) ベースタイマスタートビットを“ 1 ”にすると、ベースタイマ値がアップカウントします。
(4) INPC10 端子の立ち上がり、もしくは、立ち下がりエッジのタイミングで、インテリジェント I/O 時間計測機能 0 割り込み要求ビットが“ 1 ”になります。
(5) その割り込み毎に、ベースタイマ値が格納されます。
(6) 格納されたベースタイマ値をパルス値に変換し、リモコンデータと比較します。

― タイマ ―

(1) リーダコードから 1 フレームを計測するために、タイマを使用します。
(2) タイマ A0 をタイマモード、タイマ A1 をイベントカウントモード（タイマ A0 のアンダフローをカウント）に設定します。
(3) タイマ A0 割り込みを 1ms 間隔にするため、タイマ値を（7D00-1）h に設定します。
(4) タイマ A1 割り込みを 140ms（1 フレーム＋30%誤差）間隔にするため、タイマ値を（8C-1）h に設定します。
(5) リーダコードを検知し始めた場合、タイマ A0,A1 のカウント開始フラグを“ 1 ”にし、タイマ動作させます。
(6) タイマ A1 割り込みで、割り込み要求ビットが“ 1 ”に設定された場合、タイマ A0,A1 を停止させます。

6. リモコンデータ検出仕様

- 1 回目のデータは、リーダコード、カスタムコード（8 ビット）、反転カスタムコード（8 ビット）、データコード（8 ビット）、反転データコード（8 ビット）、ストップビット（1 ビット）、フレームスペース（赤外線を出さない区間）の 1 フレーム（リーダコードからフレームスペースまで含めた区間）が、108ms 範囲以内に受信された場合、受信完了と判断します。
- 2 回目以降のデータは、リーダコード、ストップビット（1 ビット）、フレームスペースの 1 フレームが、108ms 範囲以内に受信された場合、受信完了と判断します。
- 各コードに対して、フォーマット値より±30%の誤差以内であれば、コード認識完了と判断します。また、1 フレームも 108ms＋30%誤差範囲以内とします。
- リーダコードが検出できた場合、カスタムコード、データコード、ストップビット、フレームスペースの順に検出します。
- 各コードで受信エラーが発生した場合、次の立ち上がり、もしくは、立ち下がりエッジをリーダコード（1 回目のデータ）として判断し始めます。
- リーダコードから 1 フレーム（＋30%誤差値を含む）以上経過した場合、受信データがフレームスペースを認識している途中であれば、受信完了と判断します。
- 1 フレーム範囲以内(＋30%誤差値を含む)に、フレームスペース後のリーダコードが検出された場合、その検知された受信データは 2 回目以降のデータとして認識されます。(＋30%の誤差分を含んでいるため、フレームスペース後、1 フレーム以内に 1 回目のリーダコードが受信する場合がある)
- 図 5 にリモコンデータフォーマット図を示します。

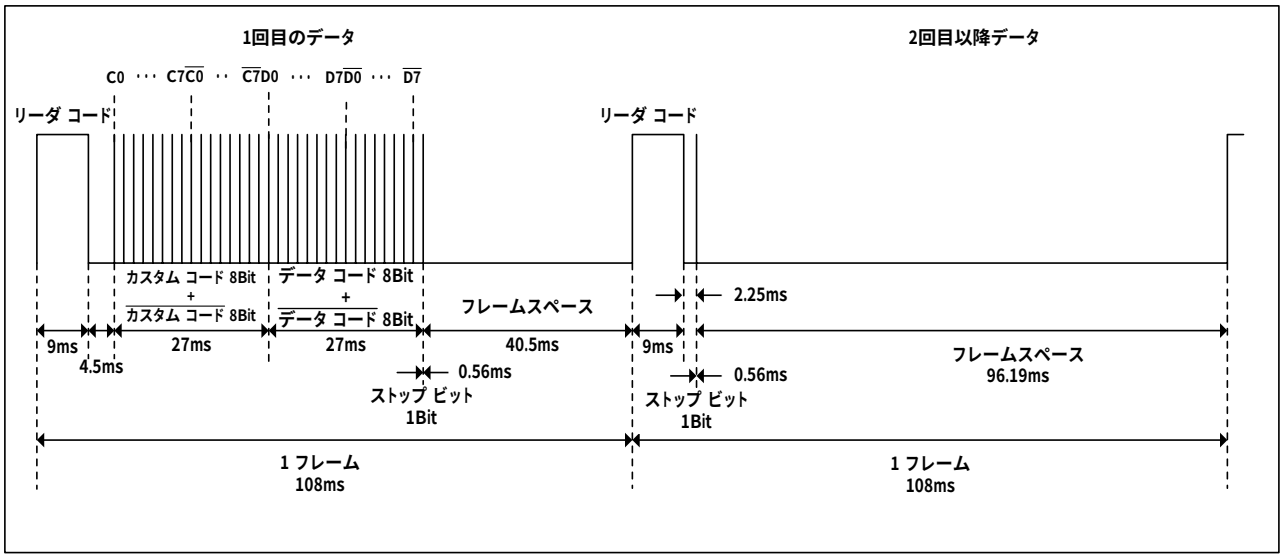

図5 リモコンデータフォーマット

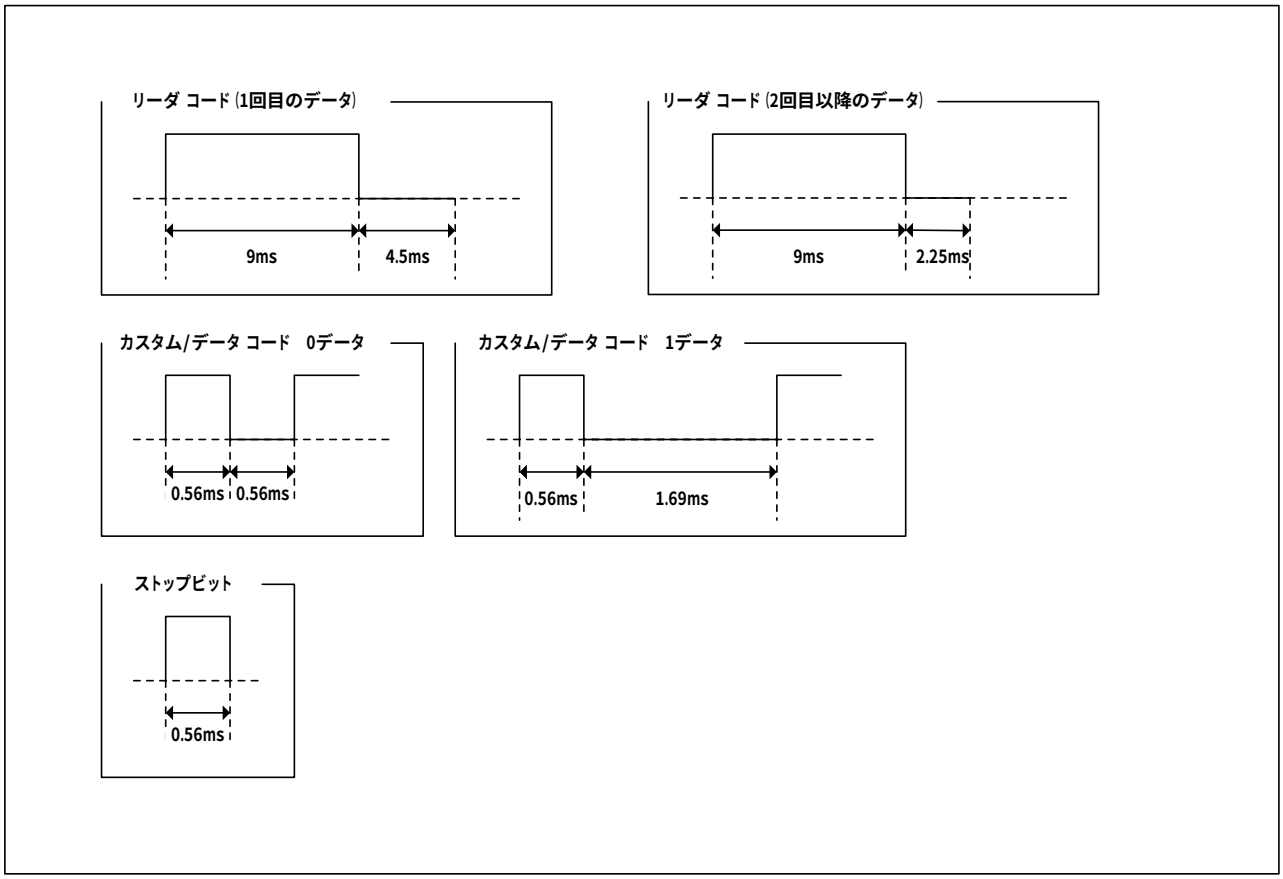

図6 各コード拡大

表 2 コード別認識範囲値

コード名	コード認識範囲値
リーダコード"H"	6.3ms ～ 11.69ms
リーダコード"L"	3.15ms ～ 5.84ms
カスタムコード"H"	0.393ms ～ 0.726ms
カスタムコード(0 データ)"L"	0.393ms ～ 0.726ms
カスタムコード(1 データ)"L"	1.183ms ～ 2.195ms
データコード"H"	0.393ms ～ 0.726ms
データコード(0 データ)"L"	0.393ms ～ 0.726ms
データコード(1 データ)"L"	1.183ms ～ 2.195ms
ストップビット	0.393ms ～ 0.726ms
フレームスペース	28.35ms ～ 52.64ms
リーダコード"H" (リピート)	6.3ms ～ 11.69ms
リーダコード"L" (リピート)	1.575ms ～ 2.923ms
ストップビット (リピート)	0.393ms ～ 0.726ms
フレームスペース (リピート)	67.33ms ～ 125.04ms

7. ソフトウェア説明

7.1 関数説明

表 3 に、使用する関数の説明を示します。

表 3 関数説明

関数名	ラベル名	機能
メイン処理	main	使用レジスタの設定、使用 RAM の初期化、割り込み許可、受信データ判定、タイマ判定
受信データ設定処理	rcv_data	受信データ割り込み検出時の受信データ設定処理
タイムオーバー設定処理	time_over	リーダコード検出から、1フレーム時間経過後の設定処理
パルス値設定処理	set_pulse_value	パルス値設定
受信データ判定処理	check_code	受信データ判定／設定
受信データ範囲チェック処理	cmp_pulse	受信データ範囲判定
反転データコード判定処理	judge_reversing_code	反転データコードの判定
反転データコードバッファ設定処理	set_reversing_code	反転データコードバッファ設定
反転データコード比較処理	cmp_reversing_code	反転データコード比較

7.2 レジスタ説明

表 4 に使用するレジスタの説明を示します。

表 4 レジスタ説明

レジスタ名		アドレス	設定値	機能
G1BCR0	ベースタイマ制御レジスタ 10	0122h	78h 7Bh	・クロック停止 ・ビット 15 のオーバフロー、62 分周、f1
G1BCR1	ベースタイマ制御レジスタ 11	0123h	00h 10h	・アップカウントモード ・ベースタイマリセット／カウント開始 ・INT0 端子、INT1 端子への"L"入力で ベースタイマリセットしない ・G1PO0 レジスタとベースタイマの一致で ベースタイマリセットしない
G1TMCR0	時間計測制御レジスタ 10	0118h	03h	・デジタルフィルタなし ・時間計測トリガ選択、両エッジ
G1FS	機能選択レジスタ 1	0127h	01h	チャネル 0 を時間計測機能
G1FE	機能許可レジスタ 1	0126h	01h	チャネル 0 の機能を動作
G1TM0	時間計測レジスタ 10	0100h	-	時間計測タイミングごとに ベースタイマ値が格納される
IIO3IC	割り込み制御レジスタ	0097h	00h	割り込み要求クリア
IIO3IR	割り込み要求レジスタ	00A3h	00h	インテリジェント I/O 時間計測機能 0 割り込み要求クリア
IIO3IE	割り込み許可レジスタ	00B3h	01h 05h	・インテリジェント I/O 時間計測機能 0 割り込み許可 ・割り込み要求を割り込みで使用 ※割り込み要求を割り込みで使用する場合、 0 ビット目を”1”にした後、他のビットを”1”にして ください
PS1	機能選択レジスタ A1	03B1h	00h	ポート P73 を入力ポート
PD7	ポートP7 方向レジスタ	03C3h	00h	ポート P73 を入力モード
TABSR	カウント開始フラグ	0340h	00h 03h	タイマ A0、A1 カウント停止／開始
TA0MR	タイマ A0 モードレジスタ	0356h	00h	・カウントソース f1 ・ゲート機能なし ・タイマモード
TA0	タイマ A0 レジスタ	0346h	7CFFh	1ms に設定
TA1MR	タイマ A1 モードレジスタ	0357h	01h	・リロードタイプ ・イベントカウントモード
TA1	タイマ A1 レジスタ	0348h	008Bh	140ms に設定（108ms の+30%誤差）
TRGSR	トリガ選択レジスタ	0343h	02h	タイマ A1 イベントトリガを TA0 のアンダフローに選択
TA0IC	割り込み制御レジスタ	006Ch	00h	優先レベル 0
TA1IC	割り込み制御レジスタ	008Ch	00h	優先レベル 0

7.3 RAM 説明

表 5 に使用する RAM の説明を示します。

表 5 RAM 説明

RAM 名	機能	データ長	使用関数
Rcv_mode	受信モード	1byte	main, rcv_data, time_over, check_code
Rcv_bit_cnt	受信カスタム／データコードビット数	1byte	main, rcv_data, check_code
Rcv_data_cnt	受信データ完了数	1byte	main, rcv_data, time_over, check_code
pulse[100]	受信データバッファ 16 進数で格納	200byte	time_over, set_pulse_value, check_code
pulse_cnt	受信データバッファ位置	1byte	main, rcv_data, time_over, set_pulse_value, check_code
code_low_cnt	受信データコード"L"カウンタ	1byte	main, rcv_data, judge_reversing_code
Rev_cnt	反転データカウンタ	1byte	main, rcv_data, judge_reversing_code, set_reversing_code, cmp_reversing_code
Rev_pulse[10]	反転データコード確認用バッファ 反転データではないコードの場合、以下に格納 0 データの場合…0xF1 1 データの場合…0xF0	10byte	set_reversing_code, cmp_reversing_code
old_tr	次回比較用の時間計測キャプチャ値	2byte	set_pulse_value

7.4 ROM 説明

表 6 に、使用する ROM の説明を示します。

表 6 ROM 説明

ROM 名	機能	データ長	使用関数
cmp_tbl[14][2]	受信コード比較テーブル ・[*][0]:各区間のフォーマット値、[*][1]:フォーマット値の±30%誤差範囲値 ・[*][0]-[*][1] ～ [*][0]+[*][1]:コード認識範囲値 ・フォーマット値より、±30%以内の誤差値の場合、認識 OK とする。 ・この範囲値は、周波数 32MHz の 62 分周として設定している。 [0][*] ： リーダコード"H"区間 (6.3ms ～ 11.69ms) [1][*] ： リーダコード"L"区間 (3.15ms ～ 5.84ms) [2][*] ： カスタムコード"H"区間 (0.393ms ～ 0.726ms) [3][*] ： カスタムコード(0 データ)"L"区間 (0.393ms ～ 0.726ms) [4][*] ： カスタムコード(1 データ)"L"区間 (1.183ms ～ 2.195ms) [5][*] ： データコード"H"区間 (0.393ms ～ 0.726ms) [6][*] ： データコード(0 データ)"L"区間 (0.393ms ～ 0.726ms) [7][*] ： データコード(1 データ)"L"区間 (1.183ms ～ 2.195ms) [8][*] ： ストップビット区間 (0.393ms ～ 0.726ms) [9][*] ： フレームスペース区間 (28.35ms ～ 52.64ms) [10][*] ： リーダコード"H"区間 (リピート) (6.3ms ～ 11.69ms) [11][*] ： リーダコード"L"区間 (リピート) (1.575ms ～ 2.923ms) [12][*] ： ストップビット区間 (リピート) (0.393ms ～ 0.726ms) [13][*] ： フレームスペース区間 (リピート) (67.33ms ～ 125.04ms)	56byte	check_code

8. 設定手順

本サンプルプログラムは、8MHz を 4 逓倍した 32MHz を CPU クロックにしています。

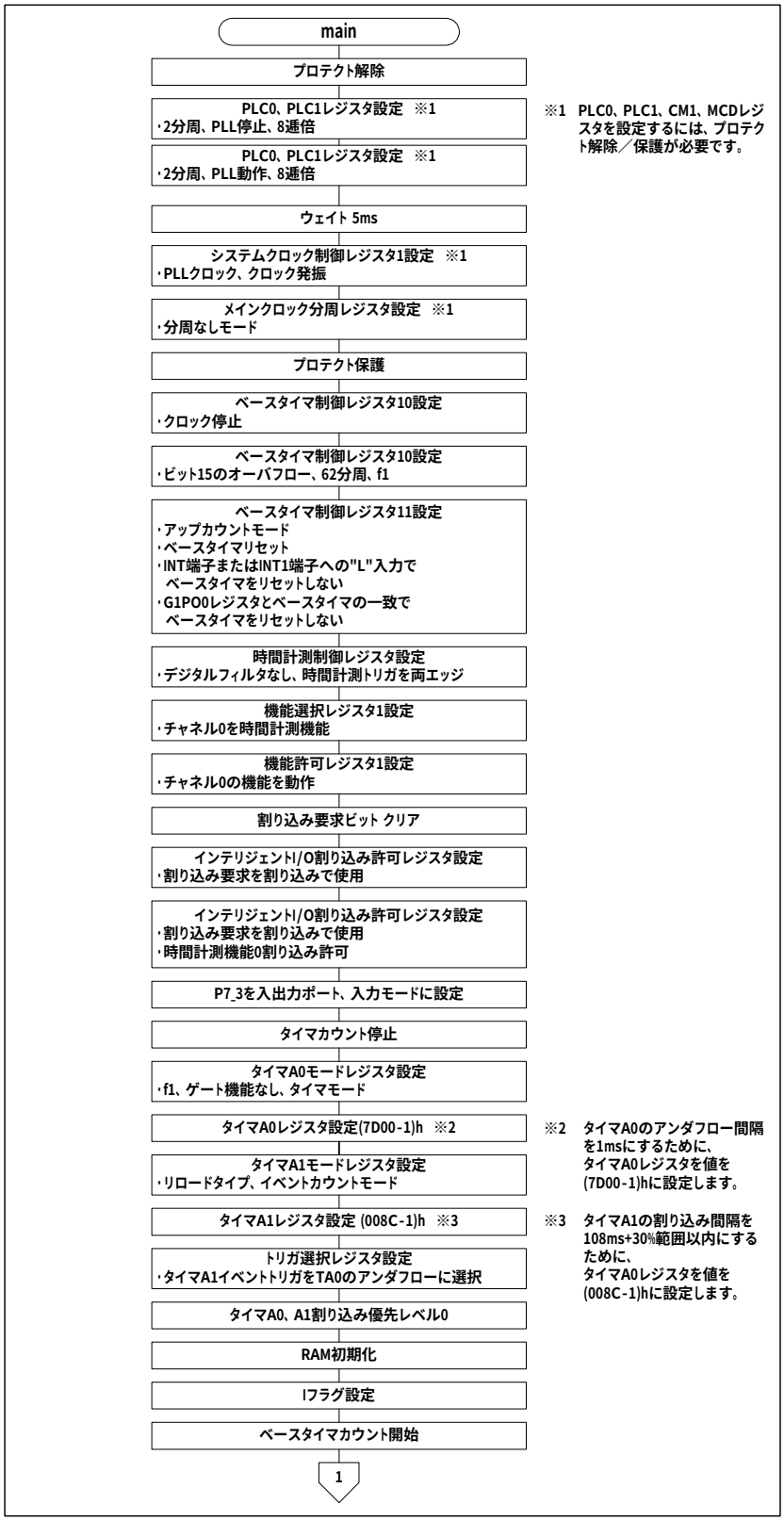

図7 フローチャート(main)

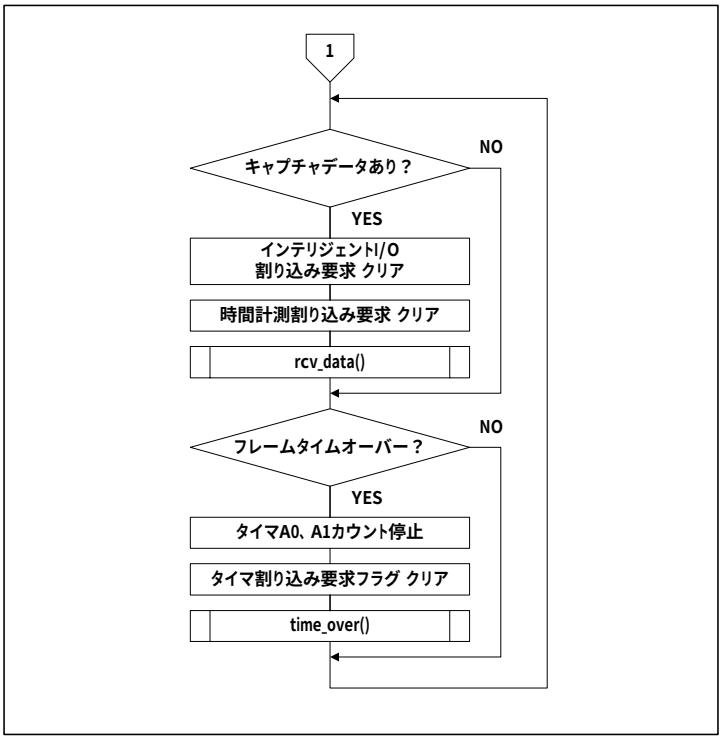

図8 フローチャート(main)

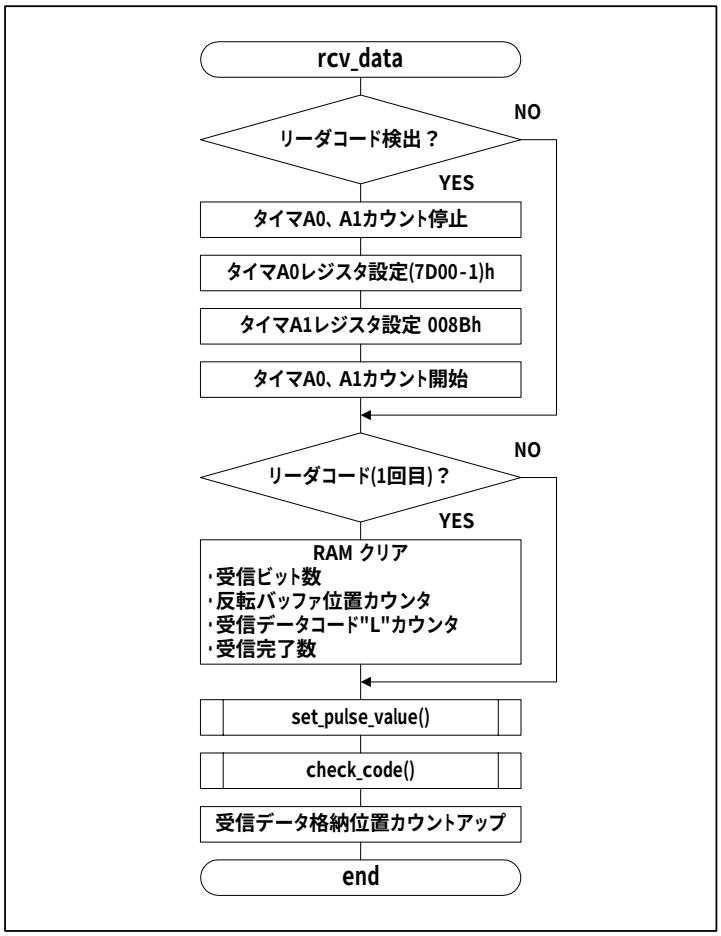

図9 フローチャート(rcv_data)

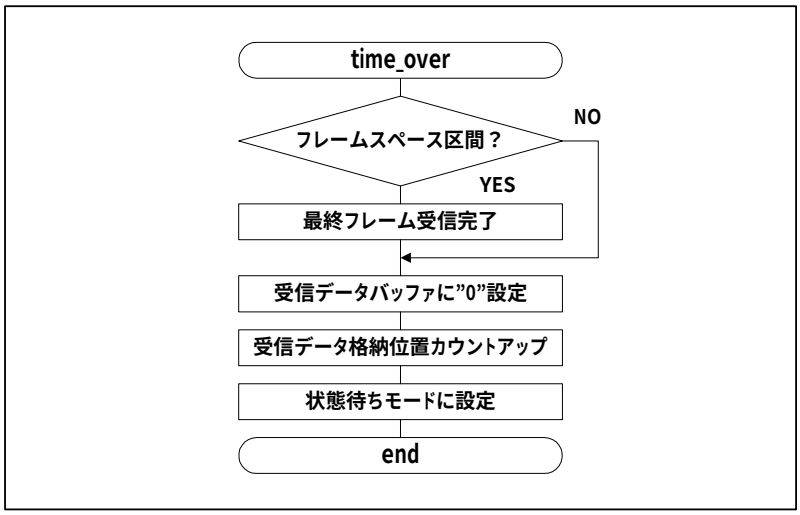

図10 フローチャート(time_over)

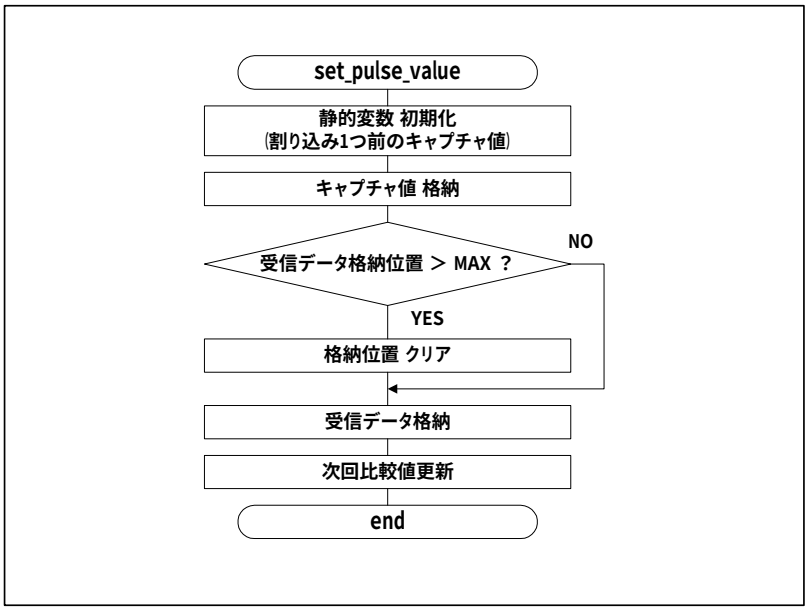

図11 フローチャート(set_pulse_value)

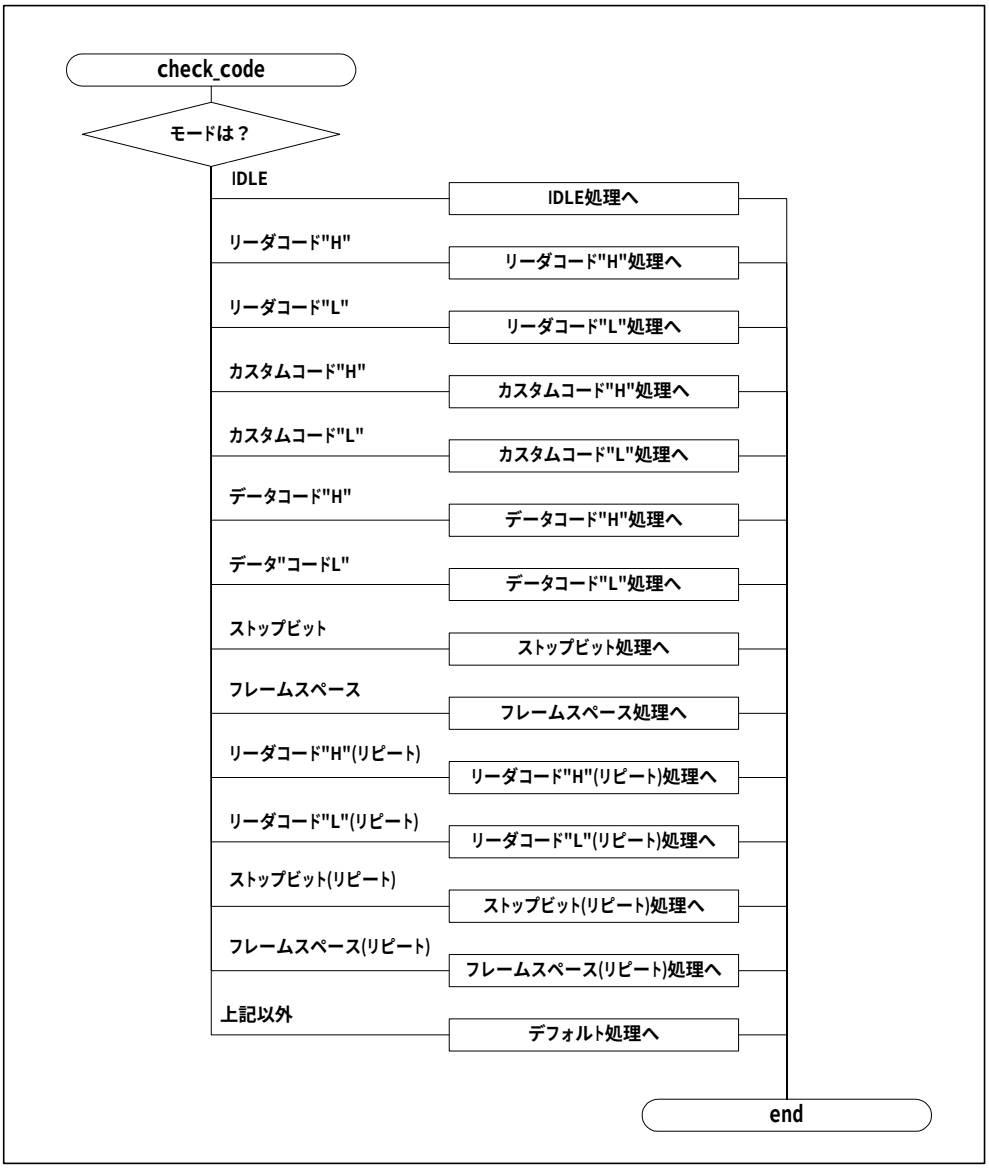

図12 フローチャート(check_code)

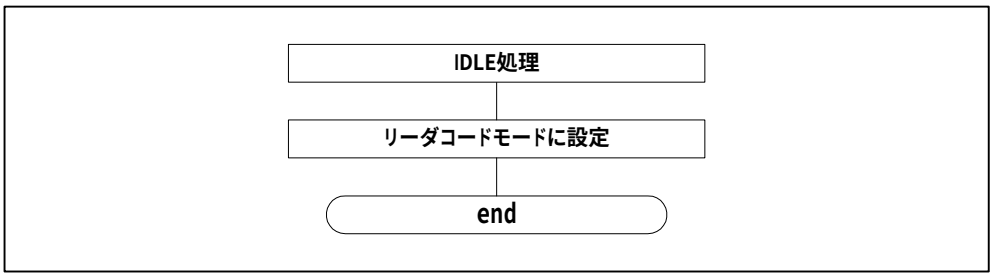

図13 フローチャート(check_code/IDLE)

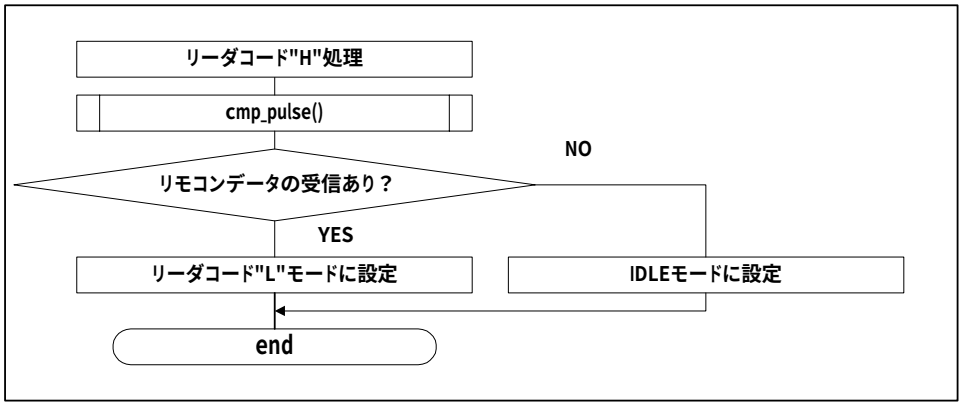

図14 フローチャート(check_code/リーダコード"H")

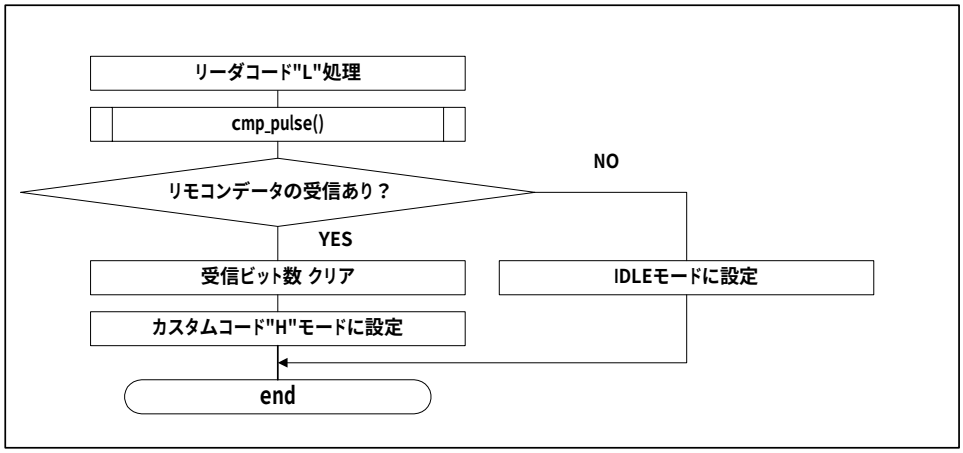

図15 フローチャート(check_code/リーダコード"L")

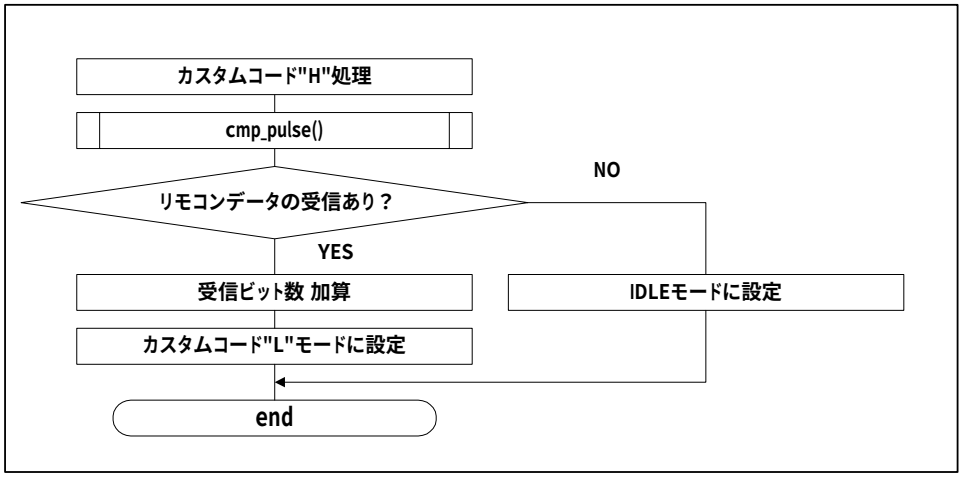

図16 フローチャート(check_code/カスタムコード"H")

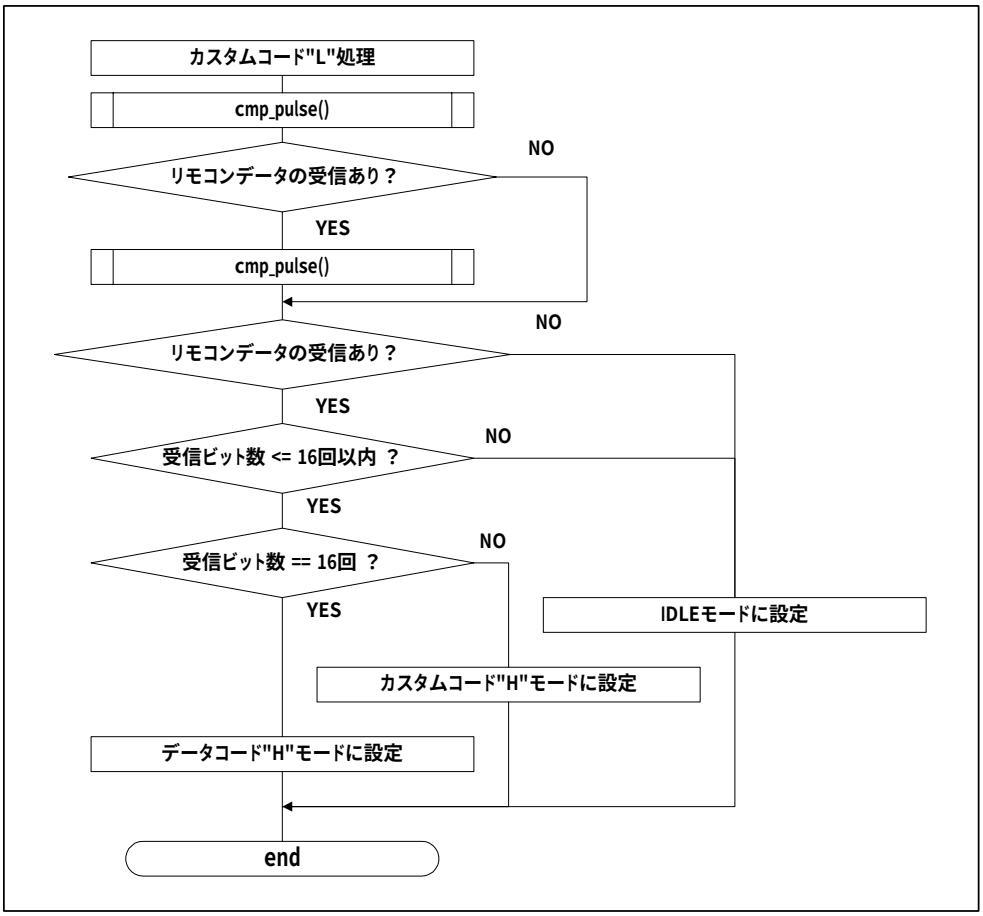

図17 フローチャート(check_code/カスタムコード"L")

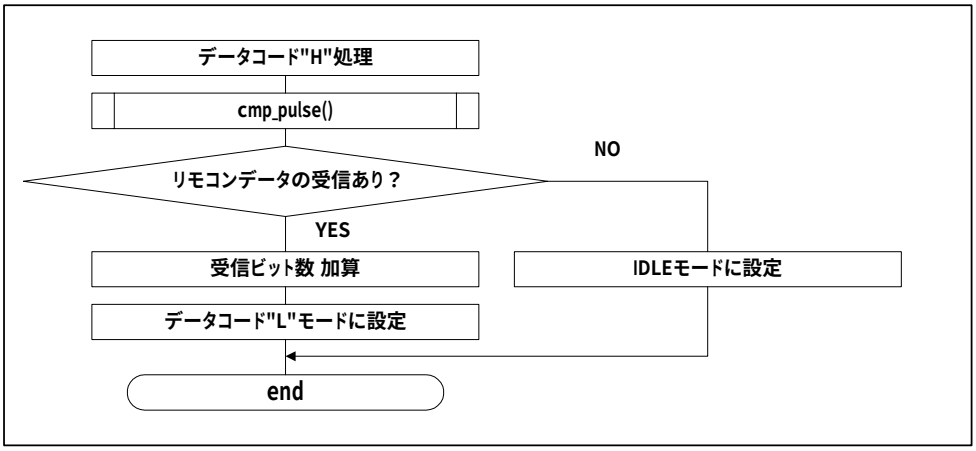

図18 フローチャート(check_code/データコード"H")

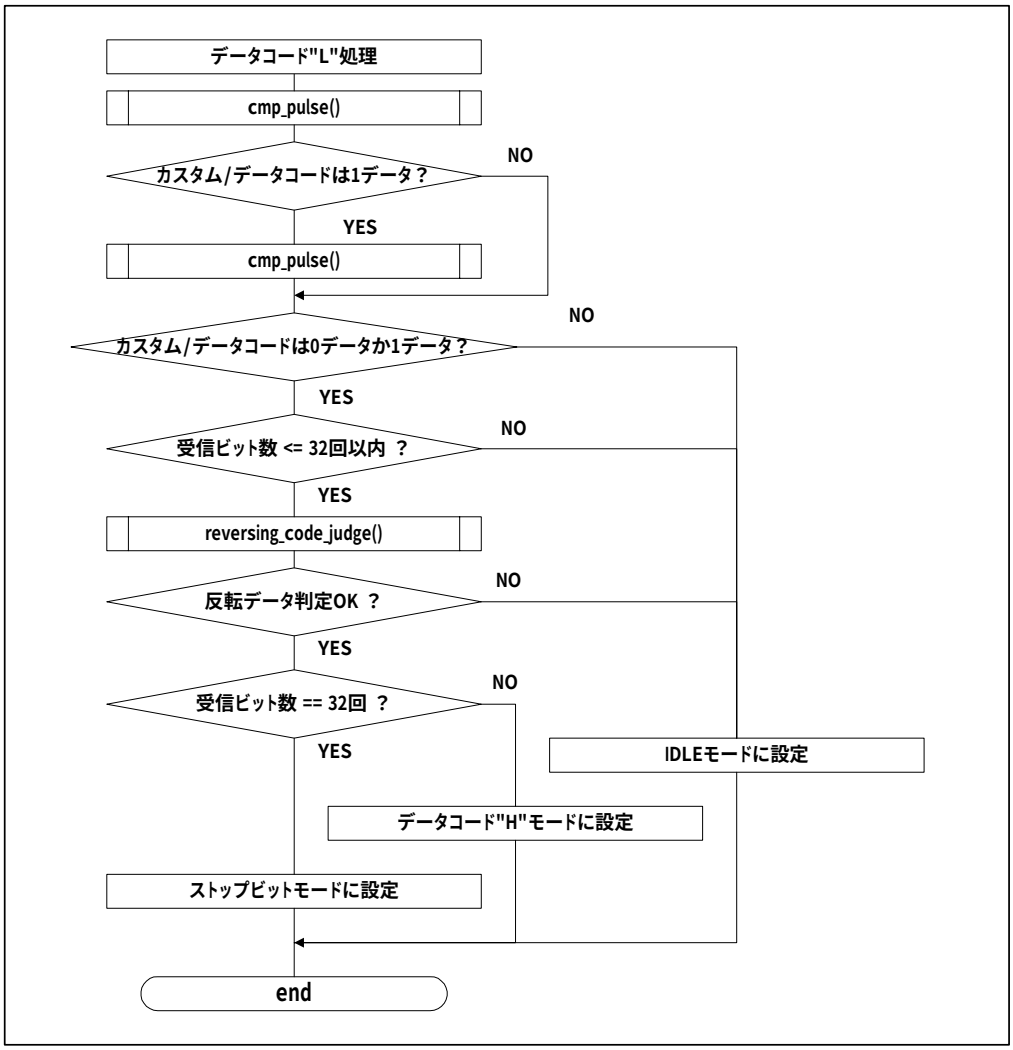

図19 フローチャート(check_code/データコード"L")

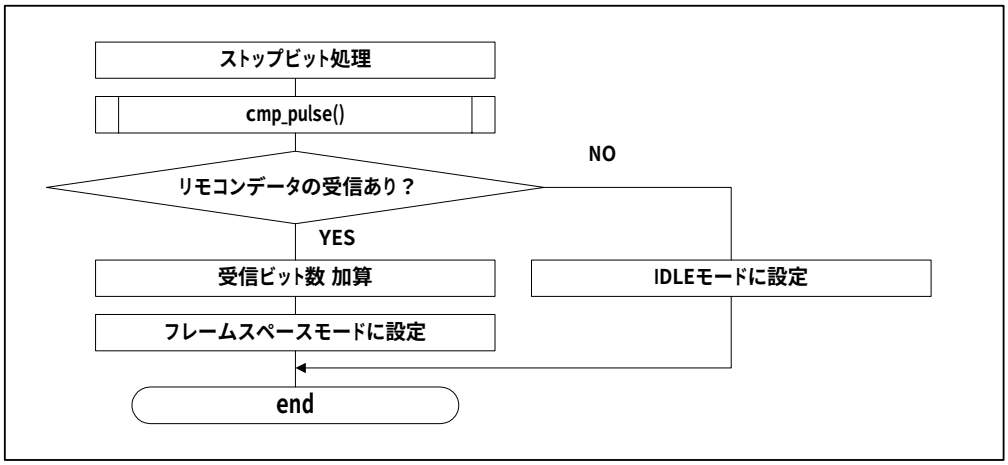

図20 フローチャート(check_code/ストップビット)

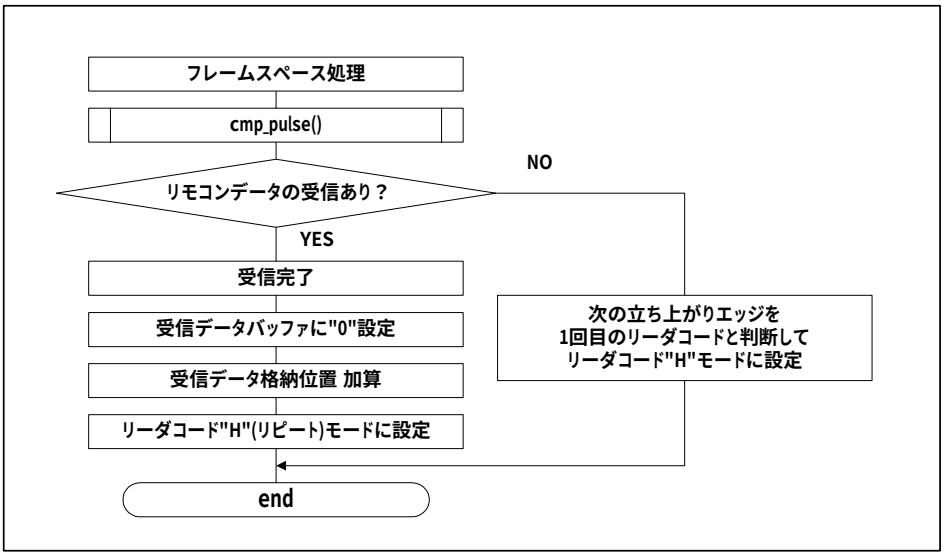

図21 フローチャート(check_code/フレームスペース)

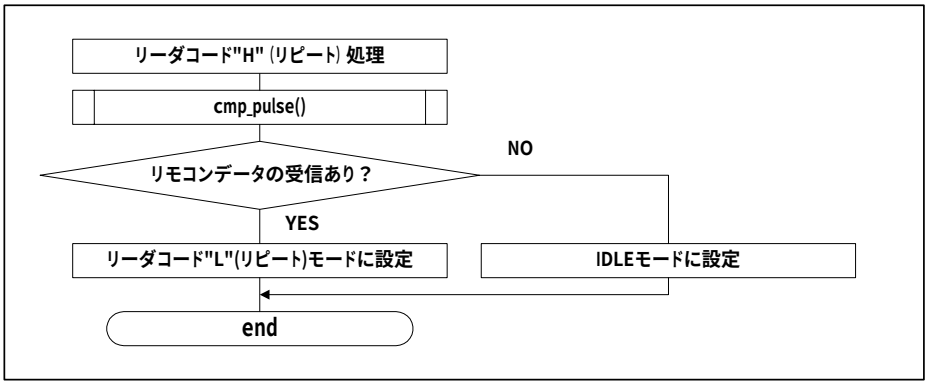

図22 フローチャート(check_code/リーダコード"H" (リピート))

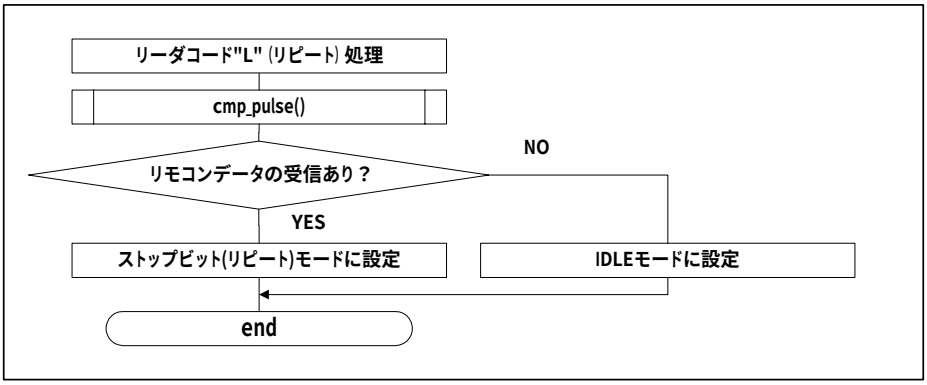

図23 フローチャート(check_code/リーダコード"L (リピート))

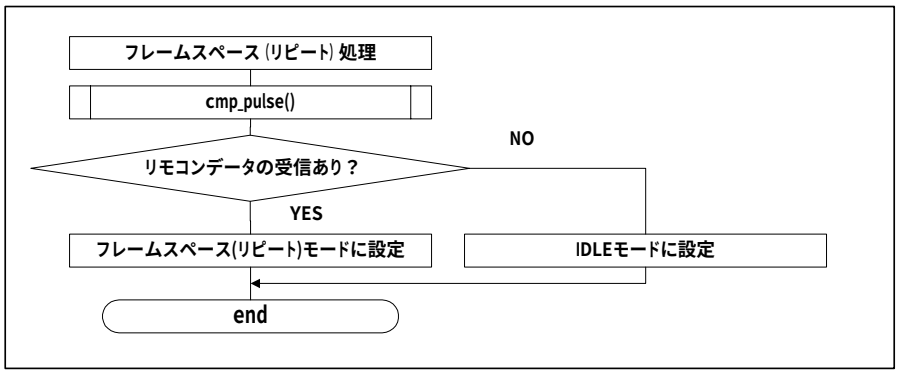

図24 フローチャート(check_code/ストップビット (リピート))

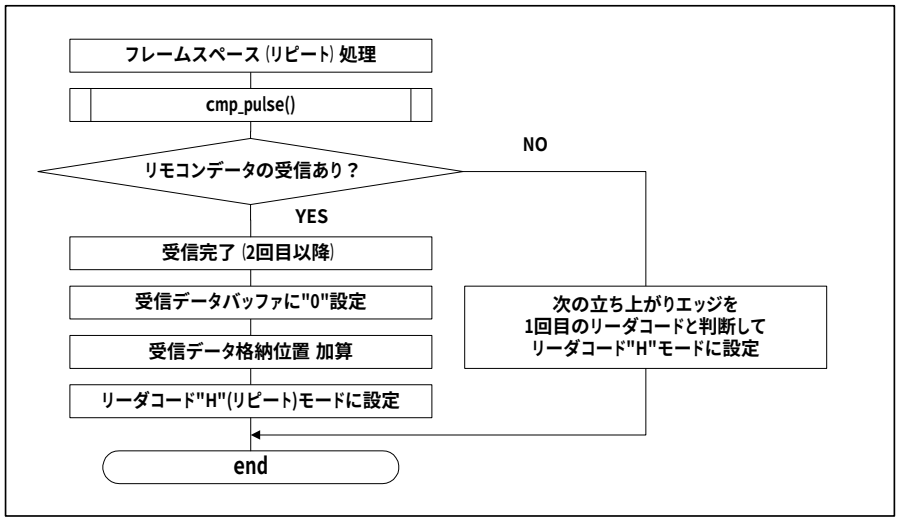

図25 フローチャート(check_code/フレームスペース (リピート))

図26 フローチャート(check_code/default)

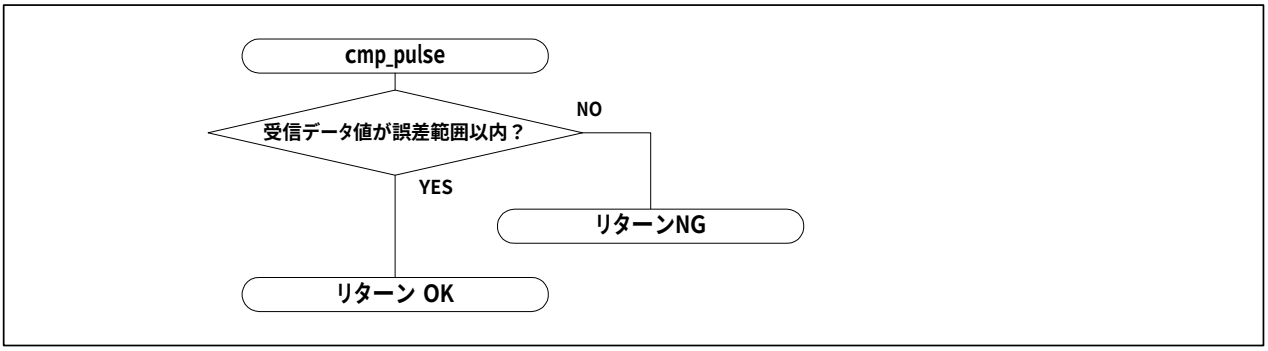

図27 フローチャート(cmp_pulse)

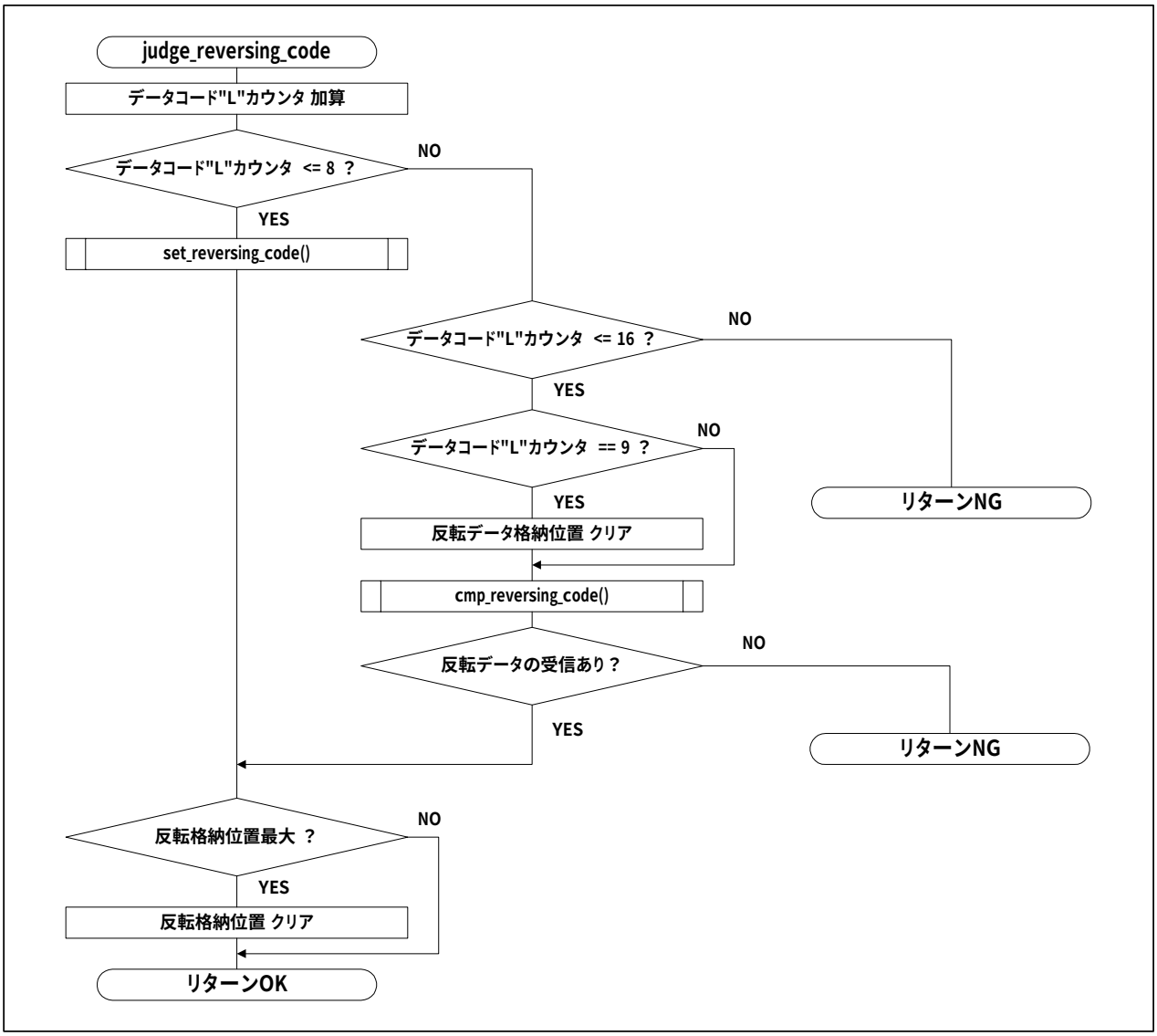

図28 フローチャート(judge_reversing_code)

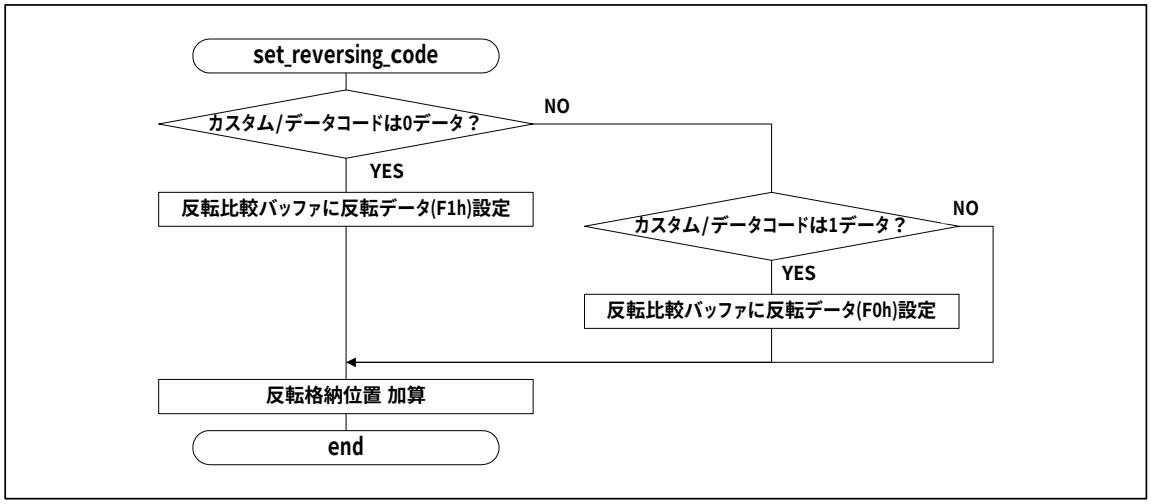

図29 フローチャート(set_reversing_code)

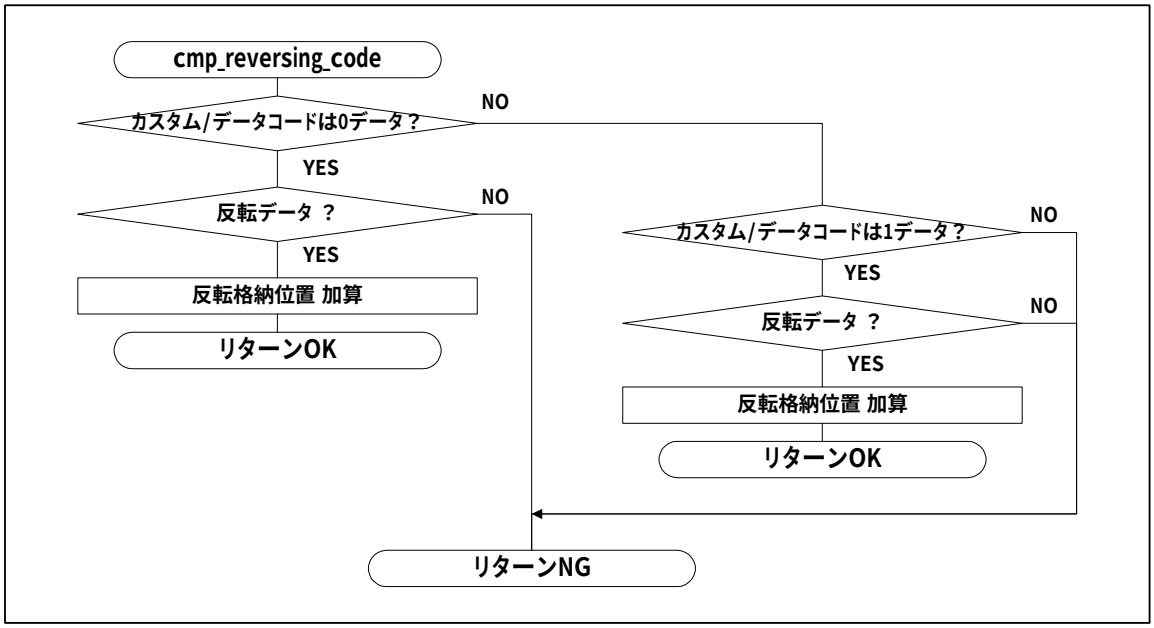

図30 フローチャート(cmp_reversing_code)

9. 参考サンプルプログラム例

本サンプルプログラムは、8MHz を 4 逓倍した 32MHz を CPU クロックにしています。

```
/***********************************************************************/
/*                                                                     */
/*      M32C/85 Program Collection                                     */
/*                                                                     */
/*      FILE NAME   :   rjj05b0859_src.c                               */
/*      CPU         :   M32C/85 Group                                  */
/*      FUNCTION    :   Remote control reception                       */
/*      HISTORY     :   2005.07.30  Ver 1.00                           */
/*                                                                     */
/*      Copyright(C)2005, Renesas Technology Corp.                     */
/*      Copyright(C)2005, Renesas Solutions Corp.                      */
/*      All rights reserved.                                           */
/*                                                                     */
/***********************************************************************/

/***********************************************************************/
/*      Include                                                        */
/***********************************************************************/
#include    "sfr32c85.h"

/***********************************************************************/
/*      Prototype declaration                                          */
/***********************************************************************/
void main(void);
void rcv_data(void);
void time_over(void);
void set_pulse_value(void);
void check_code(void);
unsigned char cmp_pulse(unsigned short, unsigned short, unsigned short);
unsigned char judge_reversing_code(unsigned char, unsigned char);
void set_reversing_code(unsigned char, unsigned char);
unsigned char cmp_reversing_code(unsigned char, unsigned char);

/***********************************************************************/
/*      DEFINE                                                         */
/***********************************************************************/
#define  OK                 0       /* OK */
#define  NG                 1       /* NG */

#define  IDLE               0       /* Receiving mode : IDLE                    */
#define  LEADER_CODE_H      1       /* Receiving mode : Leader code "H"         */
#define  LEADER_CODE_L      2       /* Receiving mode : Leader code "L"         */
```

```
#define  CUSTM_CODE_H        3       /* Receiving mode : Custm code "H"          */
#define  CUSTM_CODE_L        4       /* Receiving mode : Custm code "L"          */
#define  DATA_CODE_H         5       /* Receiving mode : Data code "H"           */
#define  DATA_CODE_L         6       /* Receiving mode : Data code "L"           */
#define  STOP_BIT            7       /* Receiving mode : Stop bit                */
#define  FRAMESPACE          8       /* Receiving mode : Frame space             */
#define  RE_LEADER_CODE_H    9       /* Receiving mode : Leader code "H"(repeat) */
#define  RE_LEADER_CODE_L    10      /* Receiving mode : Leader code "L"(repeat) */
#define  RE_STOP_BIT         11      /* Receiving mode : Stop bit       (repeat) */
#define  RE_FRAMESPACE       12      /* Receiving mode : Frame space    (repeat) */

#define  LEADER_CODE_H_POS      0    /* Range data table position : Leader code "H"          */
#define  LEADER_CODE_L_POS      1    /* Range data table position : Leader code "L"          */
#define  CUSTM_H_POS            2    /* Range data table position : Custm code "H"           */
#define  CUSTM_0_L_POS          3    /* Range data table position : Custm code "L"( 0 DATA ) */
#define  CUSTM_1_L_POS          4    /* Range data table position : Custm code "L"( 1 DATA ) */
#define  DATA_H_POS             5    /* Range data table position : Data code "H"           */
#define  DATA_0_L_POS           6    /* Range data table position : Data code "L"( 0 DATA )  */
#define  DATA_1_L_POS           7    /* Range data table position : Data code "L"( 1 DATA )  */
#define  STOP_BIT_POS           8    /* Range data table position : Stop bit                */
#define  FRAMESPACE_POS         9    /* Range data table position : Frame space             */
#define  RE_LEADER_CODE_H_POS   10   /* Range data table position : Leader code "H" (repeat) */
#define  RE_LEADER_CODE_L_POS   11   /* Range data table position : Leader code "L" (repeat) */
#define  RE_STOP_BIT_POS        12   /* Range data table position : Stop bit        (repeat) */
#define  RE_FRAMESPACE_POS      13   /* Range data table position : Frame space     (repeat) */

#define  PULSE_MAX              100     /* Base timer storage maximum position      */
#define  REV_PULSE_MAX          10      /* Reversing buffer storage maximum position */

#define  CUSTM_MAX_BIT_CNT      16      /* Number of custom code reception bits MAX   */
#define  DATA_MAX_8_BIT_CNT     8       /* Number of data code reception bits MAX     */
#define  DATA_MAX_LOW_BIT_CNT   16      /* Number of data code "L" maximum receptions */
#define  RCV_COMP_BIT_CNT       32      /* Number of reception completion bits        */

#define  PLL_WAIT_1MS           5       /* PLL wait time 5ms */
#define  PLL_WAIT_CNT           100     /* PLL wait time 1ms */

/*****************************************************************************/
/*      RAM                                                                  */
/*****************************************************************************/
unsigned char  rcv_mode;                /* Receiving mode                                 */
unsigned char  pulse_cnt;               /* Pulse storage position                         */
unsigned short pulse[PULSE_MAX];        /* Pulse storage buffer                           */
unsigned char  rcv_bit_cnt;             /* Number of reception bits                       */
unsigned char  rcv_data_cnt;            /* Number of receive data completion              */
unsigned char  rev_pulse[REV_PULSE_MAX]; /* Buffer for reversing data confirmation        */
unsigned char  rev_cnt;                 /* Buffer position for reversing data confirmation */
```

```
unsigned char  code_low_cnt;                /* Reception code "L" counter                        */

/************************************************************************************/
/*      ROM                                                                         */
/************************************************************************************/
const unsigned short cmp_tbl[14][2] =
                                            /* Receive data comparison table */
{

    /* When assuming that 62 dividing 32MHz (one cycle = 1.9375μs)  */
    /* [*][0]: Format value of each section                         */
    /* [*][1]: Error range value to format value(±30%)              */
    /* [*][0]-[*][1] to [*][0]+[*}[1] : Code recognition range value */

    {4645, 1393},           /* Range value of leader code "H"          ( 6.3ms to 11.69ms )    */
    {2322, 696},            /* Range value of leader code "L"          ( 3.15ms to 5.84ms )    */
    {289, 86},              /* Range value of custm code "H"           ( 0.393ms to 0.726ms )  */
    {289, 86},              /* Range value of custm code "L"( 0 DATA ) ( 0.393ms to 0.726ms )  */
    {872, 261},             /* Range value of custm code "L"( 1 DATA ) ( 1.183ms to 2.195ms )  */
    {289, 86},              /* Range value of data code "H"            ( 0.393ms to 0.726ms )  */
    {289, 86},              /* Range value of data code "L"( 0 DATA )  ( 0.393ms to 0.726ms )  */
    {872,261},              /* Range value of data code "L"( 1 DATA )  ( 1.183ms to 2.195ms )  */
    {289, 86},              /* Range value of stop bit                 ( 0.393ms to 0.726ms )  */
    {20903, 6270},          /* Range value of Frame space              ( 28.35ms to 52.64ms )  */
    {4645, 1393},           /* Range value of leader code "H"(repeat)  ( 6.3ms to 11.69ms )    */
    {1161, 348},            /* Range value of leader code "L"(repeat)  ( 1.575ms to 2.923ms )  */
    {289, 86},              /* Range value of stop bit       (repeat)  ( 0.393ms to 0.726ms )  */
    {49646, 14893}          /* Range value of Frame space    (repeat)  ( 67.33ms to 125.04ms ) */

};

/************************************************************************************/
/*      Pragma                                                                      */
/************************************************************************************/
/*""FUNC COMMENT""*****************************************************************
* ID            : 1.0
*-----------------------------------------------------------------------------------
* Include       : "sfr32c85.h"
*-----------------------------------------------------------------------------------
* Declaration   : void main(void)
*-----------------------------------------------------------------------------------
* Function      : main
*-----------------------------------------------------------------------------------
* Arguments     : Nothing
*-----------------------------------------------------------------------------------
* Returns       : Nothing
*-----------------------------------------------------------------------------------
```

```
* Input          : Nothing
* Ountput        : unsigned char rcv_mode     : Receiving mode
*                : unsigned char pulse_cnt    : Pulse storage position
*                : unsigned char rcv_data_cnt : Number of receive data completion
*                : unsigned char rcv_bit_cnt  : Number of reception bits
*                : unsigned char rev_cnt      : Buffer position for
*                :                              reversing data confirmation
*                : unsigned char code_low_cnt : Reception code "L" counter
*-------------------------------------------------------------------------------
* Call functions : rcv_data()
*                : time_over()
*-------------------------------------------------------------------------------
* Note           :
*-------------------------------------------------------------------------------
* History        : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""***************************************************************/
void main(void)
{

    unsigned char i;    /* PLL wait count */
    unsigned char j;    /* PLL wait count */


    /* PLL clock is made CPU source. 8MHz -> 32MHz */
    prcr = 0x01;        /* Protect Register */
    /* 00000001B */
    /*        +---------- Protect Bit 0                                    */
    /*                        Enables writing to CM0, CM1, CM2,           */
    /*                                          MCD, PLC0, PLC1 registers */
    /*                        0 : Write disable                           */
    /*                        1 : Write enable                            */

    plc = 0x0254;
    plc = 0x02D4;
    /* PLL Control Register 0 */
    /* 01010100B */
    /* |||| +++---------- Programmable counter select bit */
    /* ||||                   1 0 0 : Multiply-by-8      */
    /* |||+-------------- Set to "1"                     */
    /* ||+--------------- Set to "0"                     */
    /* |+---------------- Set to "1"                     */
    /* +----------------- Operation enable bit           */
    /*                        0 : PLL is Off             */
    /*                        1 : PLL is On              */
    /* PLL Control Register 1 */
    /* 00000010B */
    /* ||| |||+---------- Set to "0"                   */
```

```
/* ||| ||+----------- Set to "1"                    */
/* ||| |+------------ PLL clock division switch bit */
/* ||| |                 0 : Divide-by-2           */
/* ||| |                 1 : Divide-by-3           */
/* ||| +-------------- Set to "0"                    */
/* +++---------------- Set to "0"                    */


/* 5ms wait */
for(i=0;i<PLL_WAIT_1MS;i++){
    for(j=0;j<PLL_WAIT_CNT;j++){
    }
}

cm1 = 0xA0;         /* System Clock Control Register 1 */
/* 10100000B */
/* |||||||+---------- All clock stop control bit */
/* |||||||               0 : Clock oscillates   */
/* |||++++----------- Set to "0"                 */
/* ||+---------------- Set to "1"                 */
/* |+----------------- Set to "0"                 */
/* +------------------ CPU clock select bit 1     */
/*                       1 : PLL clock           */

mcd = 0x12;         /* Main Clock Division Register */
/* 00010010B */
/*    +++++---------- Main clock division select bit               */
/*                       1 0 0 1 0 : Divide-by-1(no division) mode */

prcr = 0x00;        /* Protect Register */
/* 00000000B */
/*        +---------- Protect Bit 0                                    */
/*                       Enables writing to CM0, CM1, CM2,            */
/*                                        MCD, PLC0, PLC1 registers */
/*                       0 : Write disable                            */
/*                       1 : Write enable                             */

g1bcr0 = 0x78;      /* Base Timer Control Register 10 */
/* 01111000B */
/* ||||||++---------- Count source select bit             */
/* ||||||               00 : Clock stops                  */
/* |+++++------------ Count source divide ratio select bit */
/* |                    11110 : Divide-by-62              */
/* +----------------- Base timer interrupt select bit     */
/*                       0 : Bit 15 overflows             */

g1bcr0 = 0x7B;      /* Base Timer Control Register 10 */
```

```
/* 01111011B */
/* ||||||++---------- Count source select bit             */
/* ||||||                11 : f1                          */
/* |+++++------------ Count source divide ratio select bit */
/* |                     11110 : Divide-by-62            */
/* +----------------- Base timer interrupt select bit     */
/*                       0 : Bit 15 overflows            */

g1bcr1 = 0x00;      /* Base Timer Control Register 11 */
/* 00000000B */
/*  ||| |+----------- Base timer reset cause select bit 1              */
/*  ||| |                0 : The base timer is not reset by           */
/*  ||| |                          matching with the G1PO0 register   */
/*  ||| +------------ Base timer reset cause select bit 2              */
/*  |||                  0 : The base timer is not reset by           */
/*  |||                            applying "L" to the INT0 or INT1 pin */
/*  ||+-------------- Base timer start bit                            */
/*  ||                   0 : Base timer is reset                      */
/*  ++---------------- Counter increment / Decrement control bit      */
/*                       00 : Counter increment mode                  */

g1tmcr0 = 0x03;     /* Time Measurement Control Register 10 */
/* 00000011B */
/* ||||||++---------- Time measurement trigger select bit            */
/* ||||||               11 : Both edges                            */
/* ||||++------------ Digital filter function select bit             */
/* ||||                 00 : No digital filter                      */
/* |||+-------------- Gate function select bit                       */
/* |||                  0 : Gate function is not used              */
/* ||+--------------- Gate function clear select bit                 */
/* ||                   0 : Not cleared                            */
/* |+---------------- Gate function clear bit                       */
/* |                    0 : The gate is cleared by                 */
/* |                              setting the GSC bit to "1"       */
/* |                      Set all bits 7 to 4 in the G1TMCR0 to    */
/* |                                        G1TMCR5 registers to "0". */
/* +----------------- Prescaler function select bit                */
/*                       0 : Not used                              */
/*                       Set all bits 7 to 4 in the G1TMCR0 to     */
/*                                        G1TMCR5 registers to "0". */

g1fs = 0x01;        /* Function Select Register 1 */
/* 00000001B */
/*        +---------- Channel 0 Time Measurement /                 */
/*                           Waveform Generating Function Select Bit */
/*                       1 : Selects the time measurement function  */
```

```
g1fe = 0x01;        /* Function Enable Register 1 */
/* 00000001B */
/*        +---------- Channel 0 Function Enable Bit          */
/*                        1 : Enables functions for channel 0 */

iio3ir = 0x00;      /* Interrupt Request Register */
/* 00000000B */

iio3ie = 0x01;      /* Interrupt Enable Register */
/* 00000001B */
/*        +---------- Interrupt request select bit */
/*                        1 : Interrupt request is used for interrupt */

iio3ie = 0x05;      /* Interrupt Enable Register */
/* 00000101B */
/*      | +---------- Interrupt request select bit                          */
/*      |                 1 : Interrupt request is used for interrupt       */
/*      +------------ Intelligent I/O Time Measurement 3 Interrupt Enabled   */
/*                        1 : Enables an interrupt by bit 2 in IIO0IR register */

ps1 = 0x00;         /* Function Select Register A1 */
/* 00000000B */
/* |||||||+---------- Port P70 output function select bit */
/* |||||||               0 : I/O port                    */
/* ||||||+----------- Port P71 output function select bit */
/* ||||||               0 : I/O port                    */
/* |||||+------------ Port P72 output function select bit */
/* |||||               0 : I/O port                    */
/* ||||+------------- Port P73 output function select bit */
/* ||||                0 : I/O port                    */
/* |||+-------------- Port P74 output function select bit */
/* |||                 0 : I/O port                    */
/* ||+--------------- Port P75 output function select bit */
/* ||                  0 : I/O port                    */
/* |+---------------- Port P76 output function select bit */
/* |                   0 : I/O port                    */
/* +----------------- Port P77 output function select bit */
/*                     0 : I/O port                    */

pd7 = 0x00;         /* Port P7 Direction Register */
/* 00000000B */
/* |||||||+---------- Port P70 direction bit                     */
/* |||||||               0 : Input mode (Functions as input port) */
/* ||||||+----------- Port P71 direction bit                     */
/* ||||||               0 : Input mode (Functions as input port) */
/* |||||+------------ Port P72 direction bit                     */
/* |||||               0 : Input mode (Functions as input port) */
```

```
/* ||||+------------- Port P73 direction bit                        */
/* ||||                 0 : Input mode (Functions as input port) */
/* |||+-------------- Port P74 direction bit                        */
/* |||                  0 : Input mode (Functions as input port) */
/* ||+--------------- Port P75 direction bit                        */
/* ||                   0 : Input mode (Functions as input port) */
/* |+---------------- Port P76 direction bit                        */
/* |                    0 : Input mode (Functions as input port) */
/* +----------------- Port P77 direction bit                        */
/*                      0 : Input mode (Functions as input port) */

tabsr = 0x00;       /* Count Start Flag */
/* 00000000B */
/*       |+---------- Timer A0 count start flag */
/*       |               0 : Stops counting    */
/*       +----------- Timer A1 count start flag */
/*                       0 : Stops counting    */

ta0mr = 0x00;           /* Timer A0 Mode Register */
/* 00000000B */
/* ||||||++---------- Operating mode select bit            */
/* ||||||               0 0 : Timer mode                 */
/* |||||+------------ Set to "0"                          */
/* |||++------------- Gate function select bit            */
/* |||                  0 X : Gate function disabled      */
/* |||                (TAiIN pin is a programmable I/O pin) */
/* ||+--------------- Set to "0" in timer mode            */
/* ++----------------    0 0 : f1                         */

ta0 = 0x7CFF;           /* Timer A0 Register (7D00-1) */

ta1mr = 0x01;           /* Timer A1 Mode Register */
/* 00000001B */
/* ||||||++---------- Operating mode select bit                    */
/* ||||||               0 1 : Event counter mode                 */
/* |||||+------------ Set to "0"                                  */
/* ||||+------------- Count polarity select bit                   */
/* ||||                 0 : Counts falling edges of               */
/* ||||                                an external signal         */
/* |||+-------------- Increment/Decrement switching source select bit */
/* |||                  0 : UDF registser setting                 */
/* ||+--------------- Set to "0" in event counter mode            */
/* |+---------------- Count operation type select bit             */
/* |                    0 : Reloading                             */
/* +----------------- Set to "0"                                  */

ta1 = 0x008B;           /* Timer A1 Register (8C-1)*/
```

```
trgsr = 0x02;           /* Trigger Select Register */
/* 00000010B */
/* ||||||++---------- Timer A1 event/trigger select bit         */
/* ||||||               1 0 : Selects the TA0 overflows        */
/* ||||++------------ Timer A2 event/trigger select bit         */
/* ||||                 0 0 : Selects an input to the TA2IN pin */
/* ||++-------------- Timer A3 event/trigger select bit         */
/* ||                   0 0 : Selects an input to the TA3IN pin */
/* ++---------------- Timer A4 event/trigger select bit         */
/*                      0 0 : Selects an input to the TA4IN pin */

ta0ic = 0x00;           /* Interrupt Control Register */
/* 00000000B */
/*     |+++---------- Interrupt priority level select bit     */
/*     |               0 0 0 : Level 0 (interrupt disabled) */
/*     +------------- Interrupt request bit                   */
/*                      0 : No interrupt requested           */

ta1ic = 0x00;           /* Interrupt Control Register */
/* 00000000B */
/*     |+++---------- Interrupt priority level select bit     */
/*     |               0 0 0 : Level 0 (interrupt disabled) */
/*     +------------- Interrupt request bit                   */
/*                      0 : No interrupt requested           */

/* RAM Initial */
rcv_mode = IDLE;
pulse_cnt = 0;
rcv_data_cnt = 0;
rcv_bit_cnt = 0;
rev_cnt = 0;
code_low_cnt = 0;

asm("FSET I");      /* "I" flag */

bts_g1bcr1 = 1;     /* Base timer start of counting */

while(1){
                    /* Data reception waiting  */
    if(tm10r == 1){
        /* When there is an external data reception */
        iio3ic = 0x00;      /* Interrupt request bit clear      */
        iio3ir = 0x00;      /* Interrupt Request Register clear */
        rcv_data();         /* rcv_data function                */

    }
```

```
        if(ir_talic == 1){
            /* When time passes by one frame */
            tabsr = 0x00;       /* timer A0, A1 stop           */
            ir_talic = 0;       /* Interrupt request flag clear */
            time_over();        /* time_over function          */

        }
    }
}


/*""FUNC COMMENT""*****************************************************************
* ID              : 1.1
*-----------------------------------------------------------------------------------
* Include         : "sfr32c85.h"
*-----------------------------------------------------------------------------------
* Declaration     : void rcv_data(void)
*-----------------------------------------------------------------------------------
* Function        : Receive data setting
*                 : when receive data interrupt is detected
*-----------------------------------------------------------------------------------
* Arguments       : Nothing
*-----------------------------------------------------------------------------------
* Returns         : Nothing
*-----------------------------------------------------------------------------------
* Input           : unsigned char rcv_mode     : Receiving mode
* Ountput         : unsigned char pulse_cnt    : Pulse storage position
*                 : unsigned char rcv_bit_cnt  : Number of reception bits
*                 : unsigned char rev_cnt      : Buffer position for
*                 :                              reversing data confirmation
*                 : unsigned char code_low_cnt : Reception code "L" counter
*-----------------------------------------------------------------------------------
* Call functions  : set_pulse_value()
*                 : check_code()
*-----------------------------------------------------------------------------------
* Note            :
*-----------------------------------------------------------------------------------
* History         : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
void rcv_data(void)
{

    if( (rcv_mode == IDLE) || (rcv_mode == FRAMESPACE) || (rcv_mode == RE_FRAMESPACE)){
        /* When Leader code "H" detects */
        tabsr = 0x00;       /* timer A0, A1 stop       */
        ta0 = 0x7CFF;       /* 1ms   (7D00-1)h         */
        ta1 = 0x008B;       /* 140ms (Underflow timing) */
```

```
        tabsr = 0x03;       /* timer A0, A1 start       */
    }

    if(rcv_mode == LEADER_CODE_H){
        /* When Leader code "H" (Data of the first time) */
        rcv_bit_cnt = 0;    /* Number of reception bits clear                          */
        rev_cnt = 0;        /* Buffer position for reversing data confirmation clear */
        code_low_cnt = 0;   /* Reception code "L" counter clear                        */
        rcv_data_cnt = 0;   /* Number of receive data completion clear                 */
    }

    set_pulse_value();      /* set_pulse_value function */
    check_code();           /* check_code function      */

    pulse_cnt++;            /* Pulse storage position add */

}


/*""FUNC COMMENT""*****************************************************************
* ID            : 1.2
*--------------------------------------------------------------------------------
* Include       : Nothing
*--------------------------------------------------------------------------------
* Declaration   : void time_over(void)
*--------------------------------------------------------------------------------
* Function      : Processing after it passes of one frame
*               :
*--------------------------------------------------------------------------------
* Arguments     : Nothing
*--------------------------------------------------------------------------------
* Returns       : Nothing
*--------------------------------------------------------------------------------
* Input         : unsigned char rcv_mode     : Receiving mode
* Ountput       : unsigned char rcv_data_cnt : Number of receive data completion
*               : unsigned short pulse[]     : Pulse storage buffer
*               : unsigned char pulse_cnt    : Pulse storage position
*               : unsigned char rcv_mode     : Receiving mode
*--------------------------------------------------------------------------------
* Call functions : Nothing
*--------------------------------------------------------------------------------
* Note          :
*--------------------------------------------------------------------------------
* History       : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
void time_over(void)
{
```

```
    if( (rcv_mode == FRAMESPACE) || (rcv_mode == RE_FRAMESPACE) ){
        /* When the frame space passes the fixed time */
        rcv_data_cnt++;         /* Number of receive data completion add */

    }

    pulse[pulse_cnt] = 0;       /* Stopper setting */
    pulse_cnt++;                /* Pulse storage position add  */

    rcv_mode = IDLE;            /* mode clear */

}


/*""FUNC COMMENT""*****************************************************************
* ID             : 1.3
*--------------------------------------------------------------------------------
* Include        : "sfr32c85.h"
*--------------------------------------------------------------------------------
* Declaration    : void set_pulse_value(void)
*--------------------------------------------------------------------------------
* Function       : The pulse value is set
*                :
*--------------------------------------------------------------------------------
* Arguments      : Nothing
*--------------------------------------------------------------------------------
* Returns        : Nothing
*--------------------------------------------------------------------------------
* Input          : unsigned char pulse_cnt : Pulse storage position
* Ountput        : unsigned short pulse[]  : Pulse storage buffer
*                : unsigned char pulse_cnt : Pulse storage position
*--------------------------------------------------------------------------------
* Call functions : Nothing
*--------------------------------------------------------------------------------
* Note           :
*--------------------------------------------------------------------------------
* History        : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
void set_pulse_value(void)
{

    unsigned short now_tr;              /* Present capture value */


    static unsigned short old_tr = 0;   /* The previous capture value of one */
```

```
    now_tr = (unsigned short)g1tm0;     /* capture value storage */

    if(pulse_cnt >= PULSE_MAX){
        /* When the storage position is larger than the MAX value */
        pulse_cnt = 0;                  /* storage position clear */
    }

    pulse[pulse_cnt] = now_tr - old_tr; /* pulse value storage */

    old_tr = now_tr;                    /* The present capture value is set for  */
                                        /* the previous capture value of one     */

}


/*""FUNC COMMENT""*****************************************************************
* ID             : 1.4
*--------------------------------------------------------------------------------
* Include        : Nothing
*--------------------------------------------------------------------------------
* Declaration    : void check_code(void)
*--------------------------------------------------------------------------------
* Function       : Data code check function according to mode
*                :
*--------------------------------------------------------------------------------
* Arguments      : Nothing
*--------------------------------------------------------------------------------
* Returns        : Nothing
*--------------------------------------------------------------------------------
* Input          : unsigned char rcv_mode     : Receiving mode
*                : unsigned short pulse[]     : Pulse storage buffer
*                : unsigned char pulse_cnt    : Pulse storage position
*                : unsigned char rcv_bit_cnt  : Number of reception bits
* Ountput        : unsigned char rcv_mode     : Receiving mode
*                : unsigned char rcv_bit_cnt  : Number of reception bits
*                : unsigned short pulse[]     : Pulse storage buffer
*                : unsigned char pulse_cnt    : Pulse storage position
*                : unsigned char rcv_data_cnt : Number of receive data completion
*--------------------------------------------------------------------------------
* Call functions : cmp_pulse()
*--------------------------------------------------------------------------------
* Note           :
*--------------------------------------------------------------------------------
* History        : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
void check_code(void)
{
```

```
unsigned char rtn;      /* Return value */
unsigned char rtn_0;    /* Return value for data code */
unsigned char rtn_1;    /* Return value for data code */


switch(rcv_mode){
    case IDLE:              /* When mode is IDLE */
        rcv_mode = LEADER_CODE_H;     /* Mode transition */
        break;

    case LEADER_CODE_H:       /* When mode is LEADER_CODE_H */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[LEADER_CODE_H_POS][0] + cmp_tbl[LEADER_CODE_H_POS][1],
                            cmp_tbl[LEADER_CODE_H_POS][0] - cmp_tbl[LEADER_CODE_H_POS][1]);
        if(rtn == OK){
            /* When the range of LEADER_CODE_H */
            rcv_mode = LEADER_CODE_L;   /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;

    case LEADER_CODE_L:       /* When mode is LEADER_CODE_L */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[LEADER_CODE_L_POS][0] + cmp_tbl[LEADER_CODE_L_POS][1],
                            cmp_tbl[LEADER_CODE_L_POS][0] - cmp_tbl[LEADER_CODE_L_POS][1]);
        if(rtn == OK){
            /* When the range of LEADER_CODE_L */
            rcv_bit_cnt = 0;                /* Number of reception bits clear */
            rcv_mode = CUSTM_CODE_H;        /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                    /* When the receive data error, mode clear */
        break;

    case CUSTM_CODE_H:      /* When mode is CUSTM_CODE_H */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[CUSTM_H_POS][0] + cmp_tbl[CUSTM_H_POS][1],
                            cmp_tbl[CUSTM_H_POS][0] - cmp_tbl[CUSTM_H_POS][1]);
        if(rtn == OK){
            /* When the range of CUSTM_CODE_H */
            rcv_bit_cnt++;                  /* Number of reception bits add */
            rcv_mode = CUSTM_CODE_L;        /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                    /* When the receive data error, mode clear */
```

```
        break;

    case CUSTM_CODE_L:      /* When mode is CUSTM_CODE_L */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[CUSTM_0_L_POS][0] + cmp_tbl[CUSTM_0_L_POS][1],
                            cmp_tbl[CUSTM_0_L_POS][0] - cmp_tbl[CUSTM_0_L_POS][1]);
        if(rtn != OK){
            /* When data is not 0 */
            rtn = cmp_pulse(pulse[pulse_cnt],
                            cmp_tbl[CUSTM_1_L_POS][0] + cmp_tbl[CUSTM_1_L_POS][1],
                                cmp_tbl[CUSTM_1_L_POS][0] - cmp_tbl[CUSTM_1_L_POS][1]);
        }

        if(rtn == OK){
            /* When the range of CUSTM_CODE_L( "0" data or "1" data ) */
            if(rcv_bit_cnt <= CUSTM_MAX_BIT_CNT){
                /* When The number of reception bits is within 16 times */
                if(rcv_bit_cnt == CUSTM_MAX_BIT_CNT){
                    /* When The number of reception bits is 16 times */
                    rcv_mode = DATA_CODE_H;     /* Mode transition */
                    break;
                }
                rcv_mode = CUSTM_CODE_H;        /* Mode transition */
                break;
            }
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;

    case DATA_CODE_H:       /* When mode is DATA_CODE_H */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[DATA_H_POS][0] + cmp_tbl[DATA_H_POS][1],
                            cmp_tbl[DATA_H_POS][0] - cmp_tbl[DATA_H_POS][1]);
        if(rtn == OK){
            /* When the range of DATA_CODE_H */
            rcv_bit_cnt++;                  /* Number of reception bits add */
            rcv_mode = DATA_CODE_L;         /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                    /* When the receive data error, mode clear */
        break;

    case DATA_CODE_L:       /* When mode is DATA_CODE_L */

        rtn_0 = NG;         /* Initialization */
        rtn_1 = NG;         /* Initialization */
```

```
        rtn_0 = cmp_pulse(pulse[pulse_cnt],
                          cmp_tbl[DATA_0_L_POS][0] + cmp_tbl[DATA_0_L_POS][1],
                              cmp_tbl[DATA_0_L_POS][0] - cmp_tbl[DATA_0_L_POS][1]);
        if(rtn_0 != OK){
            /* When data is not 0 */
            rtn_1 = cmp_pulse(pulse[pulse_cnt],
                              cmp_tbl[DATA_1_L_POS][0] + cmp_tbl[DATA_1_L_POS][1],
                                  cmp_tbl[DATA_1_L_POS][0] - cmp_tbl[DATA_1_L_POS][1]);
        }

        if( (rtn_0 == OK) || (rtn_1 == OK) ){
            /* When the range of DATA_CODE_L( "0" data or "1" data ) */
            if(rcv_bit_cnt <= RCV_COMP_BIT_CNT){

                /* When The number of reception bits is within 32 times */
                rtn = judge_reversing_code(rtn_0, rtn_1);   /* judge_reversing_code function */
                if(rtn == OK){
                    /* When reversing data judgment OK */
                    if(rcv_bit_cnt == RCV_COMP_BIT_CNT){
                        rcv_mode = STOP_BIT;        /* Mode transition */
                        break;
                    }
                    rcv_mode = DATA_CODE_H;         /* Mode transition */
                    break;
                }
            }
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;

    case STOP_BIT:          /* When mode is STOP_BIT */
        rtn = cmp_pulse(pulse[pulse_cnt],
                          cmp_tbl[STOP_BIT_POS][0] + cmp_tbl[STOP_BIT_POS][1],
                              cmp_tbl[STOP_BIT_POS][0] - cmp_tbl[STOP_BIT_POS][1]);
        if(rtn == OK){
            /* When the range of STOP_BIT */
            rcv_bit_cnt++;              /* Number of reception bits add */
            rcv_mode = FRAMESPACE;          /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;

    case FRAMESPACE:                /* When mode is FRAMESPACE */
        rtn = cmp_pulse(pulse[pulse_cnt],
                          cmp_tbl[FRAMESPACE_POS][0] + cmp_tbl[FRAMESPACE_POS][1],
                              cmp_tbl[FRAMESPACE_POS][0] - cmp_tbl[FRAMESPACE_POS][1]);
```

```
        if(rtn == OK){
            /* When the range of FRAMESPACE */
            /* Reception completion(The first data) */
            rcv_data_cnt++;                 /* Number of reception completion add */
            pulse[pulse_cnt] = 0;           /* Stopper setting */
            pulse_cnt++;                    /* Pulse storage position add */
            rcv_mode = RE_LEADER_CODE_H;    /* Mode transition */
            break;
        }
        rcv_mode = LEADER_CODE_H;           /* When the receive data error, mode clear */
        break;

    case RE_LEADER_CODE_H:    /* When mode is RE_LEADER_CODE_H */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[RE_LEADER_CODE_H_POS][0] + cmp_tbl[RE_LEADER_CODE_H_POS][1],
                         cmp_tbl[RE_LEADER_CODE_H_POS][0] - cmp_tbl[RE_LEADER_CODE_H_POS][1]);
        if(rtn == OK){
            /* When the range of RE_LEADER_CODE_H */
            rcv_mode = RE_LEADER_CODE_L;       /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                       /* When the receive data error, mode clear */
        break;

    case RE_LEADER_CODE_L:    /* When mode is RE_LEADER_CODE_L */
        rtn = cmp_pulse(pulse[pulse_cnt],
                        cmp_tbl[RE_LEADER_CODE_L_POS][0] + cmp_tbl[RE_LEADER_CODE_L_POS][1],
                         cmp_tbl[RE_LEADER_CODE_L_POS][0] - cmp_tbl[RE_LEADER_CODE_L_POS][1]);
        if(rtn == OK){
            /* When the range of RE_LEADER_CODE_L */
            rcv_mode = RE_STOP_BIT;     /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;

    case RE_STOP_BIT:       /* When mode is RE_STOP_BIT */
        rtn = cmp_pulse(pulse[pulse_cnt],
                          cmp_tbl[RE_STOP_BIT_POS][0] + cmp_tbl[RE_STOP_BIT_POS][1],
                              cmp_tbl[RE_STOP_BIT_POS][0] - cmp_tbl[RE_STOP_BIT_POS][1]);
        if(rtn == OK){
            /* When the range of RE_STOP_BIT */
            rcv_mode = RE_FRAMESPACE;   /* Mode transition */
            break;
        }
        rcv_mode = IDLE;                /* When the receive data error, mode clear */
        break;
```

```
        case RE_FRAMESPACE:       /* When mode is RE_FRAMESPACE */
            rtn = cmp_pulse(pulse[pulse_cnt],
                            cmp_tbl[RE_FRAMESPACE_POS][0] + cmp_tbl[RE_FRAMESPACE_POS][1],
                               cmp_tbl[RE_FRAMESPACE_POS][0] - cmp_tbl[RE_FRAMESPACE_POS][1]);
            if(rtn == OK){
                /* When the range of RE_FRAMESPACE */
                /* Reception completion(Data since the second times)*/
                rcv_data_cnt++;                     /* Number of reception completion add */
                pulse[pulse_cnt] = 0;               /* Stopper setting */
                pulse_cnt++;                        /* Pulse storage position add */
                rcv_mode = RE_LEADER_CODE_H;        /* Mode transition */
                break;
            }
            rcv_mode = LEADER_CODE_H;               /* When the receive data error, mode clear */
            break;

        default:
            rcv_mode = IDLE;                        /* When the receive data error, mode clear */
            break;

    }

}


/*""FUNC COMMENT""*****************************************************************
* ID              : 1.5
*-----------------------------------------------------------------------------------
* Include         : Nothing
*-----------------------------------------------------------------------------------
* Declaration     : unsigned char cmp_pulse(
*                 :     unsigned short,
*                 :     unsigned short,
*                 :     unsigned short     )
*-----------------------------------------------------------------------------------
* Function        : Receive data range judgment
*                 :
*-----------------------------------------------------------------------------------
* Arguments       : d_pulse : Pulse value
*                 : hi      : Pulse maximum range value
*                 : low     : Pulse minimum range value
*-----------------------------------------------------------------------------------
* Returns         : 0:OK
*                 : 1:NG
*-----------------------------------------------------------------------------------
* Input           : unsigned short d_pulse : Pulse storage buffer
```

```
* Ountput          : Nothing
*-------------------------------------------------------------------------------
* Call functions   : Nothing
*-------------------------------------------------------------------------------
* Note             :
*-------------------------------------------------------------------------------
* History          : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""***********************************************************/
unsigned char cmp_pulse(unsigned short d_pulse,unsigned short hi,unsigned short low)
{

    if( (d_pulse > low) && (d_pulse < hi) ) {
        /* When the pulse value is an error range or less */
        return OK;
    }
    return NG;

}


/*""FUNC COMMENT""***************************************************************
* ID               : 1.6
*-------------------------------------------------------------------------------
* Include          : Nothing
*-------------------------------------------------------------------------------
* Declaration      : unsigned char judge_reversing_code(
*                  :     unsigned char,
*                  :     unsigned char                  )
*-------------------------------------------------------------------------------
* Function         : Reversing data code judgment
*                  :
*-------------------------------------------------------------------------------
* Arguments        : rtn0 : judgment of data code "L" (0data)return value
*                  :           0 : OK
*                  :           1 : NG
*                  : rtn1 : judgment of data code "L" (1data)return value
*                  :           0 : OK
*                  :           1 : NG
*-------------------------------------------------------------------------------
* Returns          : 0:OK
*                  : 1:NG
*-------------------------------------------------------------------------------
* Input            : unsigned char code_low_cnt : Reception code "L" counter
*                  : unsigned char rev_cnt      : Buffer position for
*                  :                              reversing data confirmation
* Ountput          : unsigned char code_low_cnt : Reception code "L" counter
*                  : unsigned char rev_cnt      : Buffer position for
```

```
*               :                                reversing data confirmation
*-------------------------------------------------------------------------------
* Call functions : set_reversing_code()
*               : cmp_reversing_code()
*-------------------------------------------------------------------------------
* Note          :
*-------------------------------------------------------------------------------
* History       : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""***************************************************************/
unsigned char judge_reversing_code(unsigned char rtn0, unsigned char rtn1)
{

    unsigned char rtn;      /* Return value */


    code_low_cnt++;         /* Reception code "L" counter add */

    if(code_low_cnt <= DATA_MAX_8_BIT_CNT){
        /* When it is not reversing data */
        set_reversing_code(rtn0, rtn1);     /* set_reversing_code function */

    }
    else if (code_low_cnt <= DATA_MAX_LOW_BIT_CNT){

        /* When the reversing data code */
        if(code_low_cnt == DATA_MAX_8_BIT_CNT+1){
            /* When the reversing data of the first data */
            rev_cnt = 0;    /* Buffer position for reversing data confirmation clear */
        }

        rtn = cmp_reversing_code(rtn0, rtn1);       /* cmp_reversing_code function */
        if(rtn != OK){
            /* When reversing data is NG */
            return NG;
        }
    }
    else{
            return NG;
    }

    if(rev_cnt > REV_PULSE_MAX){
        rev_cnt = 0;        /* Buffer position for reversing data confirmation clear */
    }
    return OK;

}
```

```
/*""FUNC COMMENT""*****************************************************************
* ID            : 1.7
*-------------------------------------------------------------------------------
* Include       : Nothing
*-------------------------------------------------------------------------------
* Declaration   : void set_reversing_code(
*               :     unsigned char,
*               :     unsigned char      )
*-------------------------------------------------------------------------------
* Function      : Reversing data code buffer setting
*               :
*-------------------------------------------------------------------------------
* Arguments     : rtn0 : judgment of data code "L" (0data)return value
*               :          0 : OK
*               :          1 : NG
*               : rtn1 : judgment of data code "L" (1data)return value
*               :          0 : OK
*               :          1 : NG
*-------------------------------------------------------------------------------
* Returns       : Nothing
*-------------------------------------------------------------------------------
* Input         : Nothing
* Ountput       : unsigned char rev_pulse[] : Buffer for reversing data confirmation
*               : unsigned char rev_cnt     : Buffer position for
*               :                            reversing data confirmation
*-------------------------------------------------------------------------------
* Call functions : Nothing
*-------------------------------------------------------------------------------
* Note          :
*-------------------------------------------------------------------------------
* History       : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
void set_reversing_code(unsigned char rtn0, unsigned char rtn1)
{

    if(rtn0 == OK){
        /* When data is 0 */
        rev_pulse[rev_cnt] = 0xF1;  /* Reversing code(0xF1) storage */
    }
    else if(rtn1 == OK){
        /* When data is 1 */
        rev_pulse[rev_cnt] = 0xF0;  /* Reversing code(0xF0) storage */
    }
    rev_cnt++;                      /* Buffer position for reversing data confirmation add */
}
```

```
/*""FUNC COMMENT""*****************************************************************
* ID            : 1.8
*-------------------------------------------------------------------------------
* Include       : Nothing
*-------------------------------------------------------------------------------
* Declaration   : unsigned char cmp_reversing_code(
*               :     unsigned char,
*               :     unsigned char              )
*-------------------------------------------------------------------------------
* Function      : Reversing data code buffer setting
*               :
*-------------------------------------------------------------------------------
* Arguments     : rtn0 : judgment of data code "L" (0data)return value
*               :          0 : OK
*               :          1 : NG
*               : rtn1 : judgment of data code "L" (1data)return value
*               :          0 : OK
*               :          1 : NG
*-------------------------------------------------------------------------------
* Returns       : 0:OK
*               : 1:NG
*-------------------------------------------------------------------------------
* Input         : unsigned char rev_pulse[] : Buffer for reversing data confirmation
*               : unsigned char rev_cnt     : Buffer position for
*               :                             reversing data confirmation
* Ountput       : unsigned char rev_cnt     : Buffer position for
*               :                             reversing data confirmation
*-------------------------------------------------------------------------------
* Call functions : Nothing
*-------------------------------------------------------------------------------
* Note          :
*-------------------------------------------------------------------------------
* History       : 2005.07.30  Ver 1.00
*""FUNC COMMENT END""*************************************************************/
unsigned char cmp_reversing_code(unsigned char rtn0, unsigned char rtn1)
{

    /* When data is reversing data */
    if( rtn0 == OK ){
        /* When data is 0 */
        if(rev_pulse[rev_cnt] == 0xF0){
            /* When data reverses */
            rev_cnt++;
            return OK;
        }
    }
```

```
    else if(rtn1 == OK){
        /* When data is 1 */
        if(rev_pulse[rev_cnt] == 0xF1 ){
            /* When data reverses */
            rev_cnt++;
            return OK;
        }
    }
    return NG;

}
```

10. 参考ドキュメント

ハードウェアマニュアル

M32C/85 グループハードウェアマニュアル Rev.1.0
（最新版をルネサス テクノロジホームページから入手してください。）

テクニカルアップデート/テクニカルニュース

（最新の情報をルネサス テクノロジホームページから入手してください。）

11. ホームページとサポート窓口

ルネサス テクノロジ M16C ホームページ

http://japan.renesas.com/m16c

M16C ファミリ MCU 技術サポート窓口

E-mail: csc@renesas.com

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2005.09.15	-	・初版発行
1.01	2006.12.10	-	・フローチャート図 ・データ OK?→リモコンデータの受信あり? ・0 データ NG?→カスタム/データコードは 1 データ? ・0 or 1 データ OK?→カスタム/データコードは 0 データか 1 データ? ・反転データ NG?→反転データの受信あり? ・0 データ?→カスタム/データコードは 0 データ? ・1 データ?→カスタム/データコードは 1 データ?

本資料ご利用に際しての留意事項

1. 本資料は、お客様に用途に応じた適切な弊社製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について弊社または第三者の知的財産権その他の権利の実施、使用を許諾または保証するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例など全ての情報の使用に起因する損害、第三者の知的財産権その他の権利に対する侵害に関し、弊社は責任を負いません。
3. 本資料に記載の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、あるいはその他軍事用途の目的で使用しないでください。また、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、それらの定めるところにより必要な手続を行ってください。
4. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例などの全ての情報は本資料発行時点のものであり、弊社は本資料に記載した製品または仕様等を予告なしに変更することがあります。弊社の半導体製品のご購入およびご使用に当たりましては、事前に弊社営業窓口で最新の情報をご確認頂きますとともに、弊社ホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意下さい。
5. 本資料に記載した情報は、正確を期すため慎重に制作したものですが、万一本資料の記述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、弊社はその責任を負いません。
6. 本資料に記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を流用する場合は、流用する情報を単独で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断して下さい。弊社は、適用可否に対する責任は負いません。
7. 本資料に記載された製品は、各種安全装置や運輸・交通用、医療用、燃焼制御用、航空宇宙用、原子力、海底中継用の機器・システムなど、その故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのあるような機器・システムや特に高度な品質・信頼性が要求される機器・システムでの使用を意図して設計、製造されたものではありません（弊社が自動車用と指定する製品を自動車に使用する場合を除きます）。これらの用途に利用されることをご検討の際には、必ず事前に弊社営業窓口へご照会下さい。なお、上記用途に使用されたことにより発生した損害等について弊社はその責任を負いかねますのでご了承願います。
8. 第７項にかかわらず、本資料に記載された製品は、下記の用途には使用しないで下さい。これらの用途に使用されたことにより発生した損害等につきましては、弊社は一切の責任を負いません。
   1）生命維持装置。
   2）人体に埋め込み使用するもの。
   3）治療行為（患部切り出し、薬剤投与等）を行なうもの。
   4）その他、直接人命に影響を与えるもの。
9. 本資料に記載された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、実装条件およびその他諸条件につきましては、弊社保証範囲内でご使用ください。弊社保証値を越えて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、弊社はその責任を負いません。
10. 弊社は製品の品質および信頼性の向上に努めておりますが、特に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。弊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、誤動作防止設計などの安全設計（含むハードウエアおよびソフトウエア）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エアは、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致します。
11. 本資料に記載の製品は、これを搭載した製品から剥がれた場合、幼児が口に入れて誤飲する等の事故の危険性があります。お客様の製品への実装後に容易に本製品が剥がれることがなきよう、お客様の責任において十分な安全設計をお願いします。お客様の製品から剥がれた場合の事故につきましては、弊社はその責任を負いません。
12. 本資料の全部または一部を弊社の文書による事前の承諾なしに転載または複製することを固くお断り致します。
13. 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弊社営業窓口までご照会下さい。