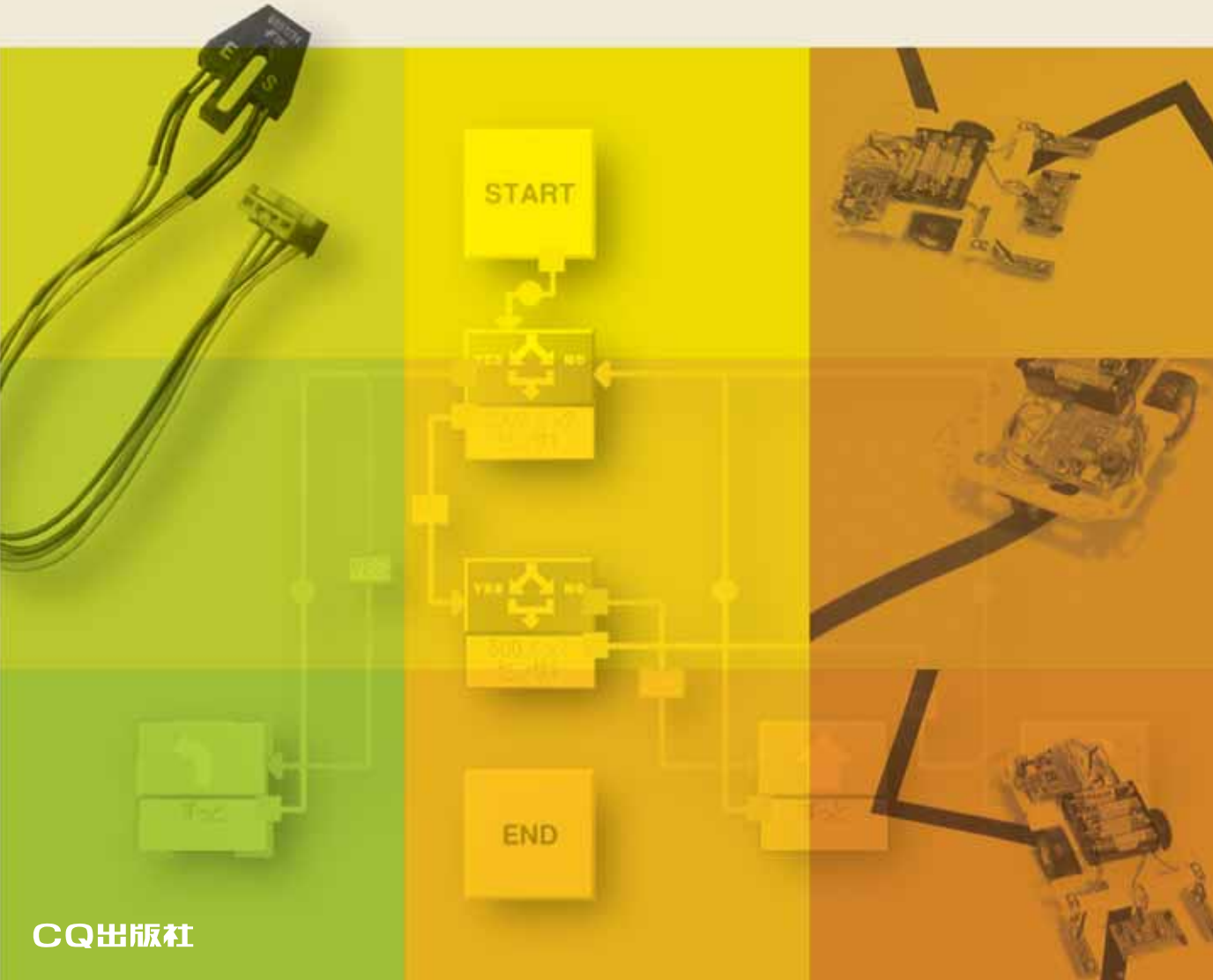
電子工作
マイコン
シリーズ

ライントレース・ロボットをC言語で実践的に使う

はじめてのH8マイコン

島田義人［著］
Yoshihito Shimada



CQ出版社

ISBN978-4-7898-4164-1

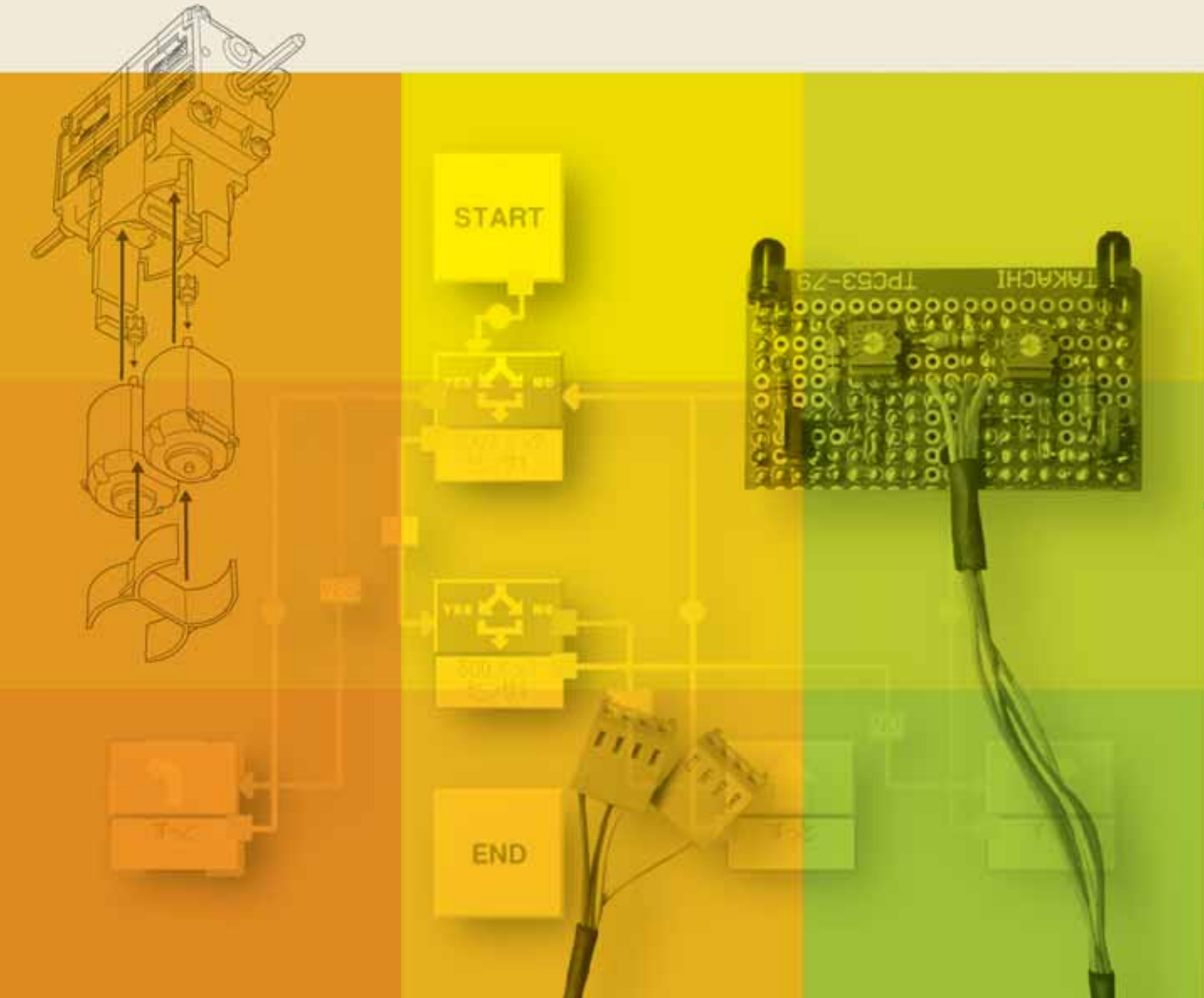
C3055 ¥2400E

CQ出版社

9784789841641

定価：本体2,400円（税別）

1923055024007



目　次

目　次

目　次

目　次

目　次

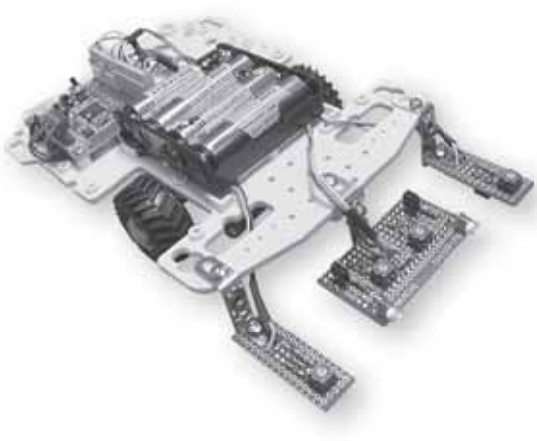
目　次

目　次

目　次

目　次

目　次



a ビュート チェイサーをベースに製作した例

b ビュート ローバーをベースに製作した例

写真1-1　製作したライントレース・ロボットの外観

●はじめに

電子機器を制御するうえで，マイコンはなくてはならない部品です．世の中で使われているマイコンには数多くの種類があり，各社特徴のあるマイコンを開発しています．中でもルネサス エレクトロニクス（株）（Renesas Electronics）[*1]のH8は，高性能なマイコンとしてアマチュアのロボットに使われる頻度が高く，ライントレース・ロボットの制御などにも広く利用されています．

H8マイコン応用編ではライントレース・ロボット（写真1-1）の製作を紹介していきます．写真1-2はライントレース・ロボットに搭載されているH8マイコン・ボード“VS-WRC003”です．現在，ヴイストン（株）からC言語でプログラミング学習ができるマイコン・ボードとしていろいろな機能が使え，2,982円（2010年4月現在）という比較的に安価な値段で販売されています．

ボードの中央部にあるのが，16ビット・マイコンH8/36064です．このH8マイコンはルネサス エレクトロニクス（株）が開発・製造しているH8/300H Tinyシリーズのマイコンの一つで，一般的な制御を行うには十分な機能をもっています．H8マイコンの基礎を学ぶにあたり，まずはプログラム開発環境を準備したあとで，ボードに実装されている二つのLEDを点灯させることからはじめてみましょう．

2010年8月にビュート チェイサーの姉妹機としてビュート ローバー（Beauto Rover）が登場しました．価格は6,300円（執筆時点）です．そこに搭載されているH8マイコン・ボード（VS-WRC003LV）は3,465円（執筆時点）と高めですが，モータ出力のパワーアップ化，HIDシリアル変換IC搭載，二つの赤外線センサを搭載，低電圧動作化などいろいろ機能が向上しています．本書ではビュート チェイサーからビュート ローバーへ容易に移行して取り組めるように，ビュート ローバーに関する内容も各章の末尾に記載しています．

＊1：2010年4月1日に名称変更．それまではルネサス テクノロジ（株）．

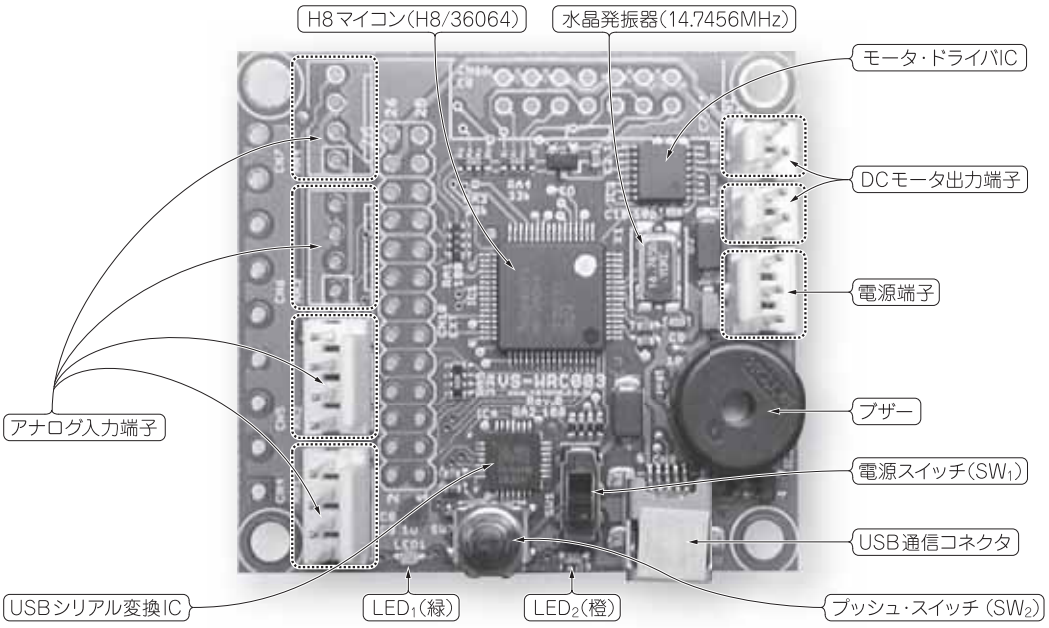


写真1-2　H8マイコン・ボード(VS-WRC003)の外観

1-1　マイコン学習に最適なH8マイコン・ボード

これから使用するマイコン・ボードがどんなものであるのか，ここで簡単に紹介しておきましょう．これを見ると，きっと“すぐにほしくなる”と思います．

●手軽にC言語プログラミングの学習ができるマイコン・ボードである

H8マイコン・ボードには，LED(2個)，ブザー，DCモータ出力(2ポート)，アナログ入力(4ポート)が標準で装備されているため，ボード単体で手軽にC言語プログラミングの学習ができるでしょう．また，拡張ポートも備えており，ピン・ヘッダ追加でI/O増設などH8マイコンの機能をフルに活用することもできます．

●パソコンで開発したプログラムはUSBケーブル1本でマイコンに書き込める

H8マイコン・ボードにはUSBシリアル変換ICが搭載されています．パソコン上で開発した制御プログラムをUSBケーブル1本で書き込むことができます．これは，特別な書き込み器が一切不要ということです．

また，H8マイコン(H8/36064)は1万回も書き換え可能なフラッシュROM[*2]を内蔵しているので繰り返し学習に最適です．単純に考えて，毎日3回マイコンにプログラムを書き込んだとしても，なんと9年以上も使える計算です．

＊2：フラッシュROMは電源を切っても内容は消えません．

前章までは，モニタを使って自分の書いたプログラムをH8マイコンのRAMに書き込んで実行してきました．モニタの機能を使うことで，プログラムを一時停止したり，レジスタやメモリの内容を確認できたりして，プログラムを開発する上でとても便利でした．

ところで，RAMは一度電源を切ると記録内容が消滅してしまうメモリです．そのため，電源をONするたびにパソコンからプログラムをRAMにダウンロードしなければなりません．これでは不便ですので，本章では自分の書いたプログラムをROM[*1]に書き込む（ROM化する）手法を説明します．

ROMは電源を切っても内容が消えない読み出し専用メモリです．ROMに書き込んだプログラムは電源投入直後から動作します．また，パソコンと USB ケーブルを接続せずに，乾電池などの外部電源でマイコンを動かす（スタンドアロンで動かす）ことができるようになります．

4-1 プログラムをROMに書き込むための準備

●まずはHEWを起動する

Windowsの［スタート］メニューから［プログラム］>［Renesas］>［High-performance Embedded Workshop］>［High-performance Embedded Workshop］でHEWを起動します．ここではLEDを点滅させるプログラム“led_blink”をROMに書き込んでみましょう．図4-1に示すように「最近使用したプロジェクトワークスペースを開く」の▼ をクリックしてみてください．もし，ここにファイルが該当しない場合は，「別のプロジェクトワークスペースを参照する」にチェックを入れてファイルを探してみてください．

●ビルドコンフィグレーションを選択する

LEDを点滅させるプログラム“led_blink”を読み込めたでしょうか．ここからROM化作業のはじまりです．まず，ビルドコンフィグレーションを選択します．図4-2に示すように，現状は「Debug」となっていると思います．ここで，ツールバーのドロップダウン/リスト・ボックスから「Release」を選んでください．

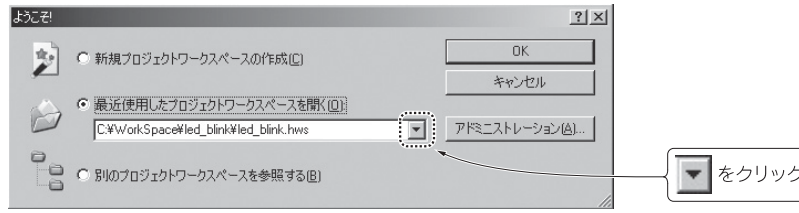


図4-1　最近使用したプロジェクトワークスペースを開く

＊1：このマイコンROMはフラッシュROMなので，複数回の書き込みを行えます，

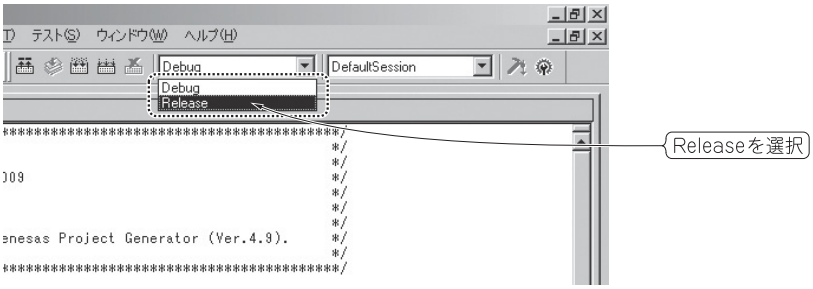


図4-2　ビルドコンフィグレーションで「Release」を選択

(a) ROMおよびRAMへ割り付けられたアドレスとセクションの初期設定(HEWのデフォルト設定)

Address	Section
0x00000400	PResetPRG
	PIntPRG
0x00000800	P
	C
	C$DSEC
	C$BSEC
	D
0x0000FB80	B
	R
0x0000FE80	S

ROM領域
RAM領域

(b) モニタを使ってRAM上で動作させる場合のアドレスとセクションの設定例

Address	Section
0x0000F840	CV0
0x0000F890	PResetPRG
	P
	C
	C$DSEC
	C$BSEC
	D
0x0000FB80	B
	R
0x0000FE80	S

RAM領域

(c) ROM化してスタンドアロン

Address	Section
0x00000000	CV0
0x00000400	PResetPRG
0x00000800	P
	C
	C$DSEC
	C$BSEC
	D
0x0000FB80	B
	R
0x0000FE80	S

ROM領域
RAM領域

図4-3　ROMおよびRAMへの割り付けたアドレスとセクションの対応

●プログラムをROM領域に配置指定する

プログラムの配置を指定する場合には，まずツールチェインを開きます．HEW画面上側にあるメニューバーから［ビルド］-［H8S, H8/300 Standard Toolchain…］を選択します．ツールチェインの画面が開いたら［最適化リンカ］のタブを選択し，［カテゴリ］の選択リストから［セクション］を選択します．

図4-3にROMおよびRAMへ割り付けたアドレスとセクションの対応を示します．比較のために図 a 初期設定（HEW のデフォルト設定）と，図 b RAM用に配置した例も掲載しておきます．ここでは，モニタを使ってRAM上で動作確認したあとのプログラムを，セクションのアドレス設定の変更によりROM化してみます．手順としては，`CV0`（CVは大文字のアルファベット，0は数字のゼロ）をアドレスの先頭`0x0000`に配置します．また，`PResetPRG`やp以下は初期設定と同じROM領域に配置します．`PIntPRG`は削除してください．配置した結果を図 c に示します．

●プログラムをビルドする

H8マイコンへ書き込むためのファイル［ロード・モジュール］をビルドすることにより作成します．メニューバーの［ビルド］メニューから［ビルド］をクリックするか，ツールバーからアイコンをクリックする方法などでビルドを実行してみてください．

5-2 volatile修飾子の存在意義を理解しよう

H8マイコンなどのハードウェアに近い分野のC言語プログラミングでは“volatile”というキーワードをよく見かけます．volatile修飾子はパソコン上で動かすC言語プログラムを書く人だとあまり使わない（縁がない）かもしれません．しかし，H8マイコンなどのC言語プログラミングでは，volatile修飾子の使い方や存在意義を知らないと予期せぬ不具合に見舞われることがあります．そこで，volatile修飾子について解説しておきましょう．

●HEWのコンパイラはプログラムを最適化して実行形式のファイルを作ってくれる

HEWでビルドを実行すると，コンパイラと呼ばれるソフトウェアがC言語のソース・プログラムをH8マイコンが実行できる形式のファイル（ロード・モジュール）に変換してくれます．このとき，コンパイラはプログラムをできるだけコンパクトに最適化しようとします．つまり実行する命令が少なくなれば，それだけプログラムを格納するのに必要なメモリが節約でき，さらに実行する速さが向上するわけです．

volatile修飾子はコンパイラによる最適化を抑制するためのキーワードです．ではなぜ，この機能を使ってわざわざ抑制しなければならないのでしょうか？ 次に説明していきましょう．

●LEDの点滅動作が非常に速くなる？

LED点滅プログラム“led_blink.c”を例にとり，volatileの有無でH8マイコンの動作にどのような違いが現れるのかを試してみましょう．“volatile unsigned int i;”のvolatileをコメント・アウトして削除するか，もしくは単に“unsigned int i;”として記述してみます．

結果はLEDの点灯のようすを見ればはっきりと違いがわかると思います．変数iの宣言でvolitileを使わないプログラムでは，点灯 / 消灯しているようすがわからないほど点滅動作が非常に速くなります．

●ループを回してウェイト（待ち時間）を作るときには“volatile”を使おう

LEDの点滅動作が非常に速くなるのは一体なぜでしょうか？ 図5-6に示すように，H8マイコン内部にはCPU（Central Processing Unit）と呼ばれるコンピュータの心臓部があります．そのCPUを中心として，外部入力，外部出力，そしてメモリがつながっています．CPUの内部には「レジスタ」と呼ばれる「一時的な記憶エリア」があります．H8マイコンでは16ビット長のレジスタを16本もっています（一般的にレジスタの数は，1本，2本と数えるようです）．

メモリはいろいろなデータ（プログラムの命令もデータの一種と考える）を蓄えるファイル・キャビネットのようなものです．一方，CPU内レジスタは，実際にCPUがデータを加工するときに一時的に記憶する場所と考えてください．マイコンはメモリのデータを加工して書き換えているように思いますが，実際はCPU内レジスタが一時的にデータを記憶し，それをCPUが加工して結果をメモリに戻しています．

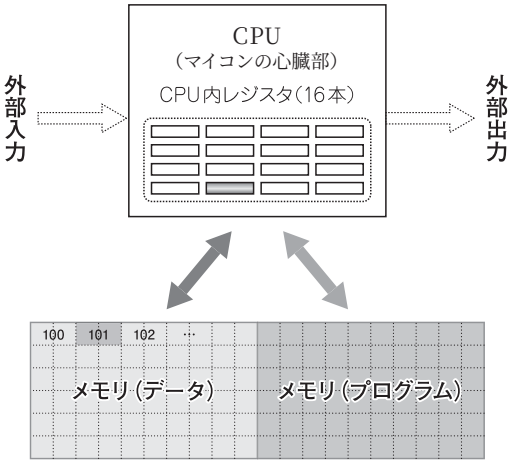


図5-6　データ処理のイメージ
（CPU内レジスタが一時的にデータを記憶し，それをCPUが加工して結果をメモリに戻す）

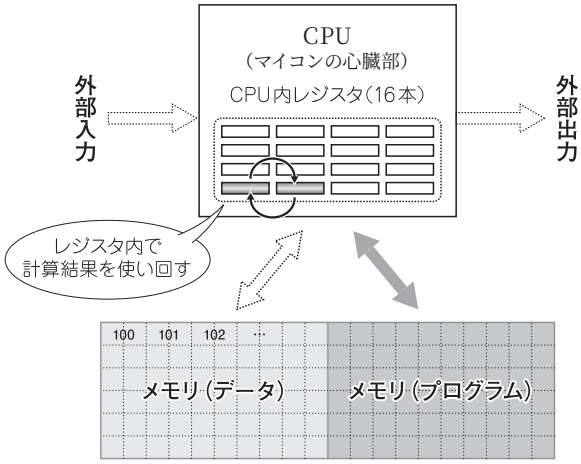


図5-7　最適化されたデータ処理のイメージ

LED点滅プログラム“led_blink.c”では，次のようなfor文で無意味なループを回してウェイト（待ち時間）を作っています．

```
for ( i = 0; i < 60000; i++);
```

マイコンとしては，このように変数iの値を何度も何度も使って計算を繰り返していくようなプログラムの場合，いちいちメモリの読み書きを行っているのは非常に効率が悪いことになります．実行スピードはメモリよりCPU内レジスタのほうがはるかに速いのです．

そこで，図5-7に示すようにコンパイラはCPU内レジスタの内容をメモリに書き出すことなく，どんどん使い回すという最適化を行います．

最適化は処理速度の向上にとって大変ありがたいのですが，ここではある程度の時間待ちをしてほしいわけです．メモリに1回1回，値の変化を書き込ませたらループ回数も少なく済みますから，そのためにvolatileを付けて最適化を抑制しています．

volatileをつけずにプログラムをビルドした場合，コンパイラは変数の利用頻度や範囲を元に，変数をCPU内レジスタに割り当てるのか，それともメモリ（RAM）に割り当てるのかを自動的に決めます．繰り返し使われる変数は最適化によって優先的にCPU内レジスタに割り当てられますが，レジスタの数が16本と限りがあるため，場合によってはメモリ（RAM）に割り当てられることがあります．そのため，volatileを付けないとプログラムによってはウェイト（待ち時間）に差が生じてしまうことがあります．

●内部I/Oレジスタの宣言には必ずvolatileをつけよう

プログラムをビルドすると，コンパイラは利用頻度の高い変数を優先的にCPU内レジスタに割り当ててくれます．しかし，変数をCPU内レジスタに割り当てられては困る場合があります．それ

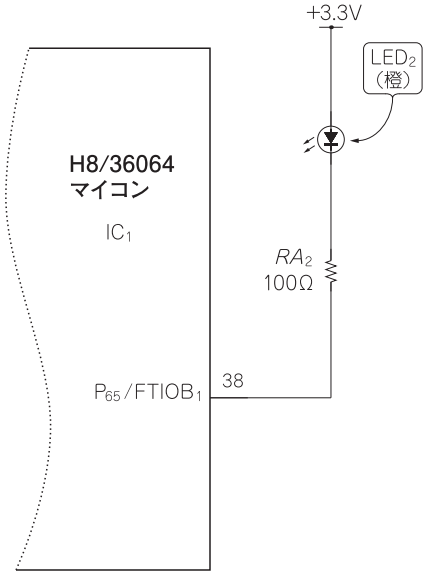


図8-1　LED_2(橙)とH8マイコンとの接続
ローバーの接続は図8-10を参照.

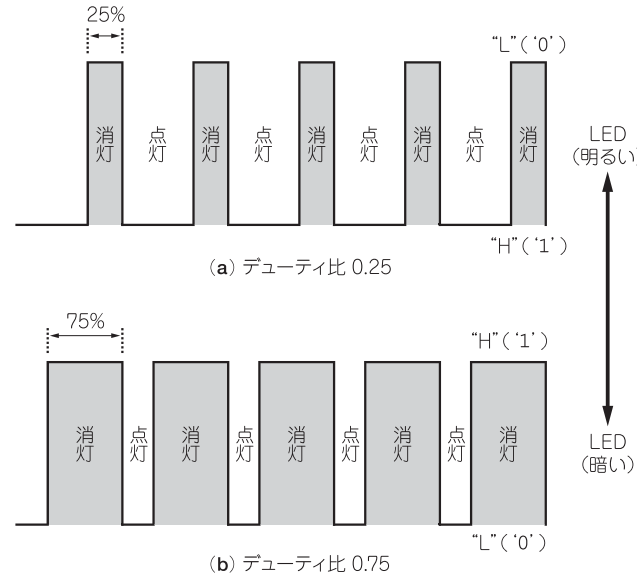


図8-2　PWM信号波形とLEDの点灯/消灯の関係

マイコンにはわりと正確な時間を刻んでいる時計のようなもの「タイマ・モジュール」があります.前章では汎用的な8ビット・タイマである「タイマV」を使って圧電ブザーを鳴らしてみました.ここでは本格的な16ビット・タイマである「タイマZ」を紹介します.そして,タイマZを使ってLEDの明るさを自由に変えられる調光にチャレンジしてみましょう.

8-1　LEDの明るさを変える調光のしくみ

これまでモニタを使いながらLEDを点灯/消灯させる実験をしましたね.今回は38番ピン(P_{65}/$FTIOB_1$)に接続されているLED_2(橙)の明るさを変化させてみましょう.図8-1に示すようにLEDのアノード側は+3.3Vの電源に接続され,カソード側は100Ωの抵抗を介してH8マイコンの入出力ポートに接続されています.このように接続すると,ポートに"L"('0')を出力するとLEDが点灯し,逆にポートを"H"('1')にするとLEDが消灯します.

ところで,H8マイコンのポートは"L"('0')と"H"('1')の二つの状態しか出力することができません.しかし,この"L"('0')と"H"('1')の二つの状態をすばやく切り換えることで,人間の目にはLEDが連続して発光しているように見えるのです.

つまり,図8-2に示すようにPWM(Pulse Width Modulation)と呼ばれる1周期に対するHighの時間比率(デューティ比)を変えた信号を使うことで,LEDの明るさを変えることができます.LEDは"L"('0')出力で点灯するため,図 a のようにデューティ比を小さくすると点灯している割り合が大きくなり,LEDが明るく発光しているように見えます.逆に図 b のようにデューティ比を大きくすると消灯している割合が大きくなり,LEDが暗く発光しているように見えます.

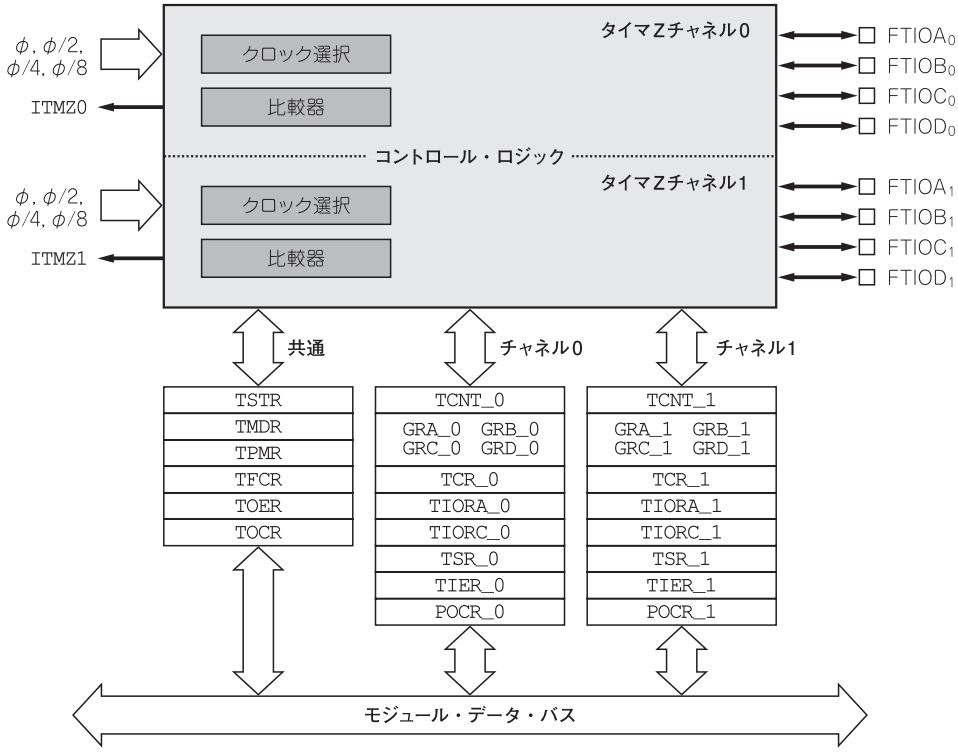


図8-3　タイマZの内部ブロック構造

●H8/36064マイコンには5種類のタイマ・モジュールがある

前章で説明したように，H8/36064マイコンには，タイマB1，タイマV，タイマZ，14ビットPWM，ウォッチ・ドッグ・タイマの五つのタイマ・モジュールが内蔵されています．本章のLEDの調光には，本格的な16ビット・タイマである「タイマZ」を使います．

タイマZの特徴として，「アウトプット・コンペア」，「インプット・キャプチャ」という機能があります．アウトプット・コンペアは，ある規則性をもった信号を出力してくれる機能です．一方，インプット・キャプチャはその反対の機能です．ある規則性をもつ信号の長さを計測してレジスタに格納するというものです．LEDの調光にはアウトプット・コンペアの機能を使います．

8-2　タイマZの内部構造

タイマZの内部構成は図8-3のようになっています．$FTIOA_0$～$FTIOD_0$端子，$FTIOA_1$～$FTIOD_1$端子の合計8端子がタイマZの入出力端子です．LED_2（橙）は$FTIOB_1$端子に接続されています．

カウンタの入力クロックは4種類の内部クロック（φ，φ/2，φ/4，φ/8）もしくは外部クロックから選択可能です．タイマZはチャネル0とチャネル1の二つのタイマをもって，それぞれ独立して動作させることができます．

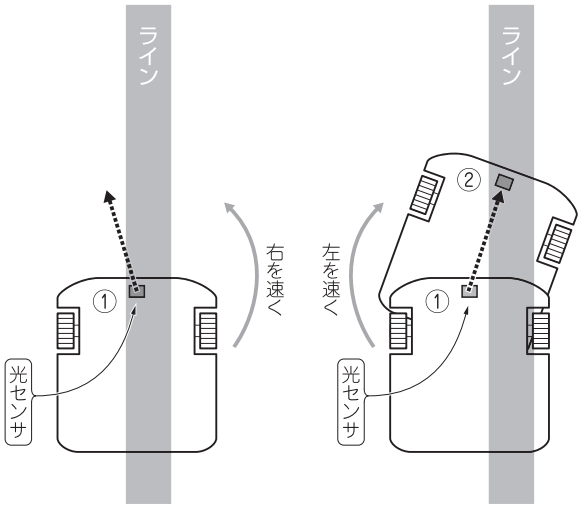


図11-3　ライントレース・ロボットの動作パターン

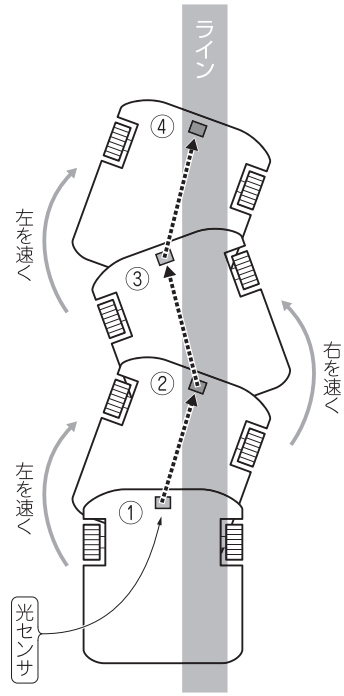


図11-4　ライントレース・ロボットがラインに沿って進むようすの説明図

11-2　ライントレース・ロボットがラインに沿って動く原理を考えてみよう

ライントレース・ロボットはフォト・リフレクタと呼ばれる光センサ使って，床から反射した光の強さを検知することによって線を認識することがわかりました．まずは簡単のため，光センサを1個搭載したライントレース・ロボットの動きを考えてみましょう．

ここでは，図11-3のようにライン左側の境界をセンサが検知する場合を考えてみたいと思います．動作のパターンは次の2通りになります．

Ⓐ 光センサがライン上にいると検知した場合：

右側の車輪の速度を速く回転させる（もしくは左側の車輪の速度を遅く回転させる）ようにモータを制御します．すると，車体は進行方向に対して左寄りに進みます．

Ⓑ 光センサがライン外にいると検知した場合：

左側の車輪の速度を速く回転させる（もしくは右側の車輪の速度を遅く回転させる）ようにモータを制御します．すると，車体は進行方向に対して右寄りに進みます．

この動作の繰り返すようすを図に示したのが図11-4です．線の上にいたら左寄りに進み，線の上にいなければ右寄りに進むという動作を繰り返していくと，ジグザグではありますがライントレース・ロボットはライン左側の境界に沿って蛇行しながら動くようになります．

なお，ライン右側の境界を光センサが検知する場合は，動作パターンはⒶとⒷが逆になります．

写真11-2　L字金具に光センサを取り付けたようす

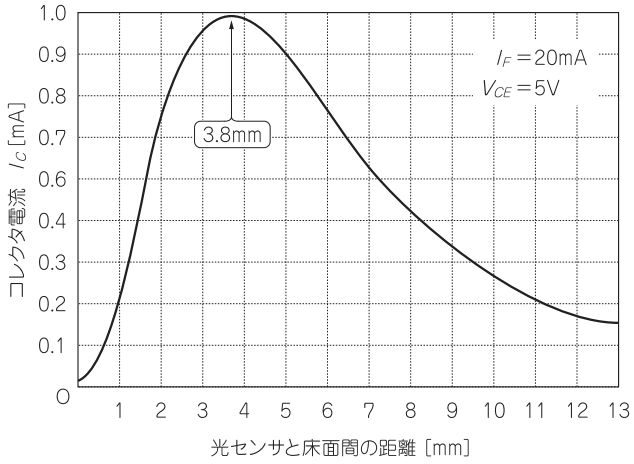


図11-5　光センサQRB1134の出力特性

11-3　ビュート チェイサーをライントレース・ロボットに変身させてみよう

動作原理がわかったところで，ライントレース・ロボットの製作をしてみましょう．これまでビュート チェイサーの製作をしてきた方なら製作はそれほど難しくはありません．ビュート チェイサーの前方に配置した光センサを，写真11-2に示すように床面に向けて固定するだけです．車高（ベース下面と床面間の距離）は約30mm程度ですので，これより短いL字金具などを使って固定するとよいでしょう．光センサ面と床面間の距離は，図11-5に示す出力特性を参考に調整してください．約3.8mmに調整するとコレクタ電流が最も大きくなります．これでハードの改造は終了です．

11-4　車輪の回転の制御はどのように行うの？

まずは，電池駆動用の小型DCモータの回転速度を変える方法について考えてみましょう．一般的な方法として，電源電圧を変化させることが考えられます．たとえば，つなぐ電池の本数を変えて1本（1.5V）よりも2本（3V）のほうがモータは速く回り，2本（3V）よりも3本（4.5V）のほうがより速く回るというわけです．

ところで，H8マイコンのI/Oポートは，出力する電圧を増減するD-Aコンバータの機能を持ち合わせておらず，High（3.3V）とLow（0V）の二つの状態しか出力することができません．このままでは，モータの回転をONとOFFのどちらかにしか取り得ないということになります．

しかし，このHighとLowをすばやく切り換えることで，中間量の電流が流れているときと同じ状態を作り出すことができます．このとき，図11-6に示すように，1周期*2に対するHighの時間比率（デューティ比）を変えることで電流の平均が変わります．これをPWM（Pulse Width

＊2：ONとOFFの1組で1周期となる．