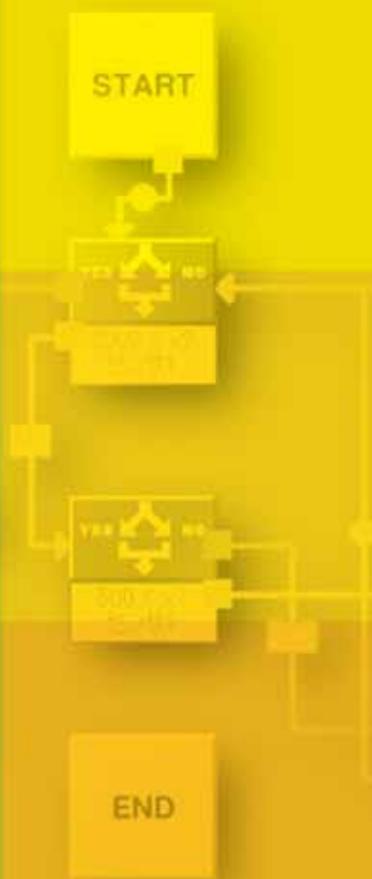


# はじめての H8マイコン



島田義人 [著]  
Yoshihito Shimada



ISBN978-4-7898-4164-1

C3055 ¥2400E

**CQ出版社**

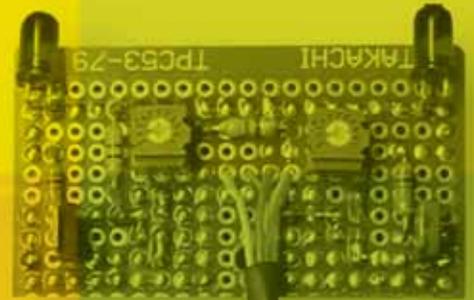
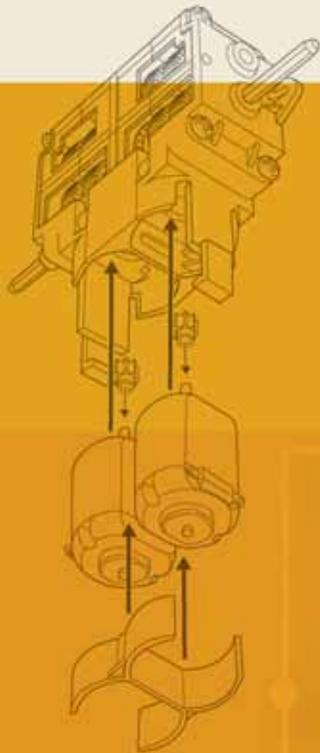
定価：本体2,400円（税別）



9784789841641



1923055024007



## 目次

はじめに	2
<b>第1章 さあ、H8マイコンをはじめよう</b>	<b>13</b>
はじめに	14
▶ 1-1 マイコン学習に最適なH8マイコン・ボード	15
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	15
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	15
USB接続で電源給電が簡単である	16
▶ 1-2 充実したプログラム開発環境	16
フローチャート形式で簡単なプログラム作成ができるビュート・ビルダー・ネオ	16
マイコン用のプログラムを効率良く開発できる“HEW”	16
マイコンにプログラムを書き込むツール“FDT”	17
マイコンをパソコン上で監視/制御できる“Hterm”	18
▶ 1-3 H8マイコン・ボードの回路について	20
USB接続で電源給電が簡単なH8マイコン・ボード	20
H8マイコン周辺の回路構成	22
外付け部品が少ないUSBシリアル変換IC( CP2102 )	24
H8マイコン・ボードの全回路図はWebページから入手できる	25
▶ 1-4 ビュート ローバー搭載H8マイコン・ボード( VS-WRC003LV )の紹介	25
モータ出力のパワーアップ化	28
HIDシリアル変換IC搭載	28
低電圧動作化	28
ビュート ローバー用H8マイコン・ボードの回路図やピン配置はWebから入手できる	28
<b>第2章 H8マイコン・ボードのLEDを点灯させてみよう</b>	<b>29</b>
▶ 2-1 H8マイコンとLEDの接続について	30
▶ 2-2 モニタのコマンドを使って二つのLEDを点灯/消灯してみよう	30
▶ 2-3 H8マイコンの構造	32
アドレスって何?	32
H8マイコンのアドレス空間とメモリ・マップ	32
H8マイコンのI/Oポートの構造	33
H8マイコンの内部I/Oレジスタ	33
▶ 2-4 C言語プログラミングを始めてみよう	35
プログラミングとは、行いたい操作を記述することである	35
新規プロジェクトワークスペースの作成からはじめます	36

# 目次

実行するプログラムにふさわしいワークスペース名をつけておこう	36
使うマイコンを選択しておこう	36
HEWのメイン画面を見てみよう	37
ワークスペース・ウィンドウに表示されているファイルを見てみよう	37
LEDを点滅させるプログラムをmain関数に書いてみよう	38
HEWが自動生成したファイルを改変する	39
▶ 2-5 プログラムはマイコンのどこに格納するの？	41
H8マイコンのアドレス空間とメモリ・マップ	41
プログラムの配置指定	42
▶ 2-6 プログラムをビルドしてみよう	43
ビルド実行前に割り込み関数ファイル“i ntprg.c”をビルドから除外する	43
ビルドを実行する	44
▶ 2-7 H8マイコンへプログラムをダウンロードしてみよう	44
プログラムのダウンロード方法	44
ソース・リストとメモリ・マップの表示	45
▶ 2-8 プログラムを実行してLEDを点滅させてみよう	46
▶ 2-9 ビュート ローバー用H8マイコン・ボード( VS-WRC003LV )の場合	46
第3章 C言語プログラムのデバッグをしてみよう	49
▶ 3-1 モニタの便利な機能( コマンド )の紹介	50
▶ 3-2 Htermの便利な機能の紹介	50
▶ 3-3 Htermの主要な機能の使い方	53
ソース・リストの表示	53
周辺機能のレジスタ表示と変更	53
メモリ内容の表示と変更	54
コマンドのヒストリ機能	55
ブレーク・ポイントの設定 / 解除と確認	55
ソース・レベルのステップの実行( Pstep )	56
▶ 3-4 C言語プログラムのデバッグをしてみよう	56
まずはバグを含んだプログラムを準備しよう	57
バグを含んだプログラムを実行してみてLEDの状態を確認する	57
ブレーク・ポイントを設定する	59
メモリやレジスタの値を確認する	59
最初のブレーク・ポイントまでプログラムを実行する	59
続いてPstepコマンドでステップ実行する	60
Gコマンドで次のブレーク・ポイントまでプログラムを実行する	60
続いてPstepコマンドをステップ実行する...レジスタの値やLEDが発光するようすを見てバグを発見する	60

# 目次

同じようにしてブレーク・ポイントまでの実行とPstepコマンドによるステップ実行を繰り返す	61
続いてGコマンドで次のブレーク・ポイントまでプログラムを実行するはずが...無限ループに突入	61
<b>第4章 C言語プログラムをROM化してみよう</b>	<b>63</b>
▶ 4-1 プログラムをROMに書き込むための準備	64
まずはHEWを起動する	64
ビルドコンフィグレーションを選択する	64
プログラムをROM領域に配置指定する	65
プログラムをビルドする	65
▶ 4-2 FDTを起動してプログラムをROMへ書き込もう	66
プログラムを実行してLEDを点滅させてみよう	67
▶ 4-3 新規からROM化用のC言語プログラムを作成する手順	67
HEWのプロジェクトワークスペースの作成からはじめる	67
LEDを点滅させるプログラムをmain関数に記述する	67
リセット・プログラム・ファイル“resetprg.c”を改変する	67
ビルドコンフィグレーションを選択する	69
プログラムの格納領域を確認する	69
プログラムをビルドしたあとで書き込んで実行する	69
<b>第5章 C言語プログラミングの基礎知識</b>	<b>71</b>
▶ 5-1 変数を理解しよう	72
変数は使う前に宣言しておく必要がある	72
変数の名前はキーワード以外なら自由に付けられる	73
変数のデータ型の種類	73
符号付きか、符号なしかを明示する型修飾子	74
変数とは値を入れて(記憶させて)おく箱のようなもの	74
H8マイコンを使って実際にオーバーフローを体験してみよう	75
▶ 5-2 volatile修飾子の存在意義を理解しよう	76
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	76
LEDの点滅動作が非常に速くなる?	76
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	76
内部I/Oレジスタの宣言には必ずvolatileをつけよう	77
まずはvolatile宣言したプログラムの動作を確認してみる	78
volatileがないとH8マイコンが動いてくれない?	79
コンパイラの最適化は設定により回避できる	80
▶ 5-3 ポインタを理解しよう	80
ポインタとは?	81

# 目次

ポインタを「デスクトップ上におくショート・カット・アイコン」としてイメージする	81
H8マイコンにおけるポインタの主な用途	82
I/Oレジスタにわかりやすい定義名をつけよう	83
▶ 5-4 I/Oレジスタ定義ファイルを活用しよう	84
I/Oレジスタ定義ファイルの中身を覗いてみよう	85
構造体(struct)について	85
共用体(union)について	87
内部I/Oレジスタを参照する場合の記述方法	87
I/Oレジスタ定義ファイルを活用してスマートなプログラムにしてみよう	88
<b>第6章 スイッチを使ってLEDを制御してみよう</b>	<b>89</b>
▶ 6-1 プッシュ・スイッチ周辺のマイコン・ボードの回路構成	90
▶ 6-2 C言語プログラムを作成してみよう	90
プログラムの実行	90
▶ 6-3 プログラムの概要	92
プログラム全体の概要	92
初期設定関数(portINIT)の概要	92
スイッチ読込関数(getSW)の概要	93
LED制御関数(LED)の概要	93
▶ 6-4 C言語のプログラムは関数の集まりである	94
関数の概念とその必要性	95
関数のプロトタイプ宣言	95
関数の定義	95
引数や戻り値がない場合の関数の書式	96
▶ 6-5 変数のスコープ	97
ローカル変数とグローバル変数	97
関数の呼び出しと引数の引渡し	98
▶ 6-6 ビュート ローバー用H8マイコン・ボード(VS-WRC003LV)の場合	99
<b>第7章 タイマ機能を使って圧電ブザーを鳴らしてみよう</b>	<b>101</b>
▶ 7-1 まずはLED点灯/消灯の知識で圧電ブザーを鳴らしてみよう	102
圧電ブザーを簡易的に鳴らすC言語プログラムを作成する	102
プログラムを実行し簡易的に圧電ブザーを鳴らしてみる	104
タイマ機能を使って圧電ブザーを正確に鳴らしてみよう	104
H8/36064マイコンには5種類のタイマ・モジュールがある	104
▶ 7-2 8ビット・タイマのタイマVの内部構造	105

# 目次

タイマV関連の主なレジスタ	105
TCNTV入カクロックとカウント条件	108
▶ 7-3 タイマVの動作説明	108
各種レジスタ設定	109
(Column) 7-1 圧電ブザーって何?	110
▶ 7-4 タイマVで圧電ブザーを鳴らすC言語プログラムを作成する	112
C言語プログラムを実行して圧電ブザーを鳴らしてみよう	112
(Column) 7-2 音階のちょっとした知識	112
▶ 7-5 ピュート ローバー用H8マイコン・ボード( VS-WRC003LV )の場合	114
<b>第8章 タイマ機能を使ってLEDの明るさを自由に変えてみよう</b>	<b>115</b>
▶ 8-1 LEDの明るさを変える調光のしくみ	116
H8/36064マイコンには5種類のタイマ・モジュールがある	117
▶ 8-2 タイマZの内部構造	117
チャンネル共通の主なレジスタ	118
各チャンネル用の主なレジスタ	120
▶ 8-3 タイマZによるPWMモードの動作	121
各種レジスタの設定	122
▶ 8-4 タイマZによるPWMモードでLEDの明るさを変えるC言語プログラムを作成する	122
プログラムの概要	123
リスト8-1のC言語プログラムを実行してLEDを光らせてみよう	124
▶ 8-5 LEDを「ホタルの光」のように変化させるC言語プログラムを作成する	125
プログラムの概要	125
▶ 8-6 ピュート ローバー用H8マイコン・ボード( VS-WRC003LV )の場合	127
<b>第9章 A-D変換モジュール機能を使って電圧を測定してみよう</b>	<b>129</b>
▶ 9-1 アナログからデジタルへの変換	130
A-D変換器のしくみ	131
逐次比較型A-D変換器の動作原理を天秤にたとえて考えてみよう	131
▶ 9-2 H8/36064マイコンのA-D変換器のブロック構成	133
A-D変換結果を格納するA-Dデータ・レジスタ( ADDR <sub>A</sub> ~ ADDR <sub>D</sub> )	134
A-D変換を操作する制御レジスタ	134

# 目次

H8/36064マイコンのA-D変換の動作	135
▶ 9-3 ブレッドボードを使ってA-D変換モジュール機能の動作確認をしてみよう	136
A-D変換モジュール機能の動作確認回路	136
A-D変換モジュールの各種レジスタの設定	137
▶ 9-4 A-D変換の動作確認用のプログラムを書いてみよう	137
プログラムの概要	138
▶ 9-5 モニタを使ってA-D変換モジュール機能の動作確認をしてみよう	139
ソース・リストを表示してブレーク・ポイントを設定する	139
A-Dモジュール関連のレジスタを表示する	139
メモリ内容を表示する	140
プログラムを実行してA-D変換する	140
◻Column 9-1 これだけは注意: A-D変換入力端子の電圧範囲	140
ステップ実行 (Pstep) してRAMに格納されたデータを確認する	141
▶ 9-6 A-D変換を使って光センサで圧電ブザーの音程を変えてみよう	142
<b>第10章 ビュート チェイサー“Beauto Chaser”を組み立てよう</b>	<b>143</b>
はじめに	144
ビュート チェイサーのステップアップ	144
▶ 10-1 ビュート チェイサーの梱包箱を開封してみよう	145
ビュート チェイサーを構成する主な部品を見てみよう	145
ダブル・ギヤボックスを構成する主な部品を見てみよう	146
▶ 10-2 ビュート チェイサーを組み立てる前に工具類を準備しよう	147
サイズに合ったドライバを準備する	147
ニッパーを準備する	147
▶ 10-3 ダブル・ギヤって何?	147
ダブル・ギヤボックスは高速タイプ(A)から低速タイプ(D)までの4種類の中から選んで組み立てる	148
高速/低速タイプを選択する上でのヒント	148
▶ 10-4 ギヤボックスの組み立て手順	149
まずはシャフト(車軸)から組み立てよう	149
シャフトにギヤを取り付ける	150
組み立てたシャフトを本体に固定する	150
DCモータをダブル・ギヤボックスに取りつけてからベースに固定する	151
▶ 10-5 パーツの組み立て	152
タイヤ、ボール・キャスタをベースに取り付ける	152
赤外線センサを前面に取り付ける	152

# 目次

ベースにH8マイコン・ボードを取り付ける	153
▶ 10-6 ビュート チェイサーの完成	153
ビュート チェイサーを動かしてみよう	154
▶ 10-7 ビュート ローバーの紹介	154
ビュート ローバーを構成する主な部品を見てみよう	156

## 第11章 Beauto Builder NEOで ライントレース・ロボットを動かしてみよう

▶ 11-1 ライントレース・ロボットはどのようにして線を認識しているの？	158
フォト・リフレクタと呼ばれる光センサを使う	158
光センサ(フォト・リフレクタ)の動作原理	158
光の量の変化を出力電圧の変化として検出するしくみ	159
◁Column▷ 11-1 光センサ(フォト・リフレクタ)の発光のようすを確認する方法	159
▶ 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	160
▶ 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	161
▶ 11-4 車輪の回転の制御はどのように行うの？	161
DCモータを駆動するモータ・ドライバIC(TB6552FN)	162
モータ・ドライバICの動作を理解する	162
▶ 11-5 Beauto Builder NEOでプログラムをつくってみよう	164
Beauto Builder NEOを起動する	164
アクション・ブロックをプログラム・エリアに配置する	165
アクション・ブロックの実行順序を矢印で接続する	166
左旋回前進と右旋回前進のアクション・ブロックの設定をする	166
分岐のアクション・ブロックの設定をする	167
▶ 11-6 プログラムをH8マイコンに書き込んでみよう	168
H8マイコン・ボードと通信を開始する	168
プログラムを書き込んで実行	169
▶ 11-7 ライントレース・ロボットの車輪(モータ)を調整しよう	169
◁Column▷ 11-2 Beauto Builder NEOの「テスト実行」方法	170
システム設定を変更する方法	171
▶ 11-8 光センサを調整しよう	171
▶ 11-9 まずはライントレース・ロボットを動かしてみよう	172
分岐と直進を増やしてプログラムを改善する	172
プログラム改善後の動作を確認してみよう	174

# 目次

第12章 ライトレース・ロボットの光センサを2個に増やして 動作を改善しよう	177
▶ 12-1 光センサを2個搭載したライトレース・ロボットの動作について考えてみよう	178
▶ 12-2 感度調整が容易にできる光センサ基板の回路設計	179
光センサ基板の回路設計	179
ラインの検出状態を知る工夫	181
ラインの検出状態を知らせる動作原理	181
▶ 12-3 光センサ基板の使用部品	182
(Column) 12-1 MOSTランジスタの種類	182
両面に銅箔があるタイプのユニバーサル基板を使用する	183
▶ 12-4 光センサ基板の部品配置と製作手順	183
(Column) 12-2 LEDに青/白色系を使用する場合の注意点	184
基板を切断・穴あけ加工する	185
(Column) 12-3 静電気に弱いCMOS-ICやMOSFETのはんだ付けには セラミック・ヒータ・タイプのはんだゴテがよい	185
▶ 12-5 ユニバーサル基板への部品実装の手順	187
(Column) 12-4 スルーホール基板などのはんだ付けには共晶はんだがよい	188
配線ケーブルを取り付けよう	189
製作した光センサ基板を車体に取り付けよう	191
▶ 12-6 光センサの受光感度の調整方法	191
最初に表示用LEDの点灯状態を見ながら受光感度の粗調整を行う	191
Beauto Builder NEOのセンサ数値を見ながら受光感度を微調整する	192
▶ 12-7 センサを2個使用したライトレース・ロボットのプログラムを作成する	192
▶ 12-8 ライトレース・ロボットを動かしてみよう	193
▶ 12-9 ビュート ローバーによるライトレース・ロボットを動かしてみよう	195
ビュート ローバー( Beauto Rover )の光センサの取り付け方	195
ビュート ローバー( Beauto Rover )の光センサの取り付け方による走行状態の違い	195
第13章 直角コースを曲がれるように ライトレース・ロボットの動作を改善しよう	197
▶ 13-1 直角コースを検出できるように光センサをさらに増設する	198

# 目次

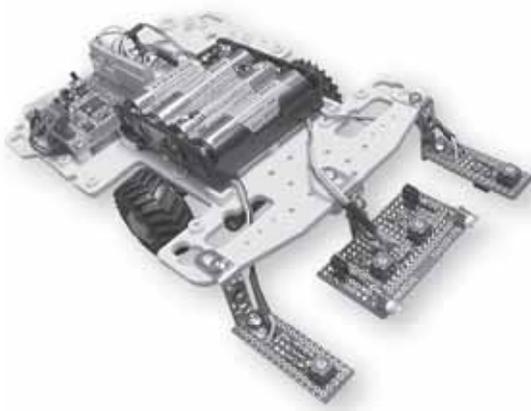
▶ 13-2 直角コース検出用の光センサ基板の回路設計	199
直角コース検出用の光センサ基板の使用部品	200
▶ 13-3 直角コース検出用の光センサ基板の製作	200
直角コース検出用の光センサ基板の部品配置	200
光センサの受光感度の調整方法	202
▶ 13-4 直角コースを曲がるライトレース・ロボットのプログラムを作成する	203
プログラムの説明	203
右左折時の少し前進(位置補正)と一時停止の役割について	204
▶ 13-5 ライトレース・ロボットを動かしてみよう	204
トラブルシューティング... 直角コースをうまく曲がれない場合の調整方法	204
▶ 13-6 直角コースを検出するピュート ローバーを動かしてみよう	206

## 第14章 C言語プログラムで

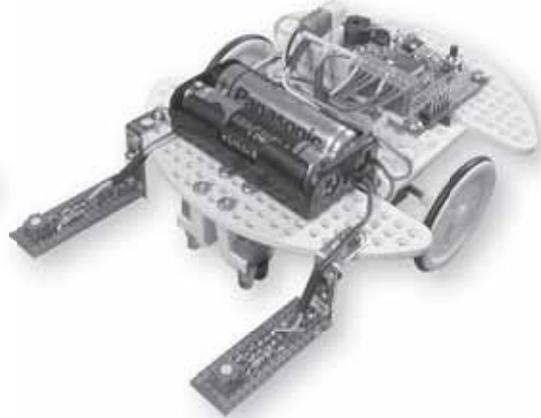
ライトレース・ロボットを動かしてみよう	209
▶ 14-1 C言語サンプル・ソースを入手する	210
プログラム作成に必要なファイルを解凍する	210
▶ 14-2 センサ1個搭載のライトレース・ロボットを制御するC言語プログラムの作成	210
新規プロジェクトワークスペースを作成する	211
関数定義ファイルをコピーする	211
C言語メイン・プログラムを作成する	211
割り込み関数ファイル“i ntprg. c”の改変	213
vs-wrc003関数ファイル“vs-wrc003. c”の改変	214
vs-wrc003関数ファイル“vs-wrc003. c”をビルドに追加する	214
ビルドコンフィグレーションを選択する	215
プログラムをビルドする	215
FDTを起動してプログラムをROMへ書き込む	216
ライトレース・ロボットを動かす	216
▶ 14-3 センサ1個搭載のライトレース・ロボットに直進動作を入れたC言語プログラム	216
▶ 14-4 センサ2個搭載のライトレース・ロボットを制御するC言語プログラムの作成	216
▶ 14-5 直角コースを曲がるライトレース・ロボットを制御するC言語プログラムの作成	217
▶ 14-6 ピュート ローバー用H8マイコン・ボード( VS-WRC003LV )の場合	218
H8マイコン・ボード( VS-WRC003LV )用のC言語サンプル・ソースを入手する	218
プログラム作成に必要なファイルを解凍する	220
割り込み関数ファイル“i ntprg. c”の改変	220
vs-wrc003l v関数ファイル“vs-wrc003l v. c”の改変	220
vs-wrc003l v関数ファイル“vs-wrc003l v. c”をビルドに追加する	220

# 目次

ビュート ローバー (Beauto Rover) を動かす	220
<b>第15章 プログラムの開発環境を導入しよう!</b>	<b>221</b>
▶ 15-1 USBシリアル・ドライバの導入方法(チェイサーのみ)	222
USBシリアル・ドライバの入手	222
USBシリアル・ドライバのインストール方法(Windows 2000/XP/Vista用の例)	222
H8マイコン・ボードの接続と認識(Windows 2000/XP/Vista用の例)	222
▶ 15-2 モニタとHtermを動作させる“VS-WRC003LVシリアルコンバータ”の導入と切り替え	224
シリアルコンバータのインストール	224
シリアル通信への切り替え方法	224
▶ 15-3 統合開発環境“HEW”の導入方法	225
無償評価版HEWの入手	225
無償評価版HEWのインストール方法	225
▶ 15-4 フラッシュ開発ツールキット“FDT”の導入方法(チェイサーのみ)	226
無償評価版FDTの入手	226
無償評価版FDTのインストール方法	226
▶ 15-5 VS-WRC003LVにプログラムを書き込むツール“H8 Writer”を導入(ローバー用)	226
H8 Writerによるプログラムの書き込み方法	227
▶ 15-6 モニタ・プログラムの導入方法	227
モニタ・プログラムの入手	227
モニタ・プログラムのインストール方法	228
モニタ・プログラムのカスタマイズ手順	228
モニタ・プログラムの書き込み手順	230
▶ 15-7 モニタ・プログラム専用通信ソフトHtermの導入方法	232
Htermの入手	232
Htermのインストール方法	233
Htermのプロパティ設定方法	233
▶ 15-8 ビュート チェイサー専用ソフトBeauto Builder NEOの導入方法	234
Beauto Builder NEOの入手	234
Beauto Builder NEOのインストール方法	235
▶ 15-9 ビュート ローバー専用ソフトBeauto Builder2の入手	235
索引	236
参考・引用*文献	238



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 製作したライトレース・ロボットの外觀

## 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*)\*<sup>1</sup>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、グイストン(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始めましょう。

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(Beuto 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な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)。

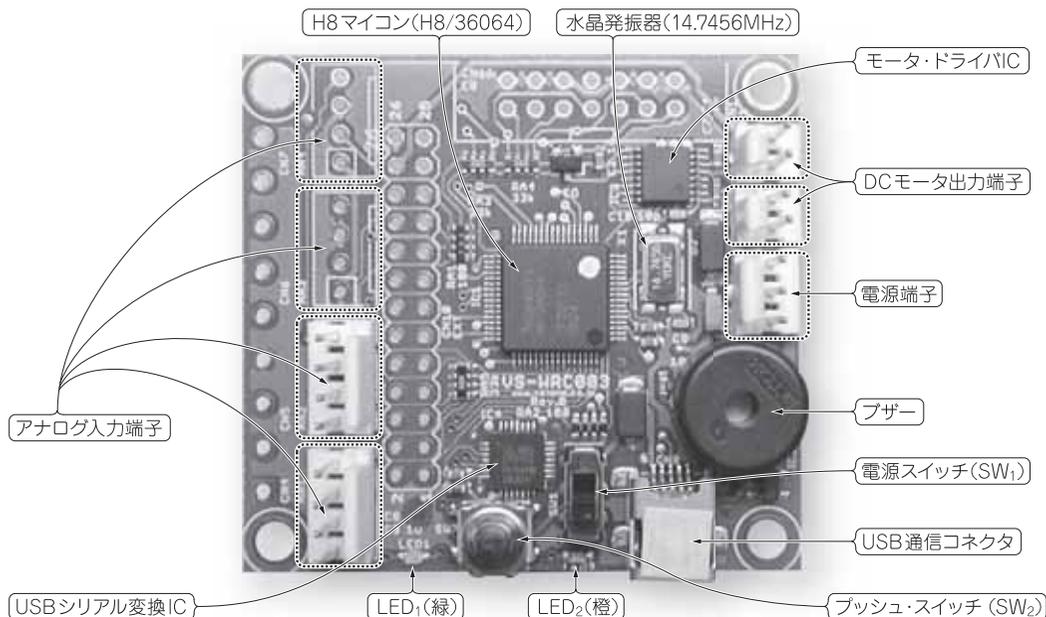


写真1-2 H8 マイコン・ボード VS-WRC003 の外観

## 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)、プザァー、DCモーター出力(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<sup>\*2</sup>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<sup>\*1</sup>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。

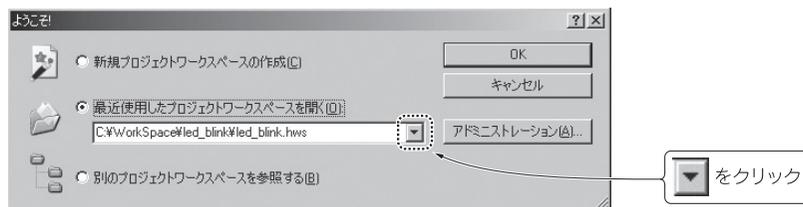


図4-1 最近使用したプロジェクトワークスペースを開く

\*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。

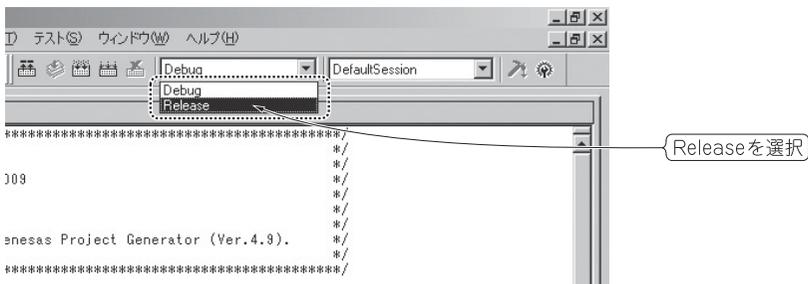


図4-2 ビルドコンフィグレーションで「Release」を選択

Address	Section	
0x00000400	PResetPRG	} ROM領域
	PIntPRG	
0x00000800	P	
	C	
	C\$DSEC	} RAM領域
	C\$BSEC	
	D	
	B	
0x0000FB80	R	} RAM領域
0x0000FE80	S	

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

Address	Section	
0x00000000	CV0	} ROM領域
0x00000400	PResetPRG	
0x00000800	P	
	C	
	C\$DSEC	} RAM領域
	C\$BSEC	
	D	
	B	
0x0000FB80	R	} RAM領域
0x0000FE80	S	

(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)

(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

(c) ROM化してスタンドアロン

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、まずツールチェーンを開きます。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。ツールチェー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。

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)かもしれませんが、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volatile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・キャピ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。

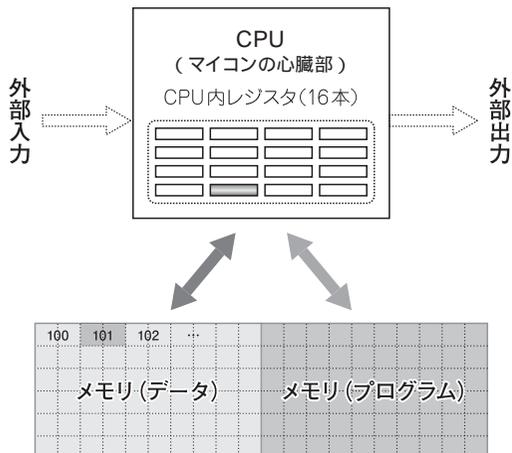


図5-6 データ処理のイメージ

( 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 )

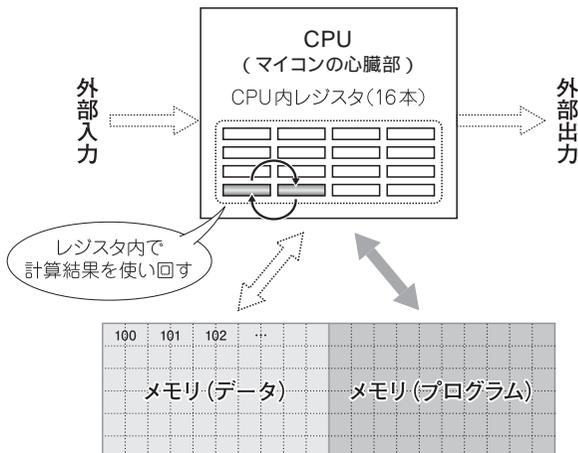


図5-7 最適化されたデータ処理のイメージ

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

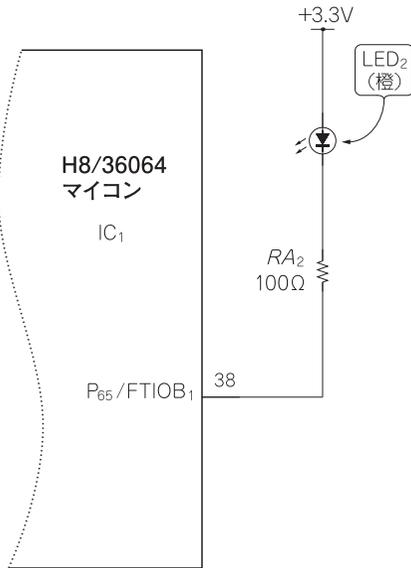


図8-1 LED<sub>2</sub>（橙）とH8マイコンとの接続  
ローバーの接続は図8-10を参照。

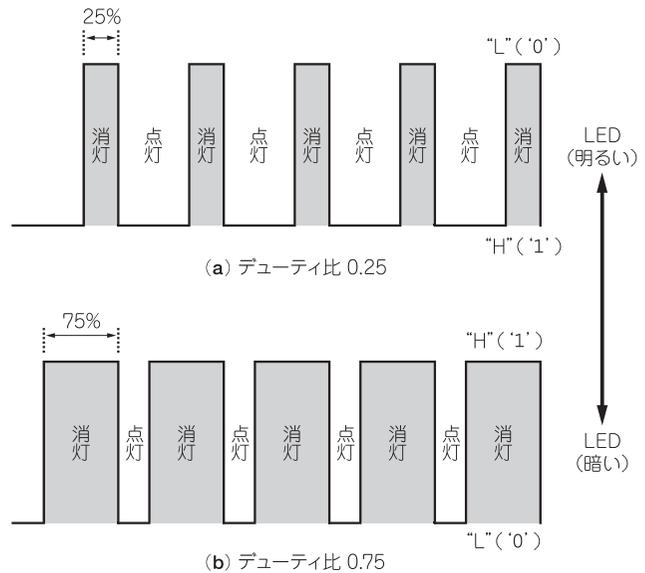


図8-2 PWM信号波形とLEDの点灯/消灯の関係

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<sub>65</sub>/FTIOB<sub>1</sub>）に接続されているLED<sub>2</sub>（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。

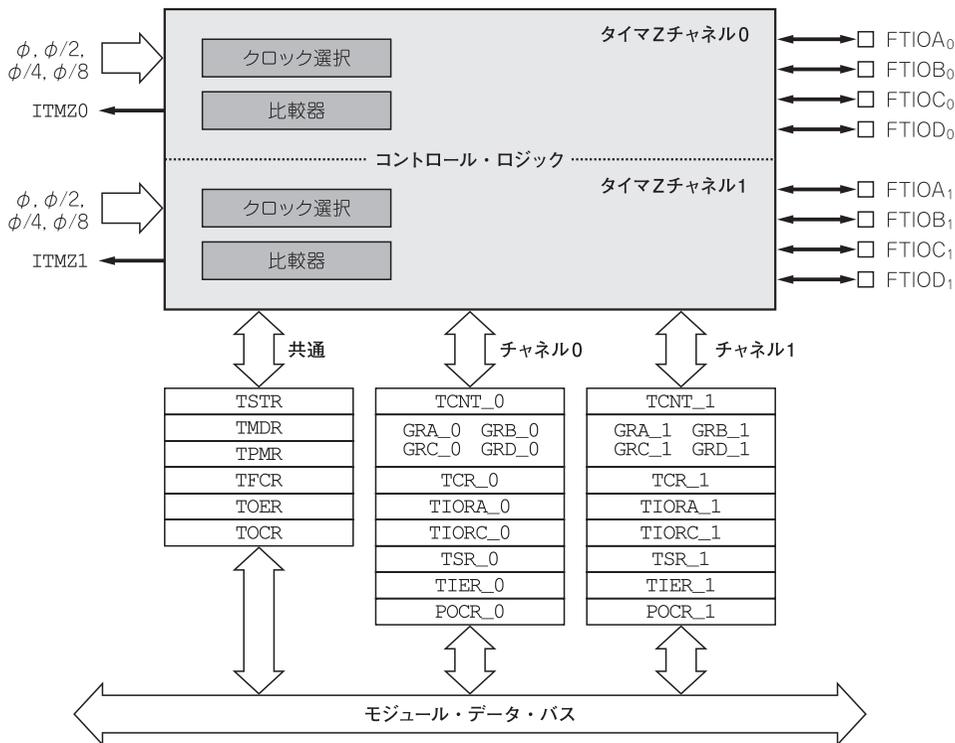


図8-3 タイマZの内部ブロック構造

## 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<sub>0</sub>～FTIOD<sub>0</sub>端子、FTIOA<sub>1</sub>～FTIOD<sub>1</sub>端子の合計8端子がタイマZの入出力端子です。LED<sub>2</sub>(橙)はFTIOB<sub>1</sub>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はチャンネル0とチャンネル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。

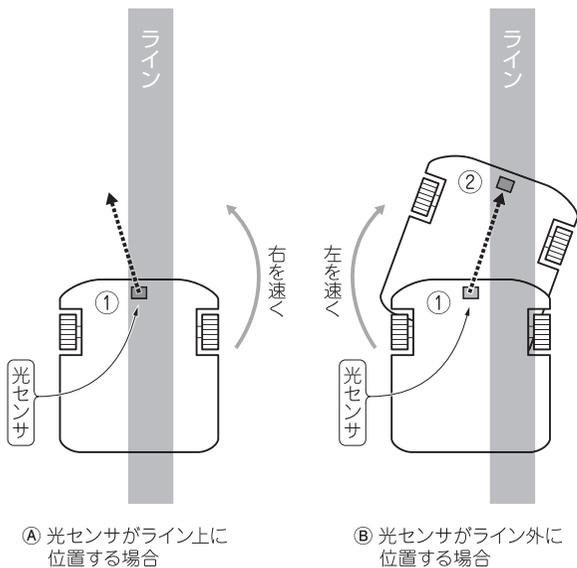


図11-3 ライントレース・ロボットの動作パターン

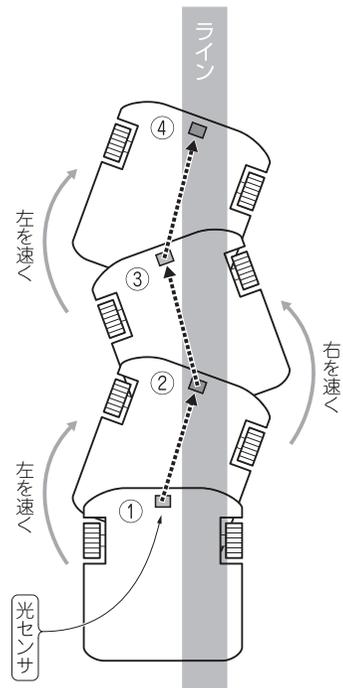


図11-4 ライントレース・ロボットがラインに沿って進むようすの説明図

## 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を使って、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。線の上にいると左寄りに進み、線の上が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。

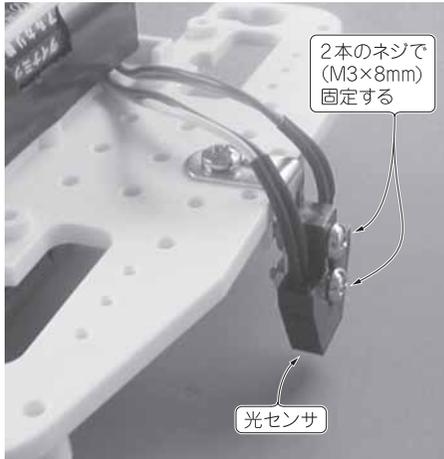


写真11-2 L字金具に光センサを取り付けたようす

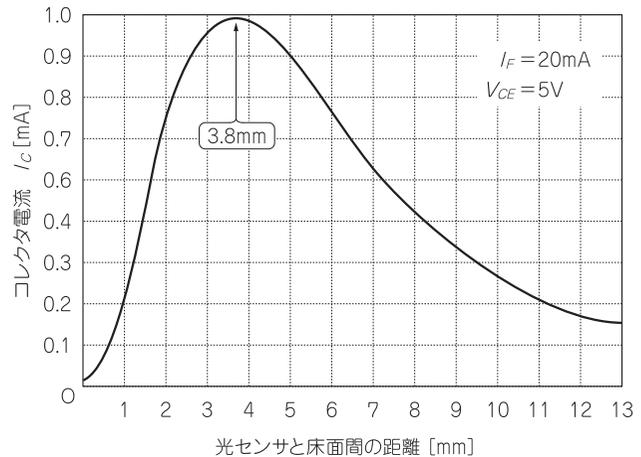


図11-5 光センサQRB1134の出力特性

### 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<sup>\*2</sup>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。