マイコンプログラミング

1. 目的

マイクロコンピュータ H8/3694 を用いて制御プログラムを製作する。プログラムの作製を通して、製作手順及び、プログラム製作方法を理解する。

2. 実験装置

マイクロコンピュータ(H8-3694)、オシロスコープ、本実験の実験 19 で製作した主基板

3. 実験方法・結果

本実験では、制御用マイコンとして H8/3694 を用いる。通常のマイコンでは ROM に機械語を書き込んで利用するが、このマイコンでは、機械語プログラムをフラッシュメモリに書き込むため、ROM が不要である。また、H8/3694 の持つ機能のうち、本実験では、以下のものを利用する。

(1) I/O
(2) シリアルコミュニケーションインタフェース
(3) タイマを用いた割り込み
(4) A/D 変換器
(5) タイマを用いた割り込み

使用法はおよそ以下のとおりである。

(1) WindowsPC のエディタを用いて、C 言語でプログラムを記述する。
(2) クロス C コンパイラ（C プログラムを別の CPU 上で実行する機械語プログラムに変換するコンパイラのこと）でコンパイルする。
(3) H8/3694CPU へシリアル通信で機械語プログラムを転送し、フラッシュメモリ（ROM に相当する）に書き込む。

フラッシュメモリに書き込まれた機械語プログラムは、電源を OFF にしても消えない。

3.1 時間待ち関数の作製（実験 20）

関数 msecWait を表 3.1.1 の仕様で作成した。C プログラム 20.c を動作させ、LED1 が接続されたポートの出力波形を観測することで、作製した関数の動作を検証した。LED1 が接続されたポート名はポート 8、ビット名は P86 である。また、この電圧波形を図 3.1.1 に示す。ただし、関数 msecWait の引数には、length=1 を入力している.

表 3.1.1 関数 msecWait の仕様

プロトタイプ void msecWait(unsigned int length)
機能 呼び出されてからlength[ms]経過後に呼び出し元に戻る。 関数内では時間を経過させる処理以外のことをしない。
引数 length：この関数が呼び出されてから 呼び出し元に戻るまでの時間。

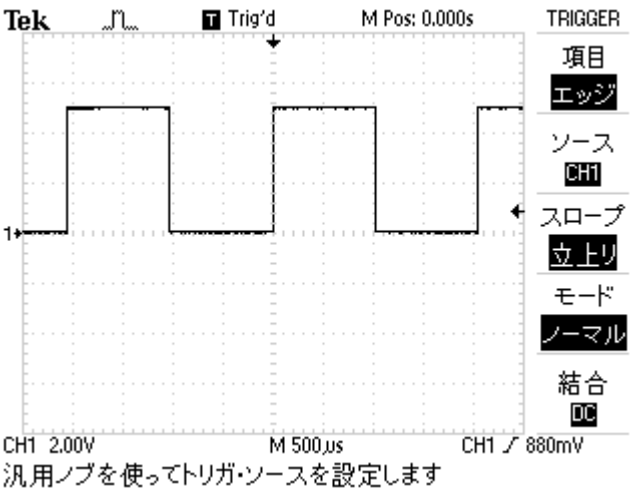

図 3.1.1　LED1 が接続されたポートの出力電圧波形

図 3.1 を見ると、出力された矩形波の周期は 2[ms]であることがわかる。動作させたプログラムでは、LED の点灯消灯の処理の間に関数 msecWait を 2 回呼び出しているため、2[ms]で動作していることは正しく、意図した動作ができていると読み取ることができる。関数 msecWait 内の for ループに用いている 568 という値は、関数内の処理にかかる時間が 1[ms]になるように、実測によって求めた。

3.2　LED の点滅とシリアル通信（実験 21）

2 個の動作確認用 LED を表 3.2.1 の組み合わせに繰り返して点灯または消灯させるプログラムの作製を行った。(21.c) 同時にその時の LED が接続された出力ポートのデータレジスタの値をシリアル通信によって出力させた。ただし、それぞれの組み合わせにおける動作の継続時間は 1 秒とした。

表 3.2.1　実験に使用した LED の点灯または消灯の組み合わせ

項番	LED1	LED2
1	OFF	OFF
2	ON	OFF
3	OFF	ON
4	ON	ON

シリアル通信によって出力させた実行結果は以下の通りである。以下の 4 つのデータを繰り返していた。

（1）00000000
（2）01000000
（3）10000000
（4）11000000

また、LED の点灯消灯の動作も正しく実行できていた。

3.3　タイマ割り込みを用いた LED の点灯（実験 22）

3 秒ごとにデューティ比を変化させながら、2 個の動作確認用 LED を以下の仕様で点灯させるプログラムを作製した。(22.c) また、表 3.3.1 の各項番において、動作確認用 LED が接続された 2 つのポートの出力電圧波形を、以下の条件で、同期をとって記録した。

(1)　タイマ A を用いて、周波数 9.766kHz で割り込みを発生させる。
(2)　割り込み回数を数え、その回数に応じて LED を点灯または消灯させることによって、デューティ比を調整する。
(3)　LED は 97.66Hz で点滅させる。
(4)　表 3.3.1 の各項番について、2 個の LED の点滅のデューティ比をシリアル通信によって出力する。

表 3.3.1 に示す条件 1-3 における出力電圧波形の図を図 3.3.1 から、図 3.3.3 に示す。ただし、波形上部は LED1 が接続されたポート（ポート 86）の出力電圧波形を、波形下部は LED2 が接続されたポート（ポート 87）の出力電圧波形を示している。

表 3.3.1　実験に使用した LED のデューティ比

項番	LED1	LED2
1	20%	90%
2	50%	50%
3	80%	10%

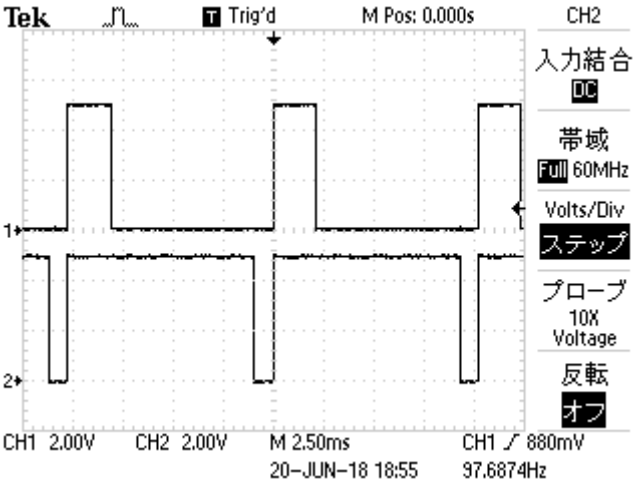

図 3.3.1　項番 1 の時の出力電圧波形

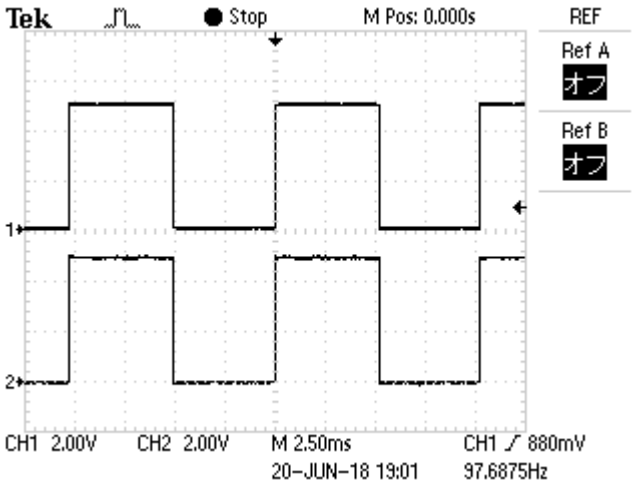

図 3.3.2　項番 2 の時の出力電圧波形

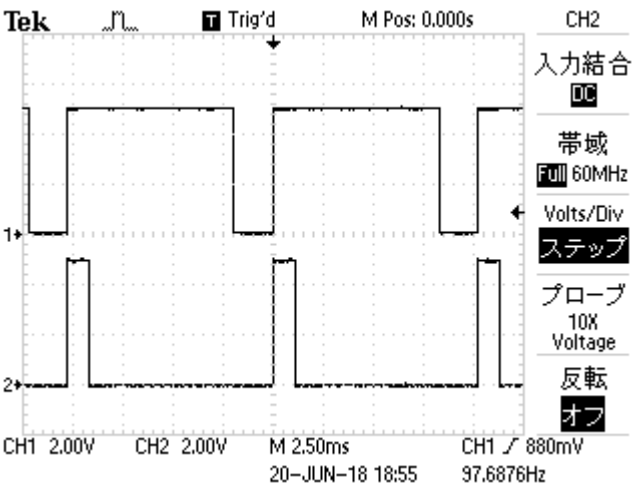

図 3.3.3　項番 3 の時の出力電圧波形

全ての波形において、デューティ比の出力が正しくできている。また、全ての波形で周期が約 10[ms]であり、周期は約 100[Hz]と、条件に合った LED の周期である 97.66[Hz]で点滅している。

シリアル通信の出力には、通信モードの初期化とデータ送信の準備をする関数(initSCI3)、指定された書式で文字列を作成し送信する関数(SCI3printf)を用いている。

シリアル通信の出力結果（TeraTerm の出力結果）は以下の通りであり、LED の点灯パターンと対応して繰り返していた。

duty1: 20[%] duty2 : 90[%]
duty1: 50[%] duty2 : 50[%]
duty1: 80[%] duty2 : 10[%]

以上より、プログラム作製時に意図した動作と同じ動作をしており、正しい動作をしていることが分かる。

4. 考察

4.1　プログラムで用いた点灯及び消灯処理について

実験 21 で作製したプログラム（21.c）では、LED の点灯及び消灯状態を細かく制御している。

ここでは、制御の具体的な方法を述べる。

動作確認用 LED には P86 と P87 を用いているため、ポート 8 の 6 ビット目および 7 ビット目を 1/0 に変化させることで、LED1 及び LED2 の状態を変化させることができる。

また、プログラム内の、IO.PDR8.BYTE は、現在のポート 8 の状態を示している。

点灯消灯の処理をする前に IO.PCR8|=0xC0;で、ポートの初期化（P86,P87）をしている。これによって、具体的なアドレスを意識することなく、プログラム上で各レジスタにアクセスできるようになる。

（1）　点灯処理（IO.PDR8.BYTE |=0x??;）

LED の現在の状態に対して、OR の処理をすることで点灯処理を行っている。点灯させたい LED のビットのみを 1 にして現在の状態と OR 演算すれば良い。例えば、P87 を点灯させたい場合、7bit 目を 1 にする。すなわち 10000000（0x80）と現在の状態を OR 演算する。（IO.PDR8.BYTE |=0x80;）

よって、P86 のみを点灯させたい場合、01000000(0x04)と現在の状態を OR 演算すればよい。(IO.PDR8.BYTE |=0x40;)また、両方の LED を点灯させたいときは 11000000(0xC0)と現在の状態を OR 演算すればよい。(IO.PDR8.BYTE |=0xC0;)

（2）　消灯処理（IO.PDR8.BYTE &=0x??;）

LED の現在の状態に対して、AND の処理をすることで消灯処理を行っている。消灯させたい LED のビットのみを 0 にして現在の状態と AND 演算すれば良い。例えば、P87 を消灯させたい場合、7bit 目を 0 にする。すなわち 01111111(0x7F)と現在の状態を AND 演算する。（IO.PDR8.BYTE &=0x7F;）

よって、P86 のみを消灯させたい場合、10111111(0xBF)と現在の状態を AND 演算すればよい。(IO.PDR8.BYTE &=0xBF;) また、両方の LED を点灯させたいときは、00111111(0x3F)と　　現在の状態を OR 演算すればよい。(IO.PDR8.BYTE |=0x3F;)

本実験で作製したプログラムは全て、バイトアクセスをして制御しているが、IO.PDR8.BYTE.B?のように、ビットアクセスをして、制御する方法もある。ビットアクセスをすると、バイトアクセスと比較すると、直感的でわかり易いプログラムになる。また、バイトアクセスは、複数処理を簡潔に書くことができる。

4.2　タイマ割り込みについて

実験 22 で作製したプログラム（22.c）では、タイマ割り込みを用いて LED の状態を制御している。タイマ割り込みとは、CPU のタイマユニットにより、一定の間隔で割り込み要求信号を発生させ、割り込み関数を実行することである。また、タイマ割り込み関数は、実験 20 で作製した関数 msecWait を用いた時間管理より、はるかに正確な時間管理ができる。（ただし、CPU のクロックの精度に依存する。）

本実験で用いたマイクロコンピュータ H8/3694 には、タイマ A、タイマ V、タイマ W の 3 つのタイマ機能がある。タイマ A はインターバルタイマ及び、時計用タイムベース機能を内蔵した 8bit のタイマである。設定されたクロックによって、8bit のアップカウンタであるタイマカウンタ（TCA）がカウントアップされ、オーバーフローした時に割り込み要求を発生させる。また、オーバーフローすると再びカウントアップされる。したがって、265 回のクロック入力につき 1 回、すなわち、クロックの 256 分の 1 の周波数で割り込み要求を発生させることができる。

(1)　タイマ A の入力クロックについて

TCA をカウントするクロックは、タイマモードレジスタ A(TMA)によって設定する。TMA をプログラム内で設定するには、TA.TMA.BYTE の値（入力クロック）を設定する。H8/3694 ハードウェアマニュアルの 10.3.1　タイマモードレジスタ A（TMA）を見ると、具体的な設定値が分かる。

実験 22 では、タイマ A を用いて周波数 9.766[kHz]で割り込みを発生させた。この条件で、タイマ A の入力クロック数を求める。TCA の入力クロックは、システムクロックの周波数は 20[MHz]より、

$$\frac{20 \times 10^6}{x} = 9.766 \times 10^3 \times 256$$

$$x = 8$$

よって、TCA の入力クロックは $\phi/8$ である。H8/3694 ハードウェアマニュアルより、入力クロック数が $\phi/8$ のときの TA.TMA.BYTE の値は 111、すなわち 0x07 にすればよい。(TA.TMA.BYTE |= 0x07;)

(2)　タイマ A で割り込み要求を発生させるための設定について

タイマ A で割り込み要求を発生させるためには、TMA によってタイマ A の動作を設定するだけでなく、CPU 内部のレジスタおよび、割り込みを制御するレジスタ及びビットの設定をする必要がある。TMA に加えて以下の 3 つのレジスタ、ビットを変更しなければならない。（記号―ビット）

①　CCR－I （コンディションコードレジスタ）

割り込み要求の発生のために、CPU 内部にある CCR の I ビットをクリアする（割り込みマスクのクリアを行う）必要がある。これは C 言語の構文絵はアクセスできない。そのためアセンブリ言語で行う必要があり、スタートアップルーチンで記述するか、C 言語のプログラムの途中で部分的にアセンブリ言語を記述すること（インラインアセンブラ）によって実現する。

本実験ではインラインアセンブラを利用して、割り込みマスクのクリアをする。次のようにすると、C 言語のプログラムの中にアセンブリ言語を記述することができる。

Asm(“*code*”);

また、次のように記述することで、割り込みマスクのクリアが行える。

Andc.b #0x7f,CCR

製作したプログラムでは、次のように記述し、マクロを定義している。

#define clearCCR_I() asm volatile("andc.b #0x7f,CCR")

volitle 修飾子は、プログラムをコンパイルする際のコンパイルによる最適化を抑止し、最適化によって、意図した位置以外で、インラインアセンブラで記述したコードが実行されることを抑止させている。

② IENR – IENTA（割り込みイネーブルレジスタ）

割り込み要求を可能にするために、割り込みイネーブルレジスタ 1 の値を変更する必要がある。割り込みイネーブルレジスタのビットアクセスは次のように記述することで行える。1 を代入することで、割り込み要求を可能にしている。

IENR1.BIT.IENTA=1;

③ IRR1 – IRRTA（割り込みフラグレジスタ）

TCA がオーバーフローした時にも割り込み要求を発生させないといけない。よって、割り込み関数内で、割り込みフラグレジスタの値をクリアする必要がある。割り込みフラグレジスタのビットアクセスは次のように記述することで行える。0 を代入することで、割り込みフラグレジスタの値をクリアしている。

IRR1.BIT.IRRTA=0;

（3） 割り込み関数の記述方法について

C 言語で割り込み関数を記述する場合は、関数名の前にプリプロセッサ指令である

#pragma interrupt

を記述する。これによって、汎用レジスタの退避や復帰など、割り込みで必要な処理が実行されるようになる。

また、割り込み関数では、シリアル通信などの長い時間を要する処理をしてはならない。割り込み関数内でのデータ転送速度は決まっているため、データ転送に使う時間がかかり、CCPU に大きな負荷がかかるためである。

5. 参考文献

H8/3694 ハードウェアマニュアル