To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

H8/300L

Driving of Stepper Motor (Stepper)

Introduction

A stepper motor translates current pulses into motor rotation. A typical stepper motor contains 4 winding coils. Applying voltage to these coils forces the motor to rotate step by step.

In normal operation, two winding coils are activated at the same time. The stepper motor moves clockwise one step per change in winding activated. If the sequence is applied in reverse order, the motor will run counterclockwise.

The speed of rotation is controlled by the frequency of the pulses. Every time a pulse is applied to the stepper motor the motor will rotate a fixed distance. A typical step rotation is 18 degrees. With 18 degrees rotation in each step will complete one rotation of the motor (360 degrees) require 20 steps. By changing the interval time, the speed of the motor can be regulated, and by counting the number of steps, the rotation angle can be controlled.

I/O port 5 [bit 3 ..bit 0] of H8/38024 MCU is used to drive the 2 phase / 4 phase stepper motor driver circuit. Two types stepper motor driver circuit will be introduce here, first is stepper motor driver IC (L298N) and second is power MOSFET transistor (2SK1095).

This application note demonstrated the use of H8/38024F SLP MCU in driving a 2 phase / 4 phase stepper motor. Two types of hardware driver circuits are discussed.

Target Device

H8/300L Super Low Power – H8/38024 Series

Contents

1. HARDWARE OVERVIEW 3
1.1 Hardware Circuit Option 1: 3
1.2 Hardware Circuit Option 2: 7

2. Software Overview 9
2.1 Software description for Example source code 1 9
2.2 Software description for example source code with timer interrupt 14

3. Other Consideration 14

Reference 14

1. HARDWARE OVERVIEW

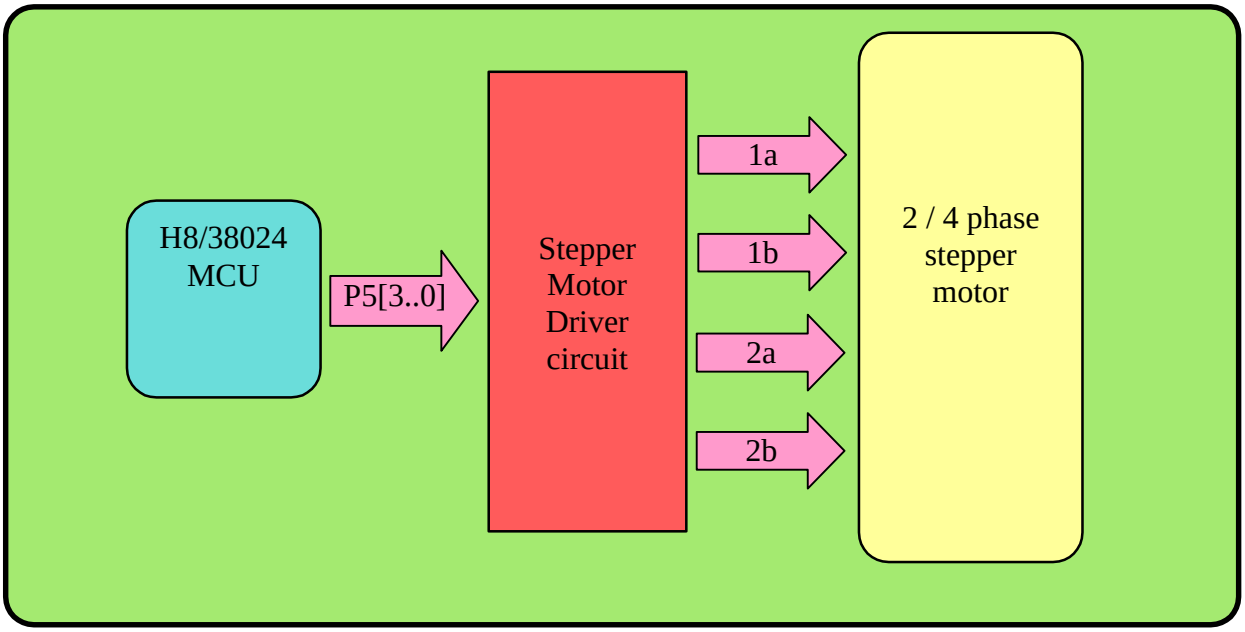

Figure 1 Hardware Block Diagram

Basically, there are 3 components in the stepper motor control circuit:
Micro-controller - used to generate the stepper motor control waveforms
Motor Driver - used to drive the stepper motor
Stepper Motor - used to evaluate the control waveform from MCU

1.1 Hardware Circuit Option 1:

In hardware design, the most complicated part is stepper motor driver circuit design. This motor driver design depends on the stepper motor characteristic. The most common way to design a stepper motor driver is using a “Stepper Motor Driver IC”. The stepper motor that I choose is SAIA-UAG2 which can connect 2-phase or 4-phase. Table 1 and table 2 show the technical data of two different stepper motor which can be used in this application note.

The L298N stepper motor driver circuit diagram (2-phase) is show by figure 2 and the R1 and R2 value depend from the load current. The protection diodes also can be reduce by changing the IC L298N to L293D.

Table 1 Technical data of SAIA-UAG2 stepper motor:

Item	Value		
Step per resolution	20		
Winding type	Unipolar		
Rated Voltage	6 volt	12 volt	24 volt
Resistance per winding	35 Ω	170 Ω	700 Ω
Maximum Power Consumption		0.4 Watt	
Winding temperature		130°C	
Duty Cycle		100%	
Holding torque		0.5 cNm	
Detent torque		0.14 cNm	
Direction of rotation		Reversible	
Rotor inertia		0.31 gcm^2	

Table 2 Technical data of Step-Syn (type:103H546-0440) stepping motor:

Item	Value
Step per resolution	200
Rated Voltage	3.15 volt
Resistance per winding	3.15 Ω
Current per phase	1 Ampare
Holding torque	1.5 kg.cm
Direction of rotation	Reversible
Rotor inertia	31 gcm^2
Weight	0.2 kg

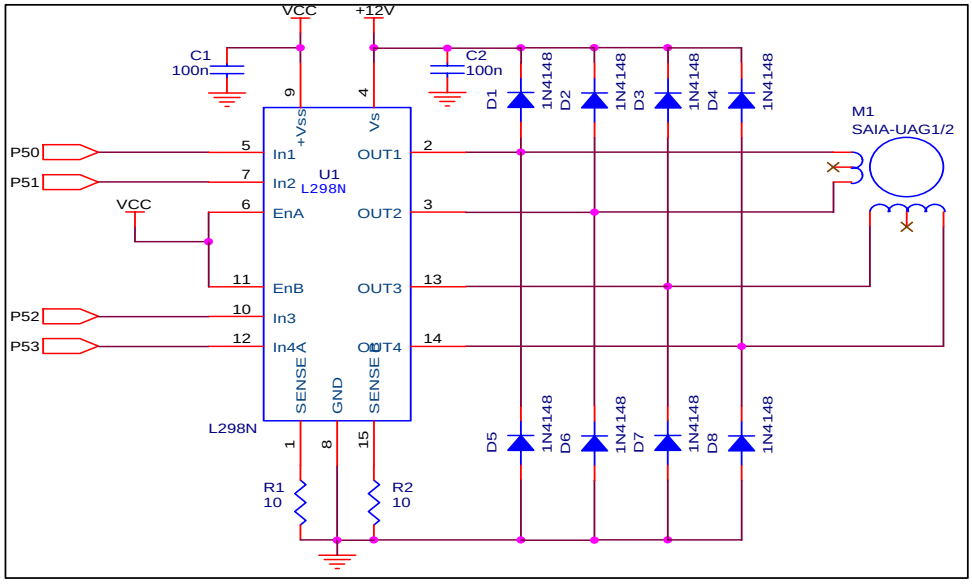

Figure 2 L298N Stepper Motor Driver Circuit(2-phase)

Table 3: Full-step sequencing

Step	P53	P52	P51	P50	PDR5
0	1	0	0	1	0x09
1	1	0	1	0	0x0A
2	0	1	1	0	0x06
3	0	1	0	1	0x05

Table 4: Half-step sequencing

Step	P53	P52	P51	P50	PDR5
0	1	0	0	0	0x08
1	1	0	1	0	0x0A
2	0	0	1	0	0x02
3	0	1	1	0	0x06
4	0	1	0	0	0x04
5	0	1	0	1	0x05
6	0	0	0	1	0x01
7	1	0	0	1	0x09

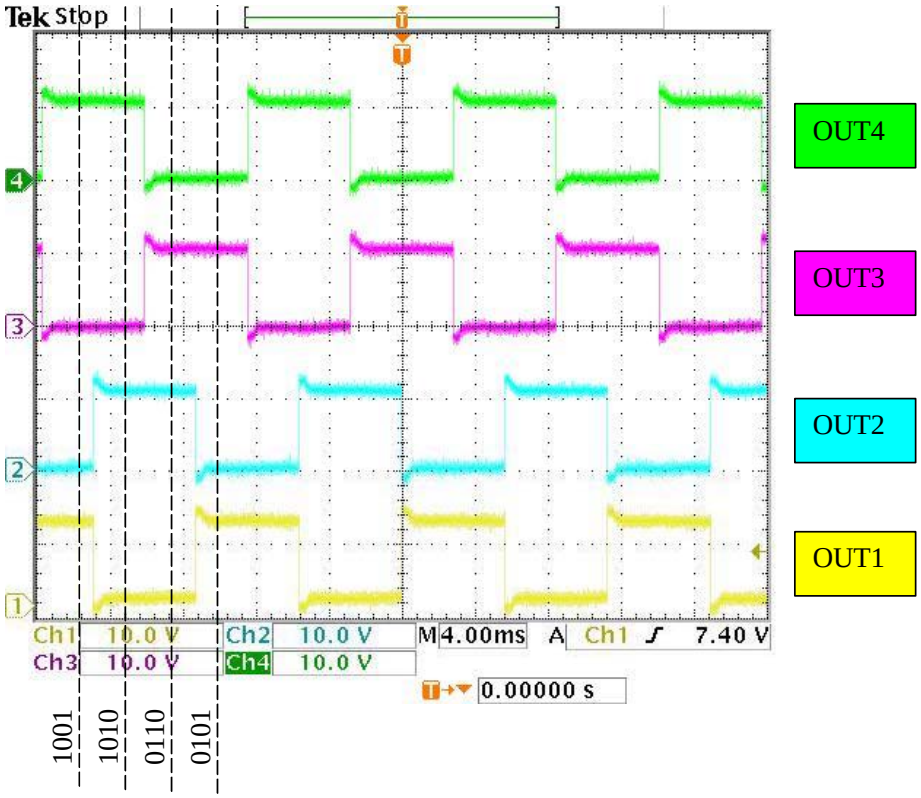

Figure 3 Waveform for Hardware circuit option 1 with example program 1

1.2 Hardware Circuit Option 2:

Besides using the stepper motor driver IC, stepper motor can be drive by 4 N-channel MOSFET 2SK1095. Figure 4 shown the MOSFET stepper motor driver circuit(4-phase). The stepper motor control waveform are driven through AND gates.

The signals are input to the gates of 4 N-Ch. MOSFET transistors (type 2SK1095), alternately switching them on and off. These transistors act as a drain for the current generated across the stepper motor coils. Four protection diodes (type HRP-22) are used to keep the voltage drop between the drain and the source to 0.55V. Motor make a lot of electrical noise so C1 and C2 is used to suppress the noise spikes.

Finally, power is supplied to the motor windings from a +12V* power supply through a pair of resistor to limit the coils current. Since the MOSFET's switching time is extremely fast, therefore a dead-time is needed between inverting phase signals (A and A-, B and B-) so that A- and B- signal will be slightly reduce before A and B are turned on. User can also increase the dead-time delay to achieve Half-step sequent. Note that R6, R7, R8 and R9 are pulled down, if MCU is trigger to standby mode, I/O port is in high impedance thus all phase will be in inactive mode.

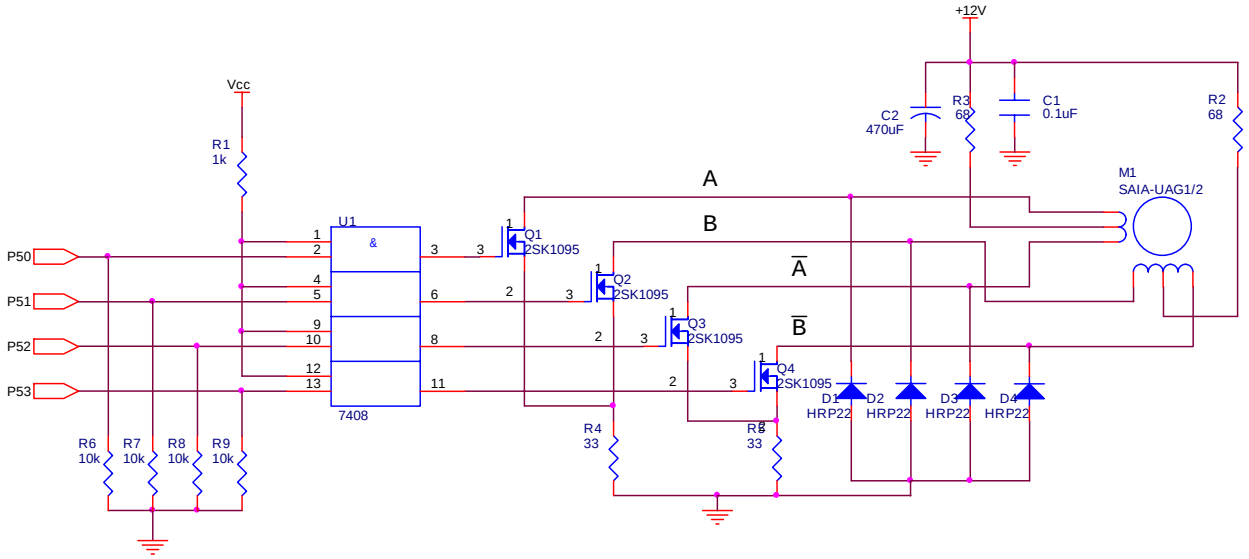

Figure 4 MOSFET Stepper Motor Driver Circuit

* User may change the +12V supply and R2, R3, R4, R5 value to suit different stepper motor. For example: Power supply = 3 volt, R2=R3=R4=R5=1Ω when use *Step-syn* stepper motor to achieve rated holding torque.

Table 5: Full-step sequencing

Step	P53	P52	P51	P50	PDR5
0	1	0	0	1	0x09
1	1	0	1	0	0x0A
2	0	1	1	0	0x06
3	0	1	0	1	0x05

Table 6: Dead-time sequencing

Step	P53	P52	P51	P50	PDR5
0	0	0	0	1	0x01
1	1	0	0	0	0x08
2	0	1	0	0	0x02
3	0	0	1	0	0x04

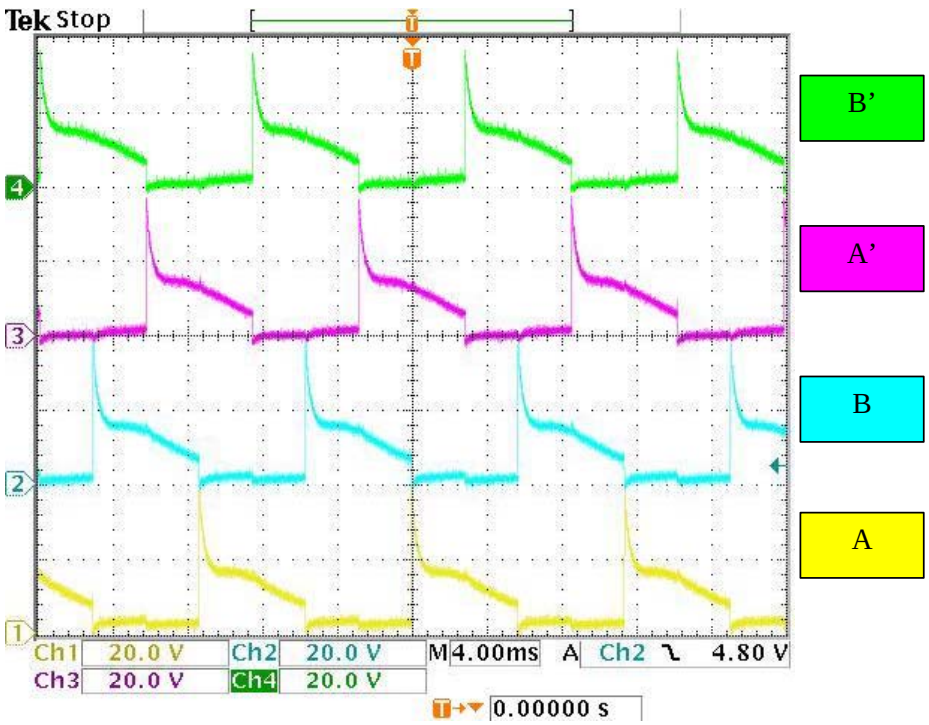

Figure 5 Waveform for Hardware circuit option 2 with example program 2

2. Software Overview

Once the desired port pins on the MCU are connected to the input pins on the stepper motor driver IC, the stepping sequence is easily implemented into software. Any I/O port on the MCU can be used to provide the input signal to the driver IC. In this example, the lower 4 bit of Port 5 is connected to the input pin of the driver IC (Hardware Circuit Option 1). To energize a winding the programmer set the “In1” pin high (binary 1) and the “In2” pin low (binary 0). The following code in figure 6 gives an example on how programmer can cause the stepper motor to continuously rotate, using the Full Stepping sequence as described in Table 3.

```
#include "iodefine.h"
#include <machine.h>

unsigned int loop_count, delay, max_step;
//an array of the positions used in the Full Stepping sequence
unsigned char motor_data[4] = {0x09,0x0a,0x06,0x05};
void main(void)
{
   P_IO.PCR5.BYTE = 0xFF; //set port 5 as output port
   P_IO.PDR5.BYTE = 0x00; //set port 5 data as 0000 0000
   loop_count = 0; max_step = 4; //Initialises the variable

   while(1)
   {
   //output stepping sequence
   P_IO.PDR5.BYTE = motor_data[loop_count++];
   //Coil energizing time
   for (delay=0 ; delay<0x300; delay++) ;
   //return to beginning
   if (loop_count==max_step) loop_count = 0;
   }
}
```

Figure 6 Example source code 1 for stepper motor control

2.1 Software description for Example source code 1

To use a different stepping sequence, simply change the motor_data and max_step value, e.g. for the Half stepping sequence, max_steps should be changed to 8 and motor_data[4] change to motor_data[8]={0x08, 0x0a, …,0x01,0x09}; (follow the Table 5).

As the MCU executes each command within microseconds, a delay need to be inserted in between each step, otherwise the rotor will not be able to rotate as the winding is not fully energise. This has been implemented with the line

for (delay=0 ; delay<0x300; delay++) ;

where the value of delay affects the speed and torque of the motor. If the value of delay were increased, it would cause the winding to be energised for longer which would increases the torque, but because the winding is being energised for longer the speed of the motor is reduced. If the value of delay is decreased, the speed of the motor is increased however this causes the torque to be reduced and the motor may not be able to rotate or be in control.

Thus, for Hardware circuit option 2, a dead-time sequence need to be insert before each stepping sequence. The example source code (shown in Figure 6) has to be modified to suit the hardware circuit option 2. The following source code in Figure 7 show the modification of the stepper motor control source code.

```
#include "iodefine.h"
#include <machine.h>

unsigned int loop_count, delay, max_step;
//an array of the positions used in the Full Stepping sequence
unsigned char motor_data[8] = {0x01, //Dead-time sequence 0
                               0x09, //stepping sequence 0
                                  0x08, //Dead-time sequence 1
                                  0x0A, //stepping sequence 1
                                  0x02, //Dead-time sequence 2
                                  0x06, //stepping sequence 2
                                  0x04, //Dead-time sequence 3
                                  0x05  //stepping sequence 3
                                  };
void main(void)
{
   P_IO.PCR5.BYTE = 0xFF;        //set port 5 as output port
   P_IO.PDR5.BYTE = 0x00;        //set port 5 data as 0000 0000
   loop_count = 0; max_step = 8; //Initialises the variable
   while(1)
   {
      P_IO.PDR5.BYTE = motor_data[loop_count++]; //output deadtime sequence
   for (delay=0 ; delay<0x10; delay++) ;      //small delay
      P_IO.PDR5.BYTE = motor_data[loop_count++]; //output stepping sequence
   for (delay=0 ; delay<0x300; delay++) ;     //Coil energizing time
      if (loop_count==max_step) loop_count = 0;
    //return to beginning
   }

}
```

Figure 7 Example source code 2 for stepper motor control

The example source code 1 and 2 above demonstrates a simple method of implementing the stepper motor control. A more efficient method to reduce the MCU load, would be to make use of the “Timer Interrupt” to generate the stepper motor control stepping sequence. Let’s modify the example source code 2 to become a timer interrupt driven method. In this example, Timer F is choose to generate the stepper motor control waveform.

There are 2 part of programming when timer interrupt is using:

a. Main program - void main(void)

b. Interrupt service routine - __interrupt(vect=15) void INT_TimerFH(void)

*Please refer to figure 8 and figure 9 for the detail source code.

```
#include "iodefine.h"
#include <machine.h>
void init_stepper_motor_io(void);
void init_timer_F(void);

unsigned int loop_count, delay, max_step;
//an array of the positions used in the Full Stepping sequence
unsigned char motor_data[8] = {0x01, //Dead-time sequence 0
                               0x09, //stepping sequence 0
                               0x08, //Dead-time sequence 1
                              0x0A, //stepping sequence 1
                              0x02, //Dead-time sequence 2
                              0x06, //stepping sequence 2
                              0x04, //Dead-time sequence 3
                              0x05  //stepping sequence 3
                             };
void main(void)
{
   init_stepper_motor_io(); //initialise I/O port
      init_timer_F();          //initialise Timer F with interrupt
   while(1)
   {
   //user program start here
   }
}
void init_stepper_motor_io(void)
{
   P_IO.PCR5.BYTE = 0xFF;
      //set port 5 as output port

   P_IO.PDR5.BYTE = 0x00;
   //set port 5 data as 0000 0000
}
```

```
void init_timer_F(void)
{
   set_imask_ccr(1);         //disable interrupt request
   P_TMRF.TCRF.BYTE = 0x04;      //set timer F 16bit mode, counting on TCFL
                                    //overflow signal
   P_TMRF.TCSRF.BIT.CCLRH = 1;   //TCF clearing by compare match


   P_TMRF.OCRF.BYTE.H = 0x00;
      P_TMRF.OCRF.BYTE.L =0x30;     //set output compare register value

   P_SYSCR.IENR2.BIT.IENTFH = 1; //Enable Timer F interrupt
   P_SYSCR.IRR2.BIT.IRRTFH = 0;  //clear Timer F interrupt request flag
   set_imask_ccr(0);             //enable interrupt request
}
```

Figure 8 Main program for stepper motor control (with timer interrupt)

```
#include "iodefine.h"
#include <machine.h>
extern unsigned char motor_data[8];
extern unsigned char loop_count;
#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 IRQ0
__interrupt(vect=4) void INT_IRQ0(void) { /* sleep(); */ }
.
.
.
.

__interrupt(vect=14) void INT_TimerFL(void) {/* sleep(); */}
// vector 15 Timer FH Overflow
__interrupt(vect=15) void INT_TimerFH(void)
{
   if (P_TMRF.TCSRF.BIT.CMFH == 1)
   {
       P_TMRF.TCSRF.BIT.CMFH = 0;
          if (loop_count%2 == 0) //DEAD-TIME_DATA
       {
                P_IO.PDR5.BYTE = motor_data[loop_count++];
                P_TMRF.OCRF.BYTE.H = 0x00;
                P_TMRF.OCRF.BYTE.L = 0x30;
       }
       else
```

```
        {
             P_IO.PDR5.BYTE = motor_data[loop_count++];
             P_TMRF.OCRF.BYTE.H = 0x03;
             P_TMRF.OCRF.BYTE.L = 0x00;
        }

        if (loop_count==8) loop_count = 0;
   }

P_SYSCR.IRR2.BIT.IRRTFH= 0;
}
// vector 16 Timer G Overflow
__interrupt(vect=16) void INT_TimerG(void) {/* sleep(); */}
// vector 17 Reserved
```

Figure 9 Interrupt program for stepper motor control (with timer interrupt)

2.2 Software description for example source code with timer interrupt

First, the MCU will initialize the stepper motor I/O port as output port with value 0x00. Then Timer F is initialized as 16-bit free-running counter with timer clock is set to ∅/32. The Timer F counter value (TCF) will clear to 0x0000 when a compare-match between OCRF and TCF occur. Finally the timer F interrupt is enable to allow the output compare-match interrupt request from Timer F.

The process start by loading the startup count into the OCRF during the timer initialization code and Timer F interrupt service routine will do the rest of the job. The Interrupt routine is serviced when the contents of OCRF match the contents TCF. Then, the dead-time sequence data is output to Port 5, the OCRF is update for dead-time duration. After that, when OCRF match with TCF again, the second time timer interrupt request occur, this time the stepper motor sequence data is output to port 5 and OCRF is update again for dead-time duration.

Interrupt routine work as below:

a. When timer F interrupt request occur, the interrupt service routine will be executed.

b. Then check for compare-match flag for '1' if yes continue the rest of the process

c. Clear CMFH flag

d. Check for loop_count value, if EVEN number then output dead-time sequence data and then setup OCRF value for dead-time duration.

e. If the loop_count value is ODD then stepping data will output to port 5 and update the OCRF for coil energised duration.

f. Reset the loop_count when all the 8 sequence data was output to port 5.

g. Clear interrupt request flag to '0'.

3. Other Consideration

The driving of stepper motor can be further enhancement, depending on the need of the application. For example, the application may need to drive the stepper motor to make numerous turns in a quickest possible time. In order to kick-start the motor, the initial delay may be longer as higher torque is required to turn the motor load. This delay can be slowly reduced due to the motor rotating inertia. In this manner, motor can be rotate in a faster speed. These delay data can be calculate based on the motor profile, load and etc.

The stepper motor used in this AN is a general low power stepper motor (400mW max), user may need to changes the driver circuit to suit the higher power stepper motor.

Reference

1. H8/38024 Series, H8/38024F-ZTAT Hardware Manual (ADE-602-231A)

2. H8/300 Using a H8/300 to control a stepper motor Application Note (AE-0057)

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep.03	-	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.