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 SLP Series

Precise Control of DC Motor (DCmotor)

Introduction

This application note demonstrates how to use the H8/38024 SLP series to control a brush-DC servomotor. Advantages of using the H8/38024 SLP MCU are the many built-in peripherals such as the PWM, Serial Communication Interface (SCI), Timers, etc. This servomotor system can be used as the position controller in a printer, plotter or scanner.

Target Device

H8/38024

Contents

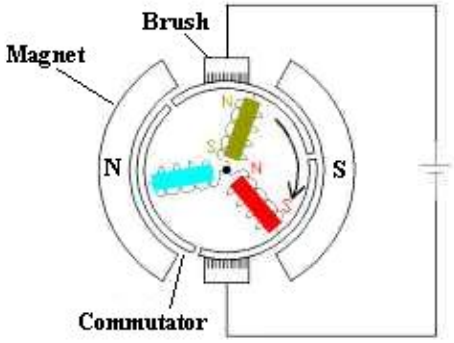
1. Introduction

There are three types of single supply brushed DC motors

Series wound (unidirectional)

- Shunt wound (bi-directional by changing connections)
- Permanent magnet (bi-directional by reversing current)

In this application note, the permanent magnet DC motor is used. The stator of the motor is composed of two or more permanent magnet pole pieces while the rotor is composed of windings that are connected to a mechanical commutator. The opposite end polarities of the energized windings and the stator magnet attract and the rotor will rotate until it is aligned with the stator. Just at the instant where the rotor reaches alignment, the brushes move the commutator contacts and energize the next winding.



Block Diagram of Permanent Magnet DC Motor

The advantages and disadvantages of DC motors are summarized in table 1.

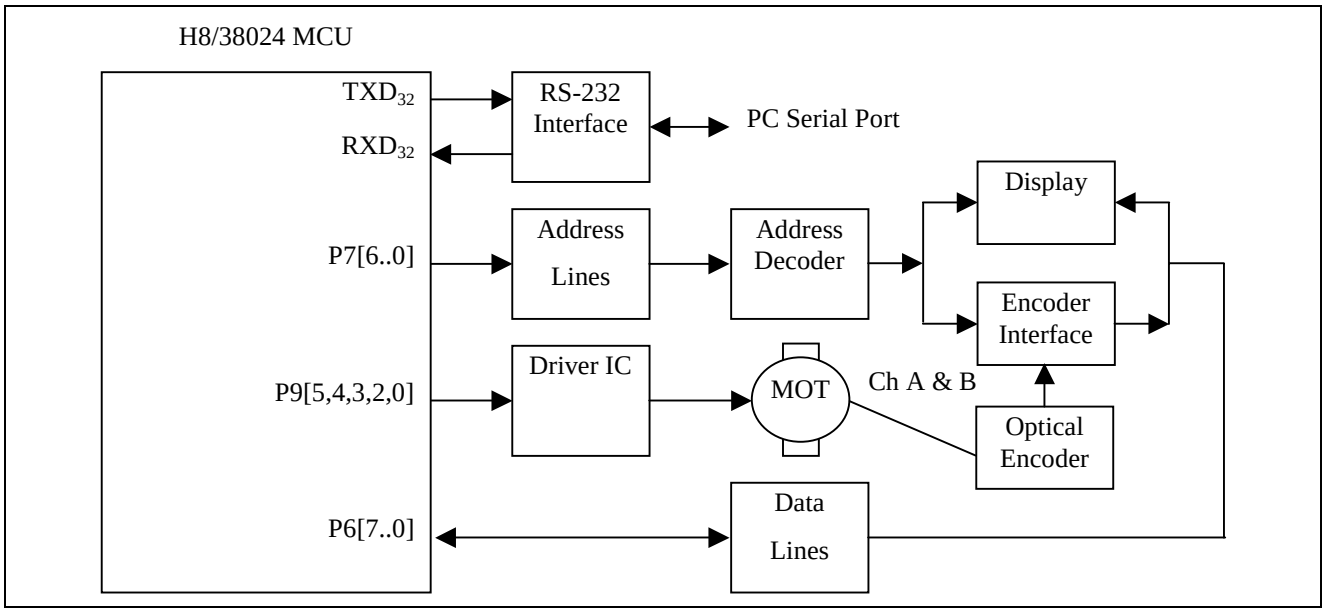
Advantages and Disadvantages of DC Motors

Advantages	Disadvantages
Easy to control	Requires maintenance
Efficient	Cannot be used in certain hazardous environments
Smaller than induction motors of the same power	Noisier than induction motors
Fewer components required for speed control	Lower horsepower (HP)

2. System Overview

The block diagram of the servomotor system described in this application note is shown in Figure 2. The system is comprised of the following components:

- H8/38024 SLP MCU
- RS-232C Interface
- Full-bridge PWM Motor Driver
- Faulhaber Brush-DC Motor with Optical Encoder



System Block Diagram

In this application note, the roles of the MCU are as follows:

- measure the motor position
- generate the speed profile
- calculate the compensation algorithm
- produce the drive signal to the PWM motor driver
- transmit the desired and actual speeds back to the PC through the RS-232 interface

One of the two available pulse-width modulation (PWM) modules with 10-bit resolution is used to generate the motor drive signal. The PWM frequency is 20 kHz at device operating frequency of 10 MHz. The torque applied to the motor is determined by duty cycle of the PWM signal. The PWM signal is connected to an H-bridge driver IC capable of delivering 1.3 A.

A Faulhaber brush-DC motor with optical encoder with the following characteristics shown in table 2 is used here:

Motor Characteristics

Parameters	Value	Units
Nominal voltage	6	V
No-load speed	9300	Rpm
Torque constant	6.10	MNm/A

3. Hardware Implementation

Table 3 summarizes the usage of peripherals.

Peripheral Usage

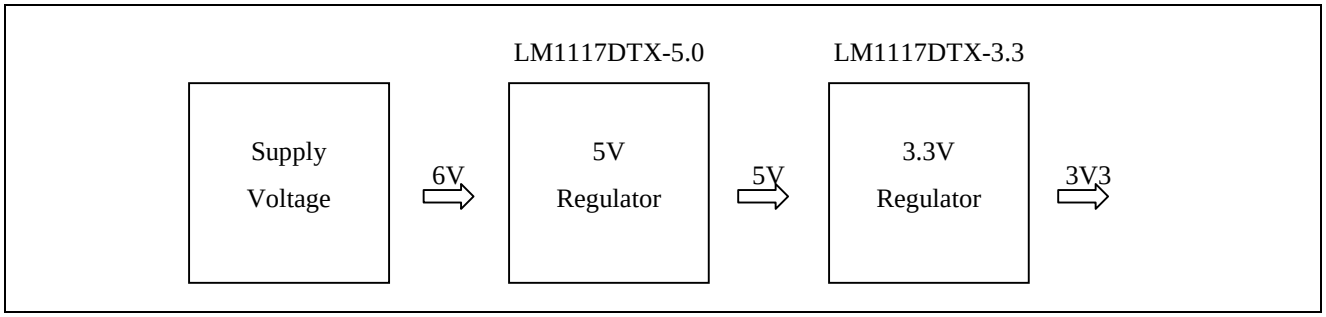
Peripheral	Function
Port 6[7..0]	"Data" bus
Port 7[6..0]	"Address" bus
P77	WRITE/READ_N
SCI (TXD_{32} and RXD_{32})	Communication with host PC
P90/PWM1	*PHASE
P92	*REF
P93	*ENABLE_N
P94	*MODE
P95	*BRAKE_N
Timer F	Servo update time-base

Note: Connected to the PWM motor driver (A3953SB). Refer to section 3.6 for details.

3.1 Power Supplies

Three separate supplies are required in this application example:

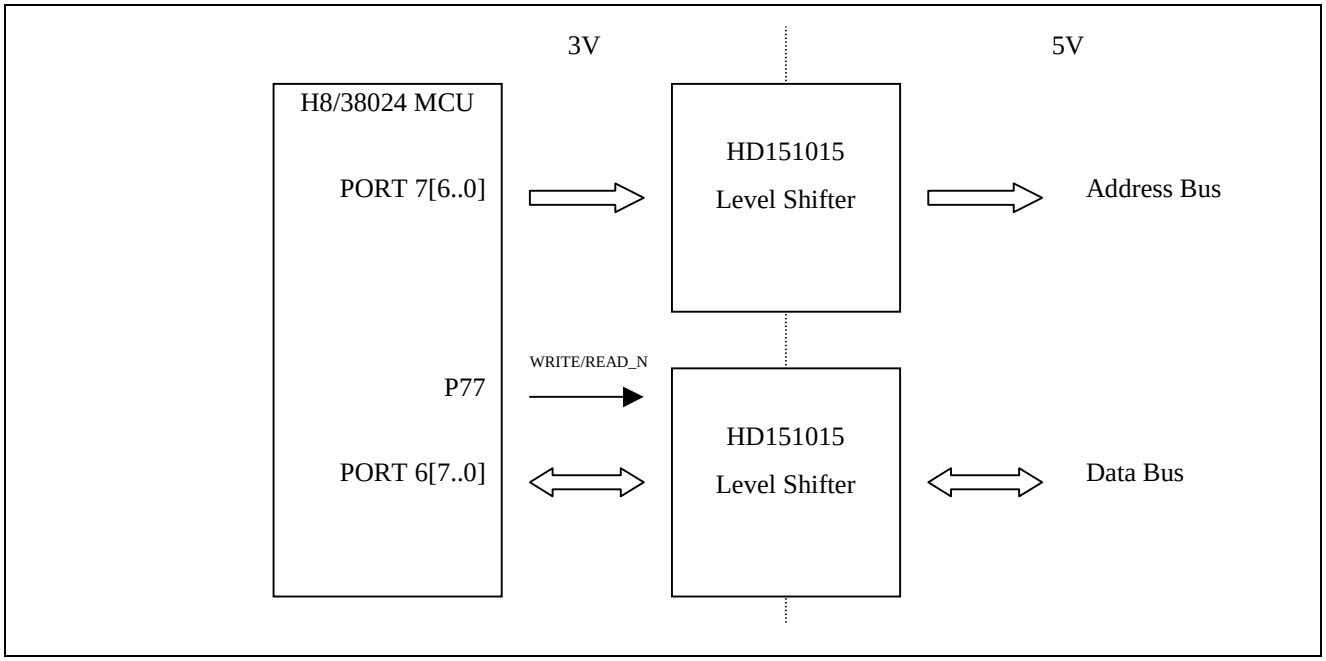
- 6 V for DC motor
- 5 V for 74HCT138, level shifter, alphanumeric display, quadrature encoder interface
- 3.3 V for MCU, RS-232 transceiver, level shifter, PWM motor driver



Power Supplies

3.2 Address and Data Buses

To allow the H8/38024 SLP MCU to access memory-mapped external devices/memory/peripherals, separate address and data buses with control signal WRITE/READ_N are constructed using general I/O ports as shown in figure 4. The MCU is operating at 3.3 V while some devices operate at 5 V. The function of the level shifters is to interface the MCU to the 5-V devices.



Address and Data Buses

3.3 Address Decoder

A 74HCT138 3 to 8 line decoder is used here to select the memory-mapped devices namely, the Alphanumeric Intelligent Display (DLR1414) and the quadrature decoder. Table 4 lists the addresses.

Address Mapping

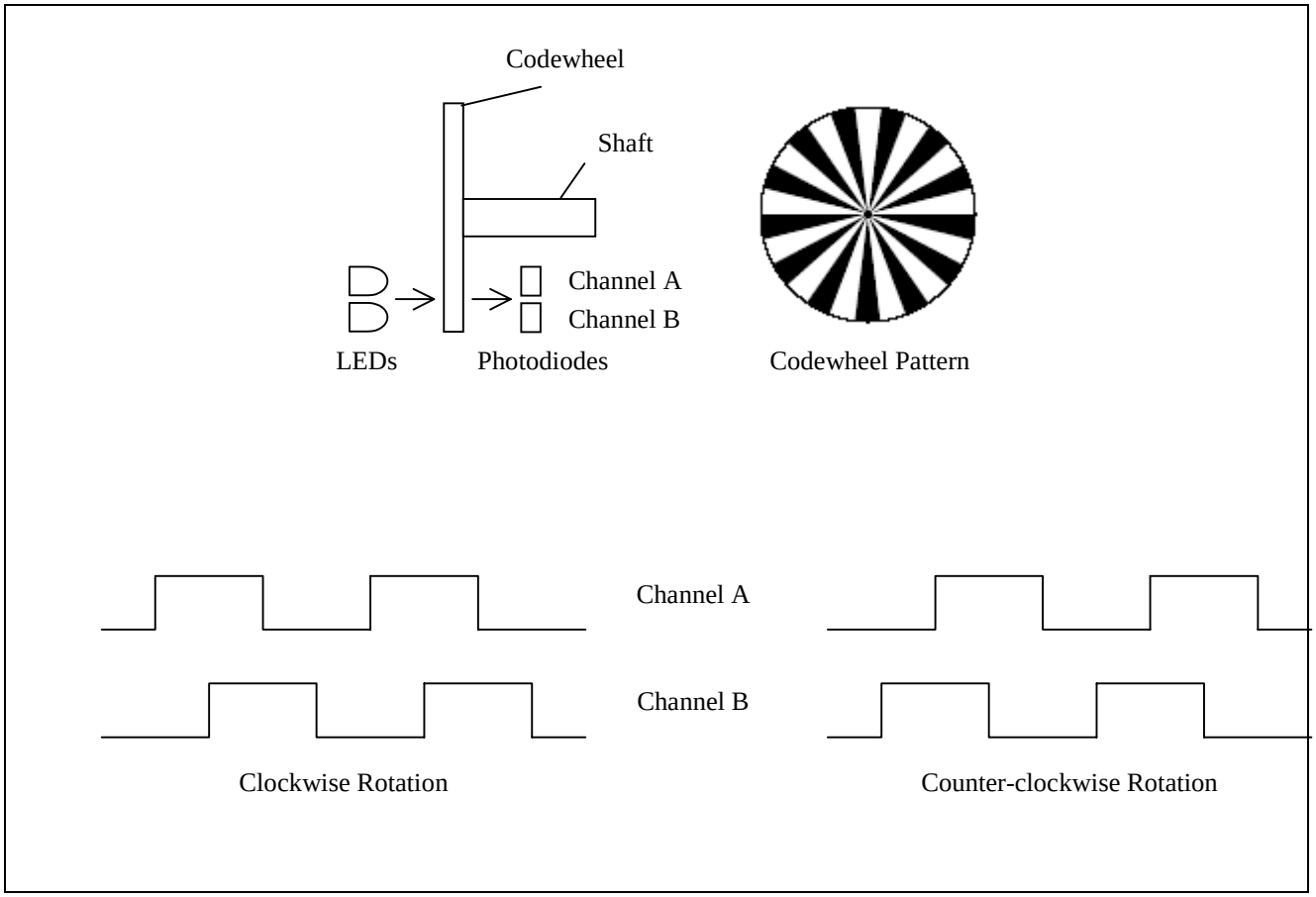
Address (Hexadecimal)	Device
44	Encoder (high byte)
45	Encoder (low byte)
C0	Display (first digit)
C1	Display (second digit)
C2	Display (third digit)
C3	Display (fourth digit)
C8	Reset encoder

3.4 RS-232C Transceiver

The Serial Communication Interface (SCI) pins TXD_{32} and RXD_{32} are connected to the Sipex SP3232 RS-232C transceiver. This allows the MCU to communicate with the Host PC.

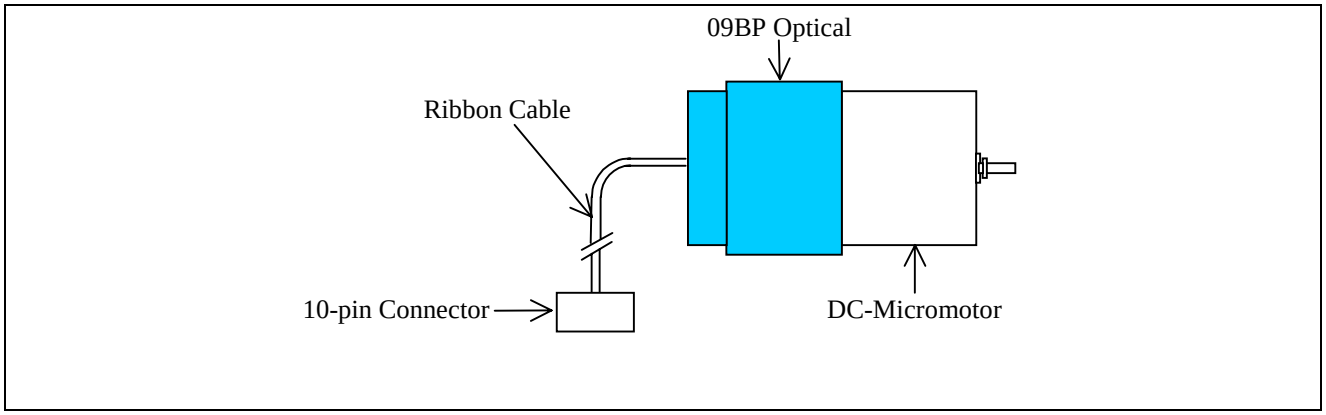
3.5 DC-Micromotor with Optical Encoder

The Faulhaber optical encoder (09BP series with 180 lines per revolution) is used in combination with the DC-Micromotor (2230 U 006 S series) for indication and control of both shaft velocity and direction of rotation, as well as positioning. Two LED sources transmit light through a metal disc with a number of light transmitting slits to give two channels (A and B) with a 90° phase shift as shown in Figure 5. For clockwise rotation, channel A leads channel B. Similarly, channel B leads channel A for counter-clockwise rotation.



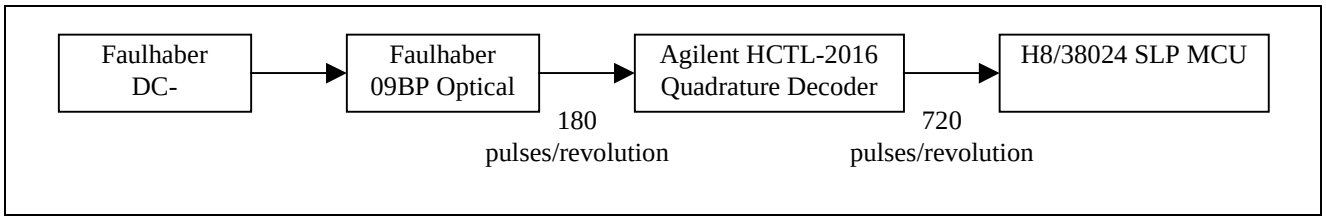
Operation of Optical Encoder

The supply voltage for the motor and encoder as well as the two channel output signals are interfaced with a 150mm ribbon cable and a 10-pin connector as shown in figure 6.



DC-Micromotor with Optical Encoder

The outputs from the encoder (channels A and B) are connected to the Agilent HCTL-2016 quadrature decoder, which consists of a 4x quadrature decoder, binary up/down 16-bit counter and bus interface function. The use of Schmitt-triggered CMOS inputs and input noise filters allows reliable operation in noisy environments.



Encoder Resolution

3.6 PWM Motor Driver

The Allegro A3953SB is a full-bridge PWM motor driver designed for bi-directional pulse-width modulated current control of inductive loads. It can deliver continuous output currents of ±1.3A and operate at voltages up to 50V. The peak load current limit is set by the user's selection of an input voltage reference voltage and external sensing resistor. The fixed off-time pulse duration is set by a user-selected external RC timing network. The internal circuit protection includes thermal shutdown with hysteresis, transient-suppression diodes, and crossover current protection.

With the ENABLE_N input held low, the PHASE input controls load current polarity by selecting the appropriate source and sink driver pair. When a logic low is applied to the BRAKE input, the braking function is enabled. This overrides ENABLE and PHASE to turn off both source drivers and turn on both sink drivers. This brake function can be used to dynamically brake brushed DC motors.

A 50% PWM duty cycle will produce zero motor torque. A 0% and 100% duty cycle will produce maximum torque in the reverse and forward direction, respectively.

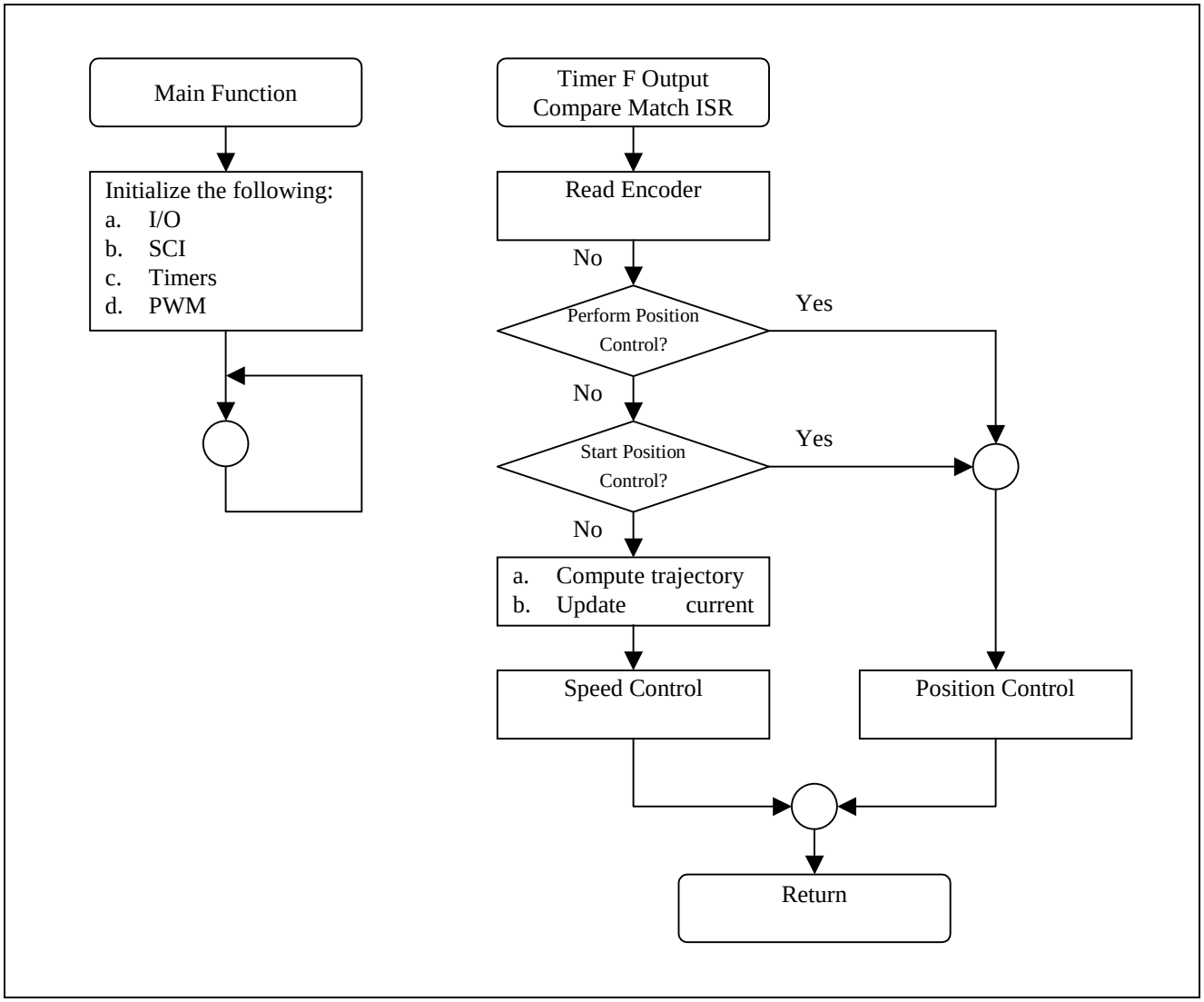
4. Software Implementation

The source code is written in the C language for easy implementation and compiled using the free H8 Tiny/SLP toolchain (version 5.0.0) for HEW Version 2.2 (Release 15).

The functions of the source codes are as follows:

- Measurement of motor position
- Calculation for the compensator algorithm
- Generation of profile
- RS-232C communication

Figure 8 shows the flowchart of the main function where the I/O ports, SCI (38400bps, 1 stop bit, parity disabled), timers and PWM are initialized. Timer F Output Compare interrupt service routine is programmed to occur every 1.5 ms. The encoder value is read. The new trajectory value and PID values are computed and the PWM output is set. The desired and actual speeds are also transmitted back to the PC.



Flowcharts

4.1 Encoder Feedback

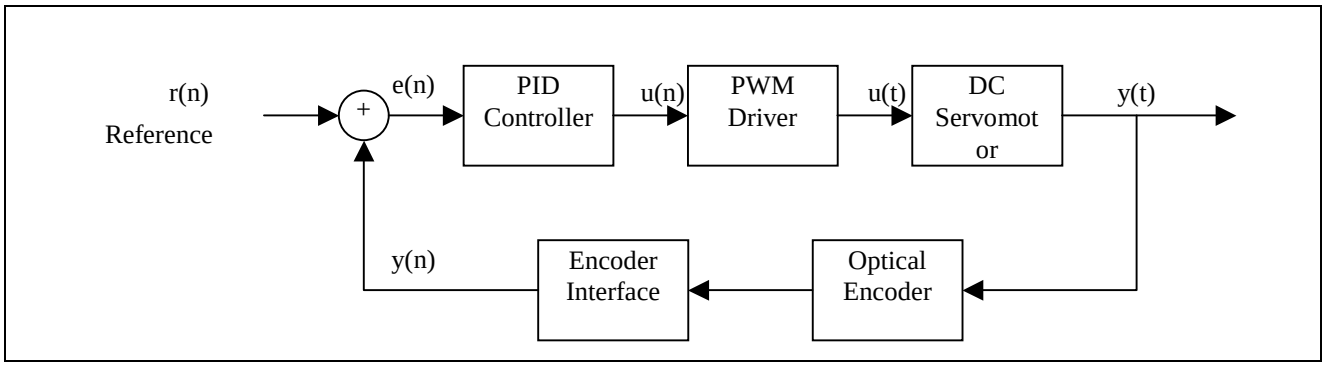
Both speed and direction feedback can be derived from the optical encoder mounted on the motor shaft. The encoder output signals are processed by the HCTL-2016 4x quadrature encoder and then fed to a binary 16-bit up/down counter. The contents of the 16-bit counter are read through an 8-bit bus interface in 2 sequential bytes. The high byte (bits 15 to 8, SEL = 0) is read first followed by the lower byte (bits 7 to 0, SEL = 1). A positive number indicates clockwise rotation whereas a negative number indicates counter-clockwise rotation. Therefore, 1 revolution will produce $4 \times 180 = 720$ pulses.

The servo sample interval is selected to be 1.5ms, which means that the 16-bit counter is read every 1.5ms. Computation is then performed and the value is then output to the PWM driver. The speed and direction of rotation can be determined by differencing the current and previous 16-bit counter values. For example,

Encoder Count		Speed (counts/sampling interval)	Direction of rotation
Current	Previous		
1000	900	100	Clockwise
-1095	-1000	-95	Counter-clockwise

4.2 PID Controller

One problem faced by the system designer is that load on the motor is subjected to change. Other factors such as age, calibration and environment also affect the operation of the motor. Consequently, feedback mechanisms are typically employed to control the motor speed. The Proportional-Integral-Derivative (PID) controller is commonly used. In this example, the reference signal r(n) indicates the desired motor speed. The actual motor speed (subject to change based on load), is measured by the optical encoder mounted on the motor.



PID Controller

The Proportional-Integral-Derivative controller is described by:

$$u(t) = K_p \cdot e(t) + K_i \cdot e_iT + K_d \cdot (de/dt)$$

where

- K_p is the proportional gain
- K_i is the integral gain
- K_d is the derivative gain
- T is the sampling interval
- e(t) is the error signal and equal to the difference between the desired and measured speeds

The proportional term considers only the difference between the present reference value and the actual speed being measured. This term is weighed by a gain factor, K_p, derived using knowledge of the specific motor under control. The integral term considers the summation of all the errors that have been measured previously. This term serves to smooth out oscillations by damping the system. For example, when the measured speed is slightly less than the desired speed, the proportional term would resolve to increase the applied power, speeding up the motor. If the last few measurements indicate the motor to be too fast, the integral term would reduce or even prevent the introduction of more power. In addition, the integral term eliminates accumulated error. Since the amount of energy introduced by the proportional term is proportional to the amount of error (deviation of actual motor speed from the desired motor speed), its influence is very small when the motor is close to the desired speed. When a very small error has existed for many iterations of the PID loop, the integral term will accumulate and produce a control change to correct that error.

The derivative term considers the rate of change at which the error is changing and helps to improve settling time. When the actual motor speed is far below its target speed, we would like to accelerate faster. If the acceleration of the motor is constant, then the rate of error changes will be constant too (i.e., the differential term). We would like to see the differential term increase to some constant value. When we approach the target speed, we would like to reduce the rate at which the motor is increasing its speed to prevent overshooting i.e., differential term decrease to zero.

The integral term is described in discrete form as the summation of error terms measured at each sampling interval.

$$e_i \cdot T = \sum_{j=0}^{i} \left(R_j - Y_j \right)$$

Similarly, the differential term can be discretely approximated as:

$$\frac{de}{dt} = \frac{\left(e_n - e_{n-1} \right)}{T}$$

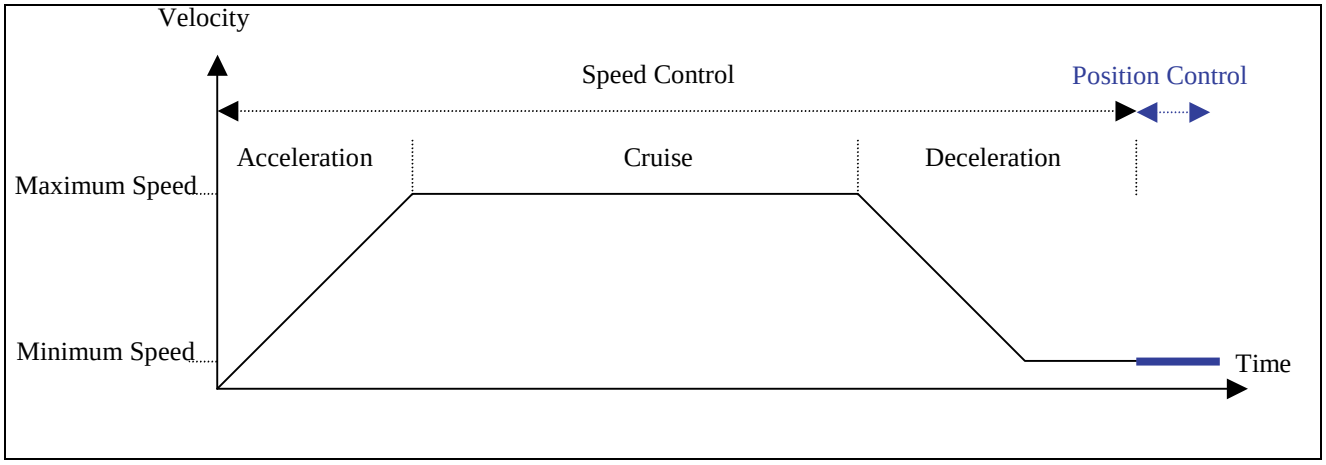
where e_n and e_{n-1} are error signals calculated from the current and previous measurements.

This allows the PID to be approximated in discrete form as:

$$u(n) = K_p \cdot e(n) + K_i \cdot \sum e_i \cdot T + K_d \cdot \frac{\left[e(n) - e(n-1) \right]}{T}$$

4.3 Profile Generation

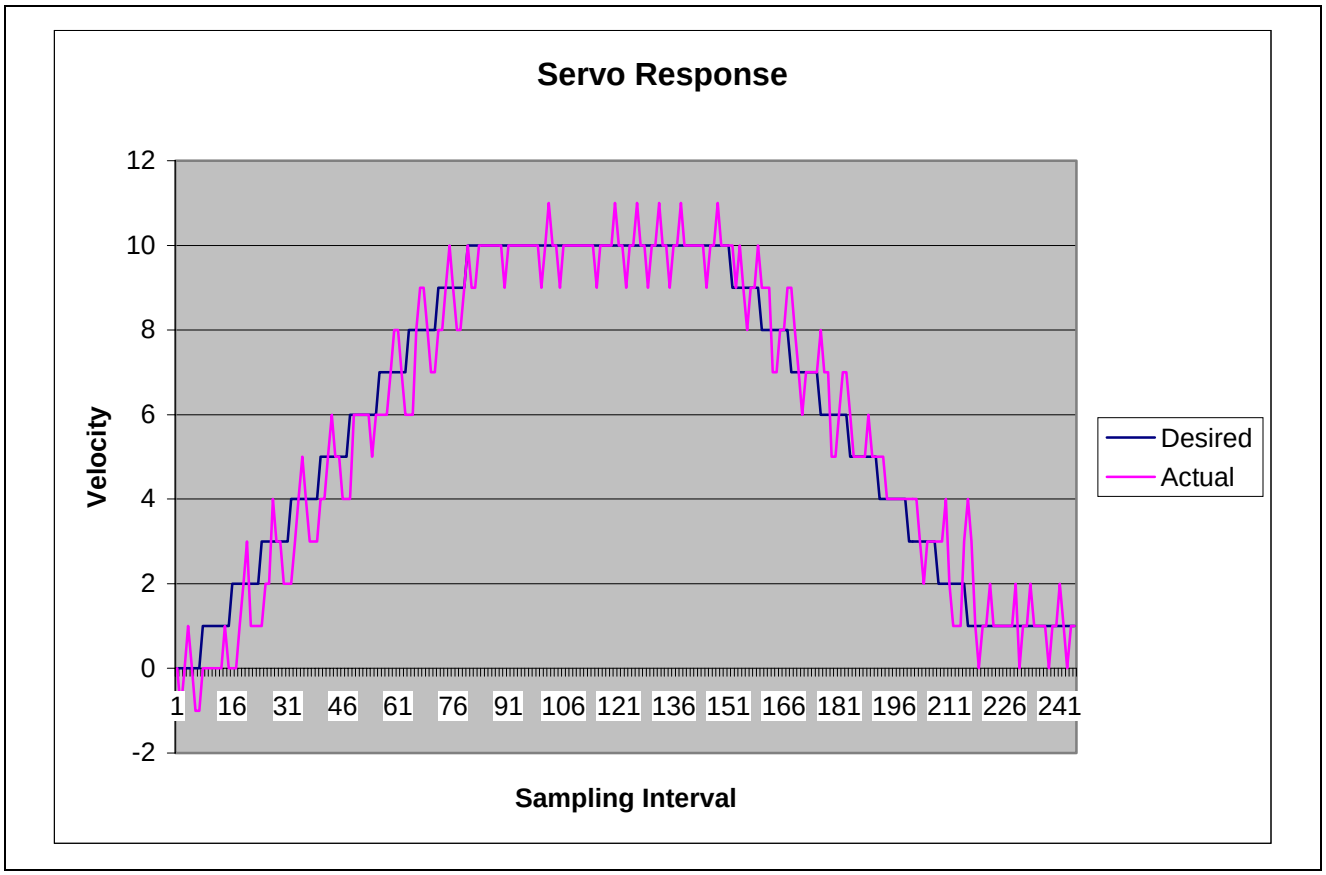
A trajectory generation is essential for motion control. A linear piecewise velocity generation is implemented in this application. The velocity is incremented by a constant value until a specified maximum velocity is maintained. This maximum velocity is maintained until 70% of the total rotation has been covered. The velocity is then decremented by the same deceleration value until the minimum velocity is reached. Once the motor has reached the required rotation, position control is then utilized to hold the motor in position (braking function).



PID Controller

4.4 RS-232C Communications

The function of the RS-232C communications is to transmit the desired and actual speeds in (counts/sampling interval) to the PC. Using any terminal emulation software such as Tera Term Pro, these values are then sent to EXCEL and the response is plotted below as shown in figure 11. By altering the various gains and plotting the results, the system can be fine-tuned.



Servo Response

4.5 PID Tuning

When designing a PID controller for a given system, follow the steps below to obtain the desired response.

- Set the PID controller to proportional mode only i.e., K_i and K_d are set to 0.
- Set the proportional gain (K_p) to a small value to limit the output so that the servomotor system will not spin out of control initially. Safety concerns should be considered especially if the motor is to be used for driving some machinery, etc.
- Set a small reference value and observe the response of the controlled variable, for example the speed or position of the motor.
- Set K_p to the point that will bring the system to the reference quickly without overshoot.
- Incorporate the integral gain (K_i) to eliminate the steady state error
- Include the derivative gain (K_d) to improve the transient response.
- Adjust the gains (K_p, K_i K_d) until the desired response is obtained.

4.6 Source Codes

The source codes are mainly in the following files

- Dcmotor.c
  - Contains the main function
  - Performs initialization of the I/O ports, SCI, timers and PWM channel.
  - Generation of trajectory, display of encoder count, computation of PID values, speed and position control
- int_prg.c
  - Contains the Timer F Output Compare Interrupt Service Routine
  - Update the position count

For example, to program the motor to rotate 1 complete revolution in the counter-clockwise direction, the following constants defined in “Dcmotor.c” have to be set:

Parameter	Value	Remarks
distance_reference	-720	1° → 2 counts 1 revolution → 360° → (360 * 2) = 720 counts Negative for counter-clockwise direction
maximum_speed_reference	-7	Maximum reference speed during acceleration/cruise Negative for counter-clockwise direction
minimum_speed_reference	-1	Minimum reference speed during deceleration Negative for counter-clockwise direction
speed_adjustment	-1	Step adjustment for reference speed Negative for counter-clockwise direction
proportional_gain	6	Proportional gain for PID controller
integral_gain	2	Integral gain for PID controller
derivative_gain	4	Derivative gain for PID controller

```
/************************************************************************/
/*                                                                      */
/*  FILE        :Dcmotor.c                                              */
/*  DATE        :Mon, Jan 12, 2004                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/38024                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

#ifdef __cplusplus
extern "C" {
#endif
void abort(void);
#ifdef __cplusplus
}
#endif

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

//-----------------------------------------------------------------------
//Constant Declarations
//-----------------------------------------------------------------------

//Constants for Address Decoder
#define  first_digit              0xC0
#define  second_digit             0xC1
#define  third_digit              0xC2
#define  fourth_digit             0xC3

#define  enc_count_high           0x44
#define  enc_count_low            0x45

#define  reset_2016               0xC8

#define  de_select                0x3F

//Constants for Motor Control
#define  distance_reference       -720        //1 degree = 2 encoder counts
#define  maximum_speed_reference  -7          //negative for counter-clockwise
#define  minimum_speed_reference  -1          //negative for counter-clockwise
#define  speed_adjustment         -1          //negative for counter-clockwise
#define  proportional_gain        6
#define  integral_gain            2
#define  derivative_gain          4

#define max_pos_con_val           256

//Others
#define  address_bus              P_IO.PDR7.BYTE    //Address Bus
#define  data_bus                 P_IO.PDR6.BYTE    //Data Bus
```

```
//-----------------------------------------------------------------------
//Function Prototypes
//-----------------------------------------------------------------------

void init_sci(void);
void char_put(char);
void PutStr(char *);

void init_port(void);

void display_char(unsigned char, unsigned char);

void init_timers(void);

void display_word(unsigned int);

void reset_encoder(void);
int  read_encoder(void);

void init_pwm(void);

void serial_transmit(unsigned int);

void speed_control(void);

void speed_profile(void);

void position_control(void);

//-----------------------------------------------------------------------

//16-bit number: +ve for clockwise, -ve for counter-clockwise
int   encoder_count, current_encoder_count, previous_encoder_count;

//16-bit number: reference speed: +ve for clockwise, -ve for counter-clockwise
int   speed_reference;

int   current_speed;
int   current_speed_error, previous_speed_error, sum_speed_error,
      diff_speed_error;
int   speed_control_output;

int   position_reference, position_reference_a;
int   current_position_error, previous_position_error, sum_position_error,
      diff_position_error;
int   position_control_output;

int   speed_increment;      //+ve for clockwise, -ve for counter-clockwise
int   max_speed_reference;  //+ve for clockwise, -ve for counter-clockwise
int   min_speed_reference;  //+ve for clockwise, -ve for counter-clockwise

int   temp_encoder_count, temp_position_reference;
```

```
unsigned int         position_control_count;

static unsigned char reference_step = 0;

int                  kp_gain, ki_gain, kd_gain;

unsigned char        start_position_control;

//----------------------------------------------------------------------

void main(void)
{
   //----------------------------------------------------------------------

   init_port();

   init_sci();

   init_timers();

   init_pwm();

   reset_encoder();

   //----------------------------------------------------------------------

   speed_reference = 0;

   speed_increment = speed_adjustment;
   max_speed_reference = maximum_speed_reference;
   min_speed_reference = minimum_speed_reference;

   previous_speed_error = 0;
   sum_speed_error = 0;
   diff_speed_error = 0;

   kp_gain = proportional_gain;
   ki_gain = integral_gain;
   kd_gain = derivative_gain;

   position_reference = distance_reference;

   position_reference_a = ((position_reference * 7)/ 10);
   if (position_reference < 0)
   {
      position_reference_a = 0 - position_reference_a;
   }

   current_position_error = 0;
   previous_position_error = 0;
   sum_position_error = 0;

   start_position_control = 0;
```

```
    position_control_count = 0;

    //---------------------------------------------------------------------

    P_SYSCR.IENR2.BIT.IENTFH = 1;         //Enable Timer FH interrupt requests

    //if Timer FH interrupt request flag (IRRTFH) is set, clear to 0
    if (P_SYSCR.IRR2.BIT.IRRTFH == 1)
       P_SYSCR.IRR2.BIT.IRRTFH = 0;

    set_imask_ccr(0);                     //Clear IMASK

    //---------------------------------------------------------------------

    while (1)
    {
    }
}

//-------------------------------------------------------------------------

void abort(void)
{

}

//-------------------------------------------------------------------------

/*
   init_pwm()
*/

void init_pwm(void)
{
   P_PWM1.PWCR1.BYTE = 0xFD;              //Conversion period is 1024/phi

   P_PWM1.PWDRL1.BYTE  = 0x00;            //Write to lower 8 bits

   P_PWM1.PWDRU1.BYTE  = 0xFE;            //Write to upper 2 bits

   P_IO.PDR9.BYTE &= 0xF7;                //Port 9[3] : Clear ENABLE_N
}

//-------------------------------------------------------------------------

/*
   reset_encoder(): Reset enocder counts
*/

void reset_encoder(void)
{
   P_IO.PCR6.BYTE  = 0xFF;                //Set Port 6[7..0] as output
```

```
   address_bus &= de_select;

   address_bus = reset_2016;

   address_bus &= de_select;
}

//---------------------------------------------------------------------

/*
   read_encoder(): Read encoder counts
*/

int   read_encoder(void)
{
   int            counts;
   unsigned char  low_byte, high_byte;

   P_IO.PCR6.BYTE  = 0x00;              //Set Port 6[7..0] as input

   address_bus &= de_select;

   address_bus = enc_count_high;        //Address

   high_byte = data_bus;                //Data

   address_bus &= de_select;

   address_bus = enc_count_low;         //Address

   low_byte = data_bus;                 //Data

   address_bus &= de_select;

   counts = (int)((high_byte << 8) | low_byte);

   return(counts);
}

//---------------------------------------------------------------------

/*
   init_timers() : Set up Timer F
*/

void init_timers(void)
{
   //Timer Control Register F
   //TOLL = '1': Initial output for TMOFH is high
   //CKSH2 = '0', CKSH1 = '0', CKSH0 = '0': 16-bit mode, counting on TCFL
overflow signal
   //Clock input to TCFL at phi/4 i.e., (10MHz/2/4 = 1.25MHz)
   P_TMRF.TCRF.BYTE = 0x86;
```

```
   //Timer Control/Status Register F
   //CCLRH = '1': in 16-bit mode, TCF clearing by compare match is enabled
   P_TMRF.TCSRF.BYTE = 0x10;

   //1.5ms -> 0x753
   P_TMRF.OCRF.BYTE.H = 0x07;                  //Output Compare Register FH
   P_TMRF.OCRF.BYTE.L = 0x53;                  //Output Compare Register FL
}

//---------------------------------------------------------------------

/*
   init_port() : Set up the I/O ports

   a. Port 6[7..0] -> Data[7..0]
   b. Port 7[7..0] -> Address[7..0]
      Note that Port 7_7 also functions as the WRITE/READ_N signal
*/

void init_port(void)
{
   P_IO.PMR3.BYTE = 0x04;                      //P32 functions as TMOFH

   P_LCD.LPCR.BYTE = 0x00;                     //SEG[32..1] as I/O Port

   P_IO.PCR6.BYTE  = 0xFF;

   P_IO.PCR7.BYTE  = 0xFF;

   P_IO.PDR6.BYTE  = 0xFF;

   P_IO.PDR7.BYTE  = 0xFF;

   P_IO.PMR9.BYTE = 0xF1;                      //PWM1
}

//---------------------------------------------------------------------

/*
   display_char()

   a. Port 6[7..0] -> Data[7..0]
   b. Port 7[7..0] -> Address[7..0]
      Note that Port 7_7 also functions as the WRITE/READ_N signal
*/

void display_char(unsigned char digit_position, unsigned char digit_info)
{
   P_IO.PCR6.BYTE  = 0xFF;              //Set Port 6[7..0] as output

   address_bus &= de_select;
```

```
    data_bus = digit_info;                //Data

    address_bus = digit_position;         //Address

    address_bus &= de_select;
}

//---------------------------------------------------------------------

/*
    display_word()
*/

void display_word(unsigned int display_data)
{
    unsigned char  position, digit_info, digit_position;

    P_IO.PCR6.BYTE  = 0xFF;                     //Set Port 6[7..0] as output

    for (position = 4 ; position != 0 ; position--)
    {
       switch (position)
       {
          case 1:
             digit_position = first_digit;
             digit_info = (unsigned char)(display_data & 0x000F);
             break;

          case 2:
             digit_position = second_digit;
             digit_info = (unsigned char)((display_data & 0x00F0) >> 4);
             break;

          case 3:
             digit_position = third_digit;
             digit_info = (unsigned char)((display_data & 0x0F00) >> 8);
             break;

          default:
             digit_position = fourth_digit;
             digit_info = (unsigned char)((display_data & 0xF000) >> 12);
             break;
       }

       if ((digit_info >= 0) && (digit_info <= 9))
          digit_info += 0x30;
       else
       {
          if ((digit_info >= 0xA) && (digit_info <= 0xF))
          {
             digit_info -= 0xA;
             digit_info += 0x41;
          }
```

```
      }

      address_bus &= de_select;

      data_bus = digit_info;             //Data

      address_bus = digit_position;      //Address

      address_bus &= de_select;
   }
}

//---------------------------------------------------------------------

/*
   serial_transmit()
*/

void serial_transmit(unsigned int display_data)
{
   unsigned char  position, digit_info;

   for (position = 4 ; position != 0 ; position--)
   {
      switch (position)
      {
         case 1:
            digit_info = (unsigned char)(display_data & 0x000F);
            break;

         case 2:
            digit_info = (unsigned char)((display_data & 0x00F0) >> 4);
            break;

         case 3:
            digit_info = (unsigned char)((display_data & 0x0F00) >> 8);
            break;

         default:
            digit_info = (unsigned char)((display_data & 0xF000) >> 12);
            break;
      }

      if ((digit_info >= 0) && (digit_info <= 9))
         digit_info += 0x30;
      else
      {
         if ((digit_info >= 0xA) && (digit_info <= 0xF))
         {
            digit_info -= 0xA;
            digit_info += 0x41;
         }
      }
```

```
      char_put(digit_info);
   }
}


//---------------------------------------------------------------------

/*
   init_sci() : Sets up the Serial Communication Interface for debugging
purposes.
*/

void init_sci(void)
{
   //SCR3 : |TIE|RIE|TE|RE|MPIE|TEIE|CKE1|CKE0|
   //TIE : Transmit interrupt enable
   //RIE : Receive interrupt enable
   //TE  : Transmit enable
   //RE  : Receive enable
   //MPIE : Multiprocessor interrupt enable
   //TEIE : Transmit end interrupt enable
   //CKE1 : Clock enable 1
   //CKE0 : Clock enable 0

   //CKE1 = CKE0 = 0
   //asynchronous mode, internal clock source, SCK32 functions as I/O port
   P_SCI3.SCR3.BYTE &= 0x00; //clear TE & RE

   //SMR : |COM|CHR|PE|PM|STOP|MP|CKS1|CKS0| : |0|0|0|0|0|0|0|0|
   //COM : Communication Mode  : 0 : asynchronous mode
   //CHR : Character Length    : 0 : character length = 8 bits
   //PE  : Parity Enable       : 0 : parity bit addition and checking disabled
   //PM  : Parity Mode         : 0 : even parity (no effect since parity is
already disabled)
   //STOP: Stop Bit Length     : 0 : 1 stop bit
   //MP  : Multiprocessor Mode : 0 : multiprocessor communication function
disabled
   //|CKS1|CKS0| : Clock Select: |0|0| : clock source for baud rate generator
= clk
   P_SCI3.SMR.BYTE = 0x00;

   //For clk = 10MHz, bit rate = 38400 bps, n = 0, N = 3
   P_SCI3.BRR = 3;

   //minimum of 1-bit delay = 417ns
   nop();
   nop();
   nop();

   //SPCR : |---|---|SPC32|---|SCINV3|SCINV2|---|---| : |1|1|1|0|0|0|0|0|
   //SPC32 = 1 : P42 functions as TXD32 output pin
   //need to set TE bit in SCR3 after setting this bit to 1
```

```
    //SCINV3 = 0 : TXD32 output data is not inverted
    //SCINV2= 0 : RXD32 input data is not inverted
    //Bits 7 and 6 are reserved and always read as 1
    //Bits 4, 1 and 0 are reserved and only 0 can be written to these bits
    P_SCI3.SPCR.BYTE = 0xE0;

    P_SCI3.SCR3.BYTE |= 0x30; //Set TE & RE
}

//---------------------------------------------------------------------

/*
   char_put() : Transmits a character to the PC for debugging purposes.
*/

void char_put(char OutputChar)              //Serial Port
{
     //SSR  : |TDRE|RDRF|OER|FER|PER|TEND|MPBR|MPBT|
    //TDRE : transmit data register empty
    //RDRF : receive data register full
    //OER  : overrun error
    //FER  : framing error
    //PER  : parity error
    //TEND : transmit end
    //MPBR : Multiprocessor bit receive
    //MPBT : Multiprocessor bit transfer

    while ((P_SCI3.SSR.BIT.TDRE) == 0);         //Wait for TDRE = 1

    P_SCI3.TDR = OutputChar;

    while ((P_SCI3.SSR.BIT.TEND) == 0);         //Wait for TEND = 1

    P_SCI3.SSR.BIT.TEND = 0;
}

//---------------------------------------------------------------------

/*
   PutStr() : Transmits a string of characters to the PC for debugging
purposes.

*/
void PutStr(char *str)
{
     while (*str != 0)
    {
       char_put(*str++);
    }
}

//---------------------------------------------------------------------
```

```
/*
   speed_control()

   1. current_speed = current_encoder_count - previous_encoder_count

   2. current_speed_error = speed_reference - current_speed

   3. output = kp_gain * speed_error + ki_gain * sum(speed_error) + kd_gain *
(current_speed_error - previous_speed_error);

   range-limited from -512 to 511
*/

void speed_control(void)
{
   //-------------------------------------------------------------

   previous_encoder_count = current_encoder_count;
   current_encoder_count = encoder_count;

   current_speed = current_encoder_count - previous_encoder_count;

   //-------------------------------------------------------------

   serial_transmit(speed_reference);

   //-------------------------------------------------------------

   previous_speed_error = current_speed_error;

   //-------------------------------------------------------------

   current_speed_error = speed_reference - current_speed;

   PutStr("/");
   serial_transmit(current_speed);
   PutStr("\r\n");

   //-------------------------------------------------------------

   sum_speed_error += (current_speed_error);

   diff_speed_error = (current_speed_error - previous_speed_error);

   //-------------------------------------------------------------

   speed_control_output = (int)((kp_gain * current_speed_error) +
(sum_speed_error / ki_gain) + (diff_speed_error * kd_gain));

   //-------------------------------------------------------------
   //limit speed_control_output from -511 to +511

   if (speed_control_output > 511)
```

```
      speed_control_output = 511;
   else
   {
      if (speed_control_output < -511)
         speed_control_output = -511;
   }

   //introduce offset of 512
   speed_control_output += 512;

   P_PWM1.PWDRL1.BYTE  = (unsigned char)(speed_control_output & 0x00FF);
   //Write to lower 8 bits

   P_PWM1.PWDRU1.BYTE  = (unsigned char)((speed_control_output & 0xFF00)>> 8);
   //Write to upper 2 bits

   //-------------------------------------------------------------
}

//---------------------------------------------------------------------

/*
   speed_profile()   - generates the speed profile
*/

void speed_profile(void)
{
   int   temp_encoder_count;

   reference_step++;

   temp_encoder_count = encoder_count;

   if (position_reference < 0)
      temp_encoder_count = 0 - encoder_count;

   if (temp_encoder_count > position_reference_a)
   {
      //decelerate
      if (reference_step > 7)
      {
         reference_step = 0;

         //For clockwise rotation, speed_increment is +ve
         //For counter-clockwise rotation, speed_increment is -ve

         //For example
         //a. clockwise, speed_increment = 1, speed_reference = 5 ->
speed_reference = 5 - 1 = 4
         //b. counter-clockwise, speed_increment = -1, speed_reference = -5 ->
speed_reference = -5 - (-1) = -4

         if (speed_reference < 0)  //Counter-clockwise
```

```
         {
            speed_reference -= speed_increment;

            if (speed_reference > min_speed_reference)
               speed_reference = min_speed_reference;
         }
         else                       //Clockwise
         {
            speed_reference -= speed_increment;

            if (speed_reference < min_speed_reference)
               speed_reference = min_speed_reference;
         }
      }
   }
   else
   {
      //accelerate and cruise
      if (reference_step > 7)
      {
         reference_step = 0;

         //For clockwise rotation, speed_increment is +ve
         //For counter-clockwise rotation, speed_increment is -ve

         //For example
         //a. clockwise, speed_increment = 1, speed_reference = 5 ->
speed_reference = 5 + 1 = 6
         //b. counter-clockwise, speed_increment = -1, speed_reference = -5 ->
speed_reference = -5 + (-1) = -6

         speed_reference += speed_increment;

         if (speed_reference < 0)  //Counter-clockwise
         {
            if (speed_reference < max_speed_reference)
               speed_reference = max_speed_reference;
         }
         else                       //Clockwise
         {
            if (speed_reference > max_speed_reference)
               speed_reference = max_speed_reference;
         }
      }
   }
}

//---------------------------------------------------------------------

/*
   position_control()

   1. position_error = reference_encoder_count - current_encoder_count
```

```
   2. output = kp_gain * current_position_error
             + sum(position_error) / ki_gain
             + kd_gain * (position_error - previous_position_error);

   range-limited from -512 to 511
*/

void position_control(void)
{
   kp_gain = 10;
   ki_gain = 2;
   kd_gain = 3;

   position_control_count++;

   //-------------------------------------------------------------

   previous_position_error = current_position_error;

   //-------------------------------------------------------------

   current_position_error = position_reference - encoder_count;

   //-------------------------------------------------------------

   sum_position_error += (current_position_error);

   diff_position_error = (current_position_error - previous_position_error);

   //-------------------------------------------------------------

   position_control_output = (int)((kp_gain * current_position_error) +
(kd_gain * diff_position_error));

   //-------------------------------------------------------------
   //limit position_control_output from -128 to +128

   if (position_control_output > max_pos_con_val)
      position_control_output = max_pos_con_val;
   else
   {
      if (position_control_output < -max_pos_con_val)
         position_control_output = -max_pos_con_val;
   }

   //introduce offset of 512
   position_control_output += 512;

   P_PWM1.PWDRL1.BYTE  = (unsigned char)(position_control_output & 0x00FF);
   //Write to lower 8 bits
```

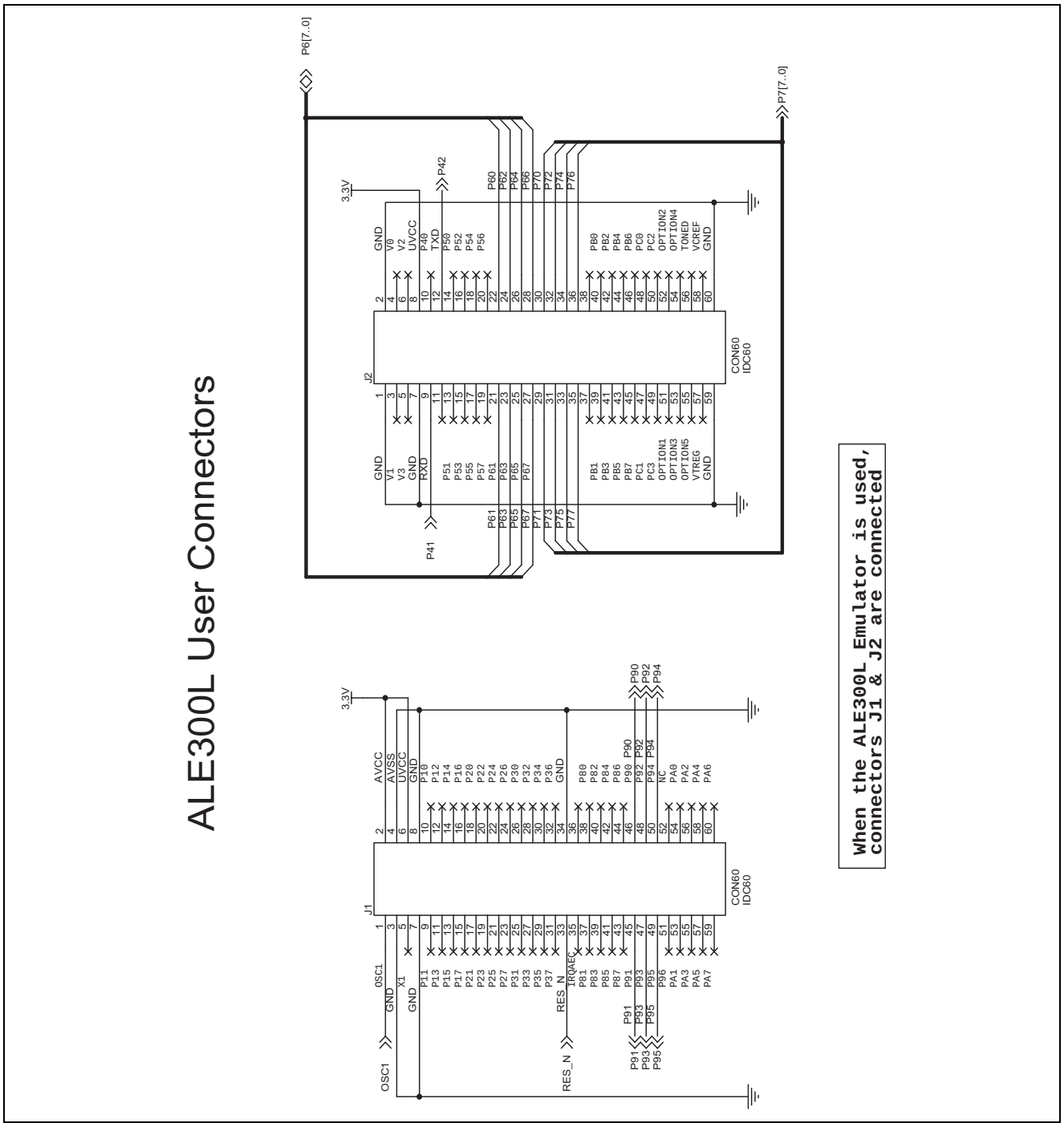
```
   P_PWM1.PWDRU1.BYTE  = (unsigned char)((position_control_output & 0xFF00)>>
8);   //Write to upper 2 bits

   //---------------------------------------------------------------
}

//-------------------------------------------------------------------------
```

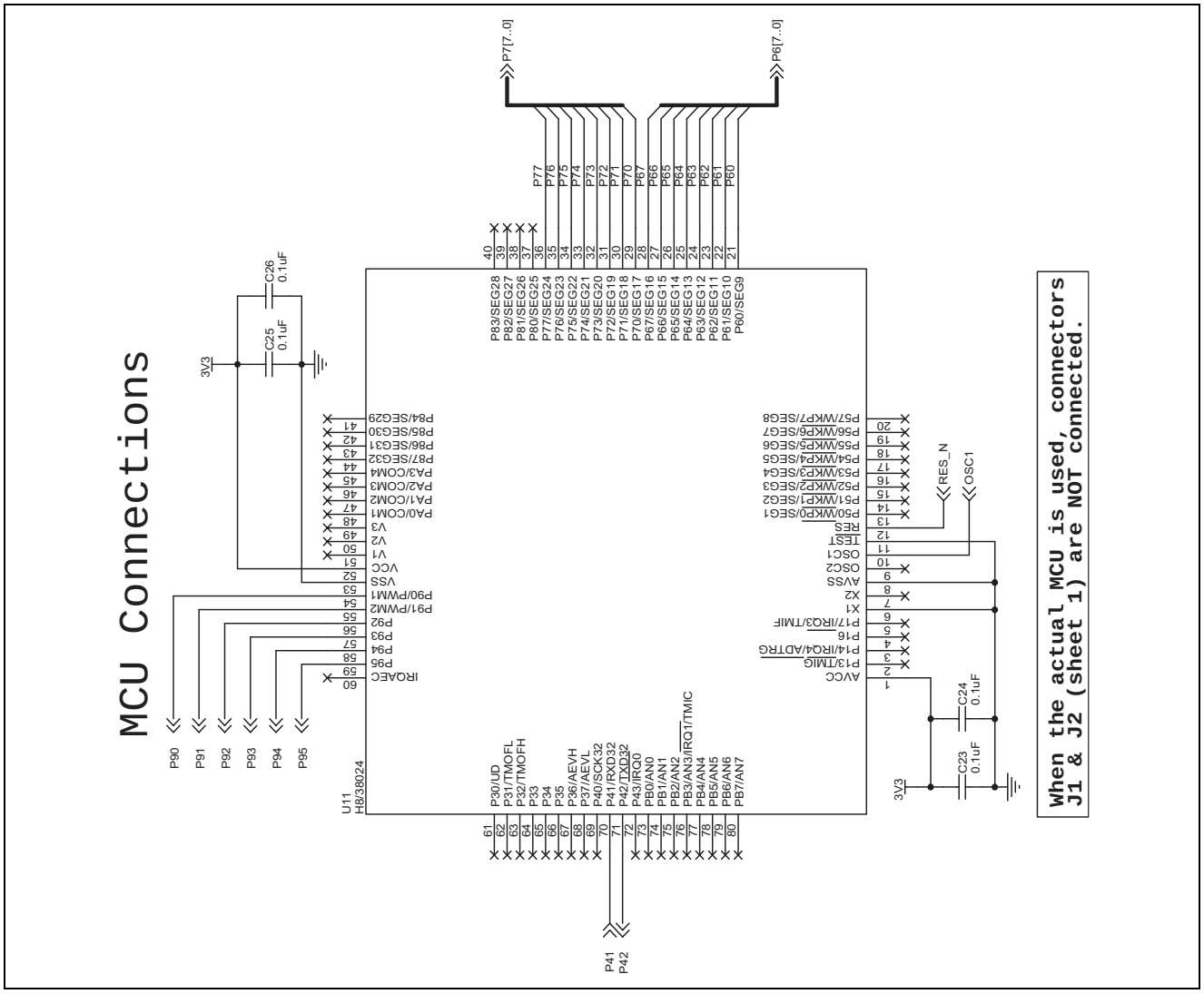
```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Mon, Jan 12, 2004                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/38024                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

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

//-----------------------------------------------------------------------
extern void display_word(unsigned int);
extern int  read_encoder(void);
extern void serial_transmit(unsigned int);
extern void speed_control(void);

extern void speed_profile(void);

extern void position_control(void);

extern int           encoder_count, position_reference;
extern int           temp_encoder_count, temp_position_reference;
extern unsigned int  position_control_count;
extern unsigned char start_position_control;

//-----------------------------------------------------------------------

#pragma section IntPRG

//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

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

//  vector 5 IRQ1
__interrupt(vect=5) void INT_IRQ1(void) {/* sleep(); */}

//  vector 6 IRQAEC
__interrupt(vect=6) void INT_IRQAEC(void) {/* sleep(); */}

//  vector 7 Reserved

//  vector 8 Reserved
```

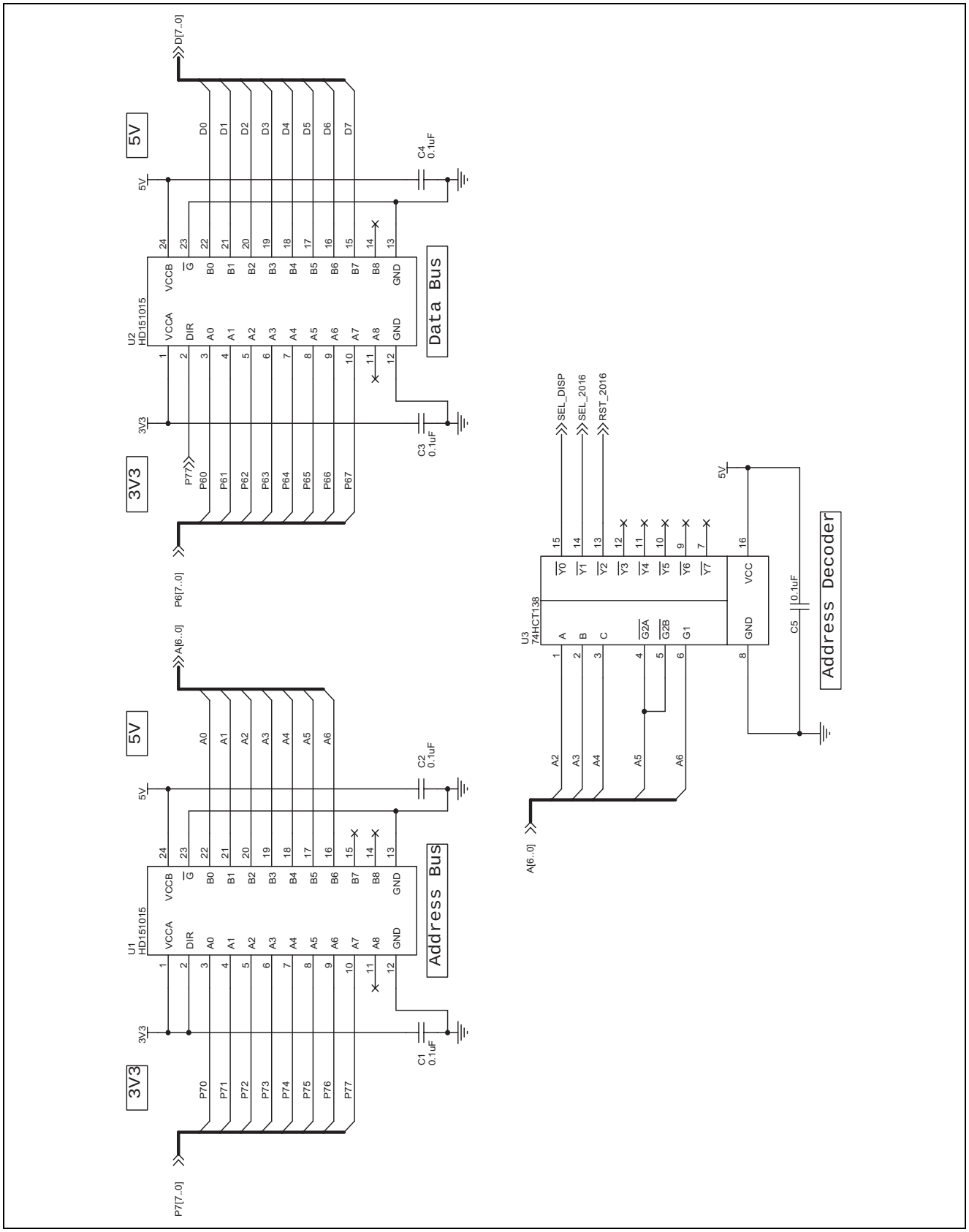
```
//  vector 9 WKP0_7
__interrupt(vect=9) void INT_WKP0_7(void) {/* sleep(); */}

//  vector 10 Reserved

//  vector 11 Timer A Overflow
__interrupt(vect=11) void INT_TimerA(void) {/* sleep(); */}

//  vector 12 Counter Overflow
__interrupt(vect=12) void INT_Counter(void) {/* sleep(); */}

//  vector 13 Reserved

//  vector 14 Timer FL Overflow
__interrupt(vect=14) void INT_TimerFL(void) {/* sleep(); */}

//  vector 15 Timer FH Overflow

__interrupt(vect=15) void INT_TimerFH(void)
{
   //if Timer FH interrupt request flag (IRRTFH) is set, clear to 0
   if (P_SYSCR.IRR2.BIT.IRRTFH == 1)
      P_SYSCR.IRR2.BIT.IRRTFH = 0;

   if (P_TMRF.TCSRF.BIT.CMFH == 1)
   {
      P_TMRF.TCSRF.BIT.CMFH = 0;

      encoder_count = read_encoder();

      if (start_position_control)
      {
         if (position_control_count < 300)
            position_control();
         else
            P_IO.PDR9.BYTE &= 0xDF;      //Apply BRAKE_N -> P95
      }
      else
      {
         if (position_reference < 0)
         {
            temp_encoder_count = 0 - encoder_count;
            temp_position_reference = 0 - position_reference;
         }
         else
         {
            temp_encoder_count = encoder_count;
            temp_position_reference = position_reference;
         }

         if (temp_encoder_count > temp_position_reference)
         {
            start_position_control = 1;
```

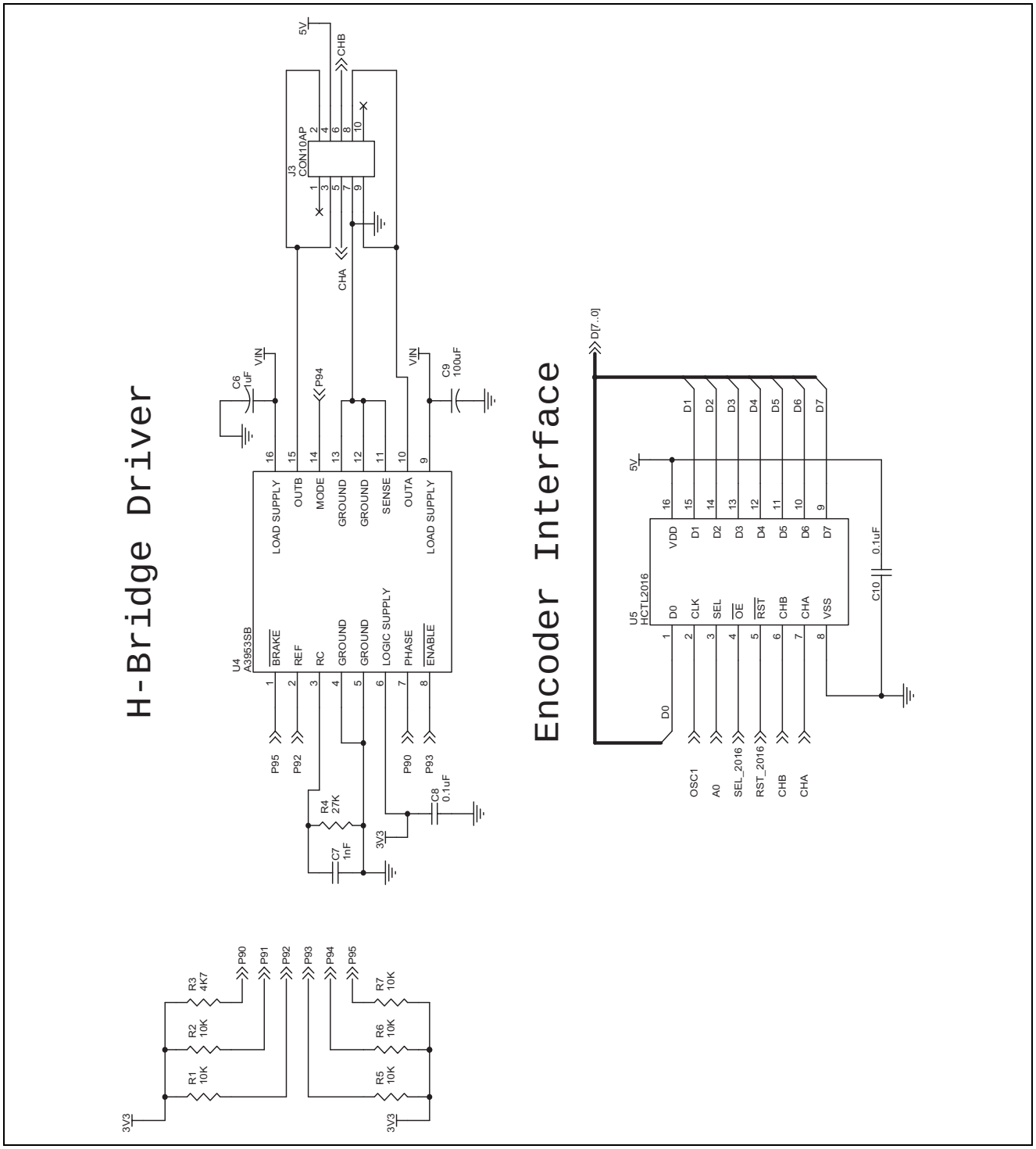
```
            position_control();
         }
         else
         {
            P_IO.PDR9.BYTE ^= 0x02;      //Toggle P91

            speed_profile();
            speed_control();
            P_IO.PDR9.BYTE ^= 0x02;      //Toggle P91
         }
      }

      display_word(encoder_count);
   }

   /* sleep(); */
}

//  vector 16 Reserved

//  vector 17 Reserved

//  vector 18 SCI3
__interrupt(vect=18) void INT_SCI3(void) {/* sleep(); */}

//  vector 19 ADI
__interrupt(vect=19) void INT_ADI(void) {/* sleep(); */}

//  vector 20 Direct Transition
__interrupt(vect=20) void INT_Direct_Transition(void) {/* sleep(); */}
```

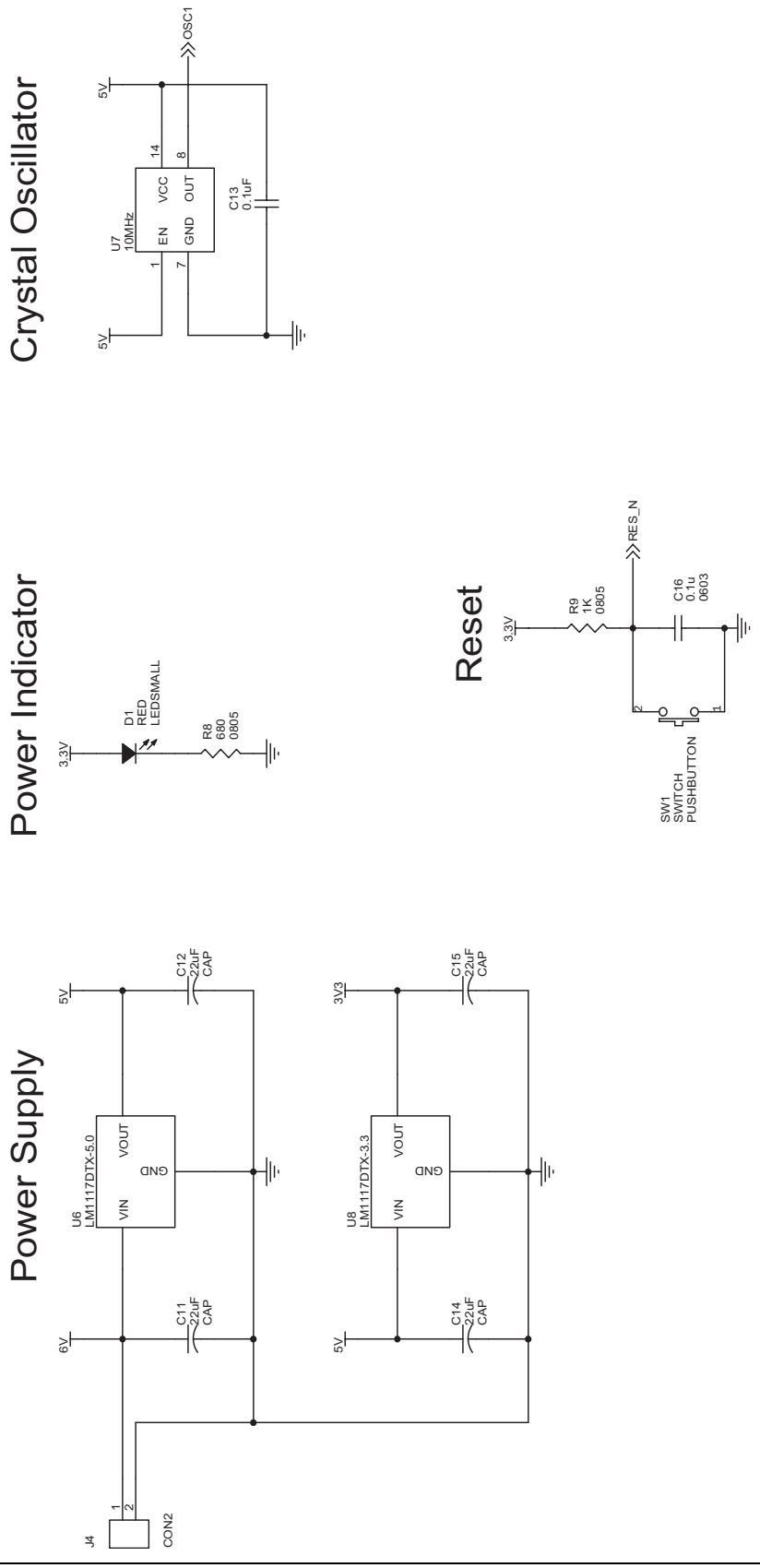
5. Hardware Schematics

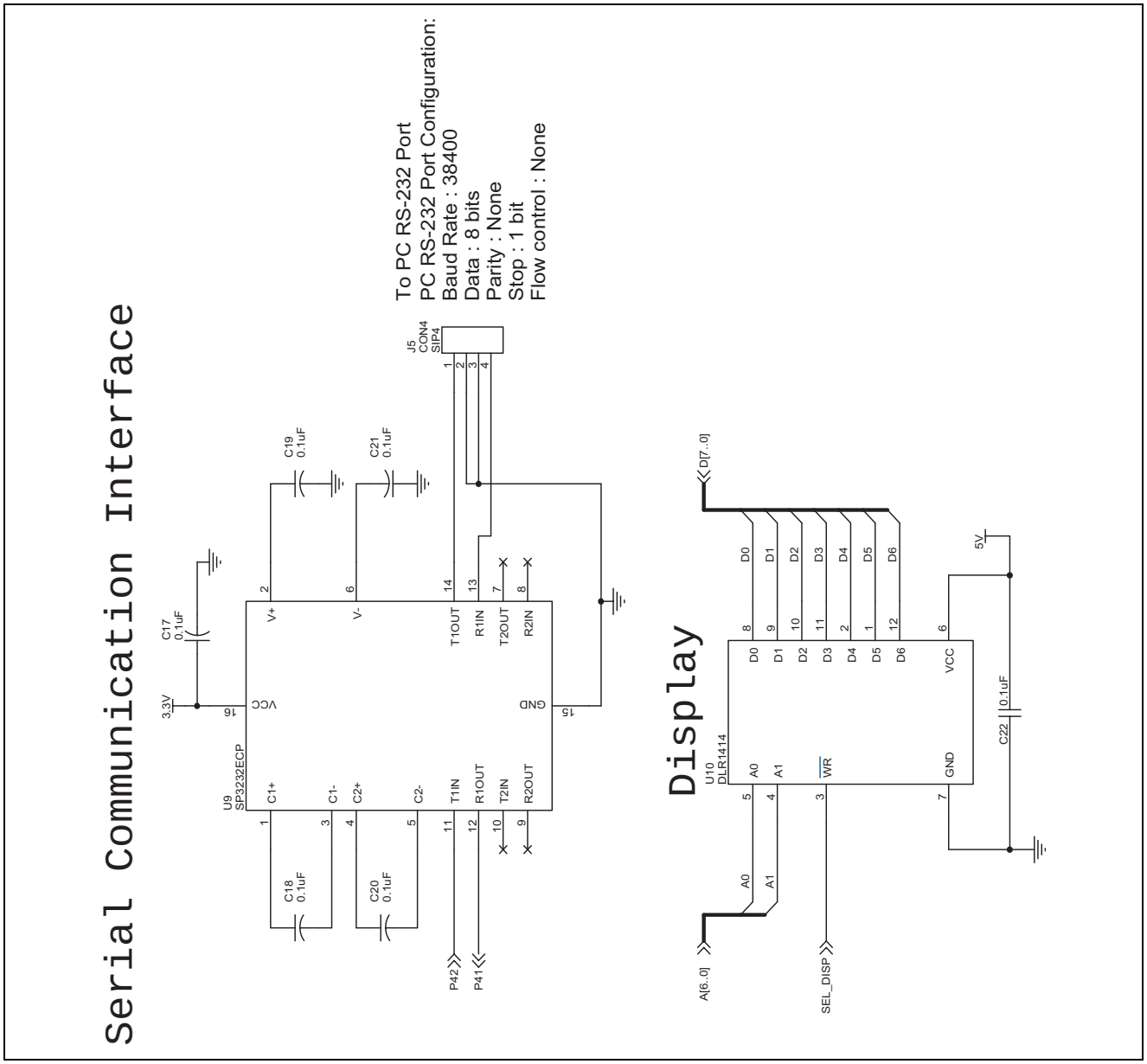



MCU Connections
When the actual MCU is used, connectors J1 & J2 (sheet 1) are NOT connected.
U11
H8/38024
P7[7..0]
P6[7..0]
3V3
C26 0.1uF
C25 0.1uF
C24 0.1uF
C23 0.1uF
RES_N
OSC1
P90
P91
P92
P93
P94
P95
P41
P42


Address Bus
Data Bus
Address Decoder
U1
HD151015
U2
HD151015
U3
74HCT138
5V
3V3
C1
0.1uF
C2
0.1uF
C3
0.1uF
C4
0.1uF
C5
0.1uF
P7[7..0]
P6[7..0]
A[6..0]
D[7..0]
SEL_DISP
SEL_2016
RST_2016


H-Bridge Driver
Encoder Interface
5V
CHB
CHA
J3
CON10AP
C6
1uF
VIN
P94
C9
100uF
U4
A3953SB
LOAD SUPPLY
OUTB
MODE
GROUND
GROUND
SENSE
OUTA
LOAD SUPPLY
BRAKE
REF
RC
GROUND
GROUND
LOGIC SUPPLY
PHASE
ENABLE
P95
P92
P90
P93
R4
27K
C7
1nF
C8
0.1uF
3V3
R3
4K7
R2
10K
R1
10K
R7
10K
R6
10K
R5
10K
P90
P91
P92
P93
P94
P95
D[7..0]
U5
HCTL2016
VDD
D0
CLK
SEL
OE
RST
CHB
CHA
VSS
D1
D2
D3
D4
D5
D6
D7
C10
0.1uF
OSC1
A0
SEL_2016
RST_2016


Crystal Oscillator
OSC1
5V
14
8
U7
10MHz
EN
VCC
OUT
GND
C13
0.1uF
1
7
5V
Power Indicator
3.3V
D1
RED
LEDSMALL
R8
680
0805
Reset
RES_N
3.3V
R9
1K
0805
C16
0.1u
0603
SW1
SWITCH
PUSHBUTTON
Power Supply
5V
C12
22uF
CAP
U6
LM1117DTX-5.0
VIN
VOUT
GND
C11
22uF
CAP
6V
3V3
C15
22uF
CAP
U8
LM1117DTX-3.3
VIN
VOUT
GND
C14
22uF
CAP
5V
J4
1
2
CON2


Serial Communication Interface
To PC RS-232 Port
PC RS-232 Port Configuration:
Baud Rate : 38400
Data : 8 bits
Parity : None
Stop : 1 bit
Flow control : None
J5
CON4
SIP4
C19
0.1uF
C21
0.1uF
C17
0.1uF
3.3V
U9
SP3232ECP
V+
V-
VCC
GND
C1+
C1-
C2+
C2-
T1IN
R1OUT
T2IN
R2OUT
T1OUT
R1IN
T2OUT
R2IN
C18
0.1uF
C20
0.1uF
P42
P41
Display
D[7..0]
U10
DLR1414
D0
D1
D2
D3
D4
D5
D6
A0
A1
WR
VCC
GND
5V
C22
0.1uF
A[6..0]
SEL_DISP

6. References

1. LM1117/LM1117I 800mA Low-Dropout Linear Regulator, 2002, National Semiconductor Corporation.
2. Quadrature Decoder/Counter Interface ICs, 2002, Agilent Technologies.
3. 3953 Full-bridge PWM Motor Driver, 2002, Allegro MicroSystems, Inc.
4. HD151015 9-bit Level Shifter/Transceiver with 3 State Outputs, 3rd Edition, June 1993, Renesas Technology Corp. (Ref. no.: REJ03D0300-0400, http://renesas.com)
5. DLR1414 4-character 5 × 7 Dot Matrix Alphanumeric Intelligent Display with Memory/Decode/Driver, Infineon Technologies.
6. DC-Micromotor 2230 U-006S with 09BP Optical Encoder, Faulhaber Group (http://www.faulhaber.com/).

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.10.04	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. 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 Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.