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.

RENESAS

Application Note

μPD780988 Subseries

8-Bit Single-Chip Microcontrollers

3-Phase Brushless DC Motor Control

μPD780982 **μPD780982(A)**
μPD780983 **μPD780983(A)**
μPD780984 **μPD780984(A)**
μPD780986 **μPD780986(A)**
μPD780988 **μPD780988(A)**
μPD78F0988A **μPD78F0988A(A)**

Document No. U16486EE1V1AN00
Date Published January 2003

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

- The information in this document is current as of 24.01, 2003. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

- NEC Electronics products are classified into the following three quality grades: “Standard”, “Special” and “Specific”.

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated “quality assurance program” for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"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.

"Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is “Standard” unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact NEC Electronics sales representative in advance to determine NEC Electronics 's willingness to support a given application.

Notes: **1.** " NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

2. " NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02.10

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

Sucursal en España
Madrid, Spain
Tel: 091- 504 27 87
Fax: 091- 504 28 60

Succursale Française
Vélizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

Filiale Italiana
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

Branch The Netherlands
Eindhoven, The Netherlands
Tel: 040-244 58 45
Fax: 040-244 45 80

Branch Sweden
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

United Kingdom Branch
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Singapore
Tel: 65-6253-8311
Fax: 65-6250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos, Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

Table of Contents

List of Figures

List of Tables

Chapter 1 Overview

1.1 Abstract

NEC's µPD780988A Subseries microcontroller is specifically designed for motor control applications. This Application Note serves as an example of 3-Phase BLDC motor control with Hall Signals by using the NEC µPD78F0988A device. The concept of this application is based on a closed loop speed control with Hall sensor signal mode.

1.2 Introduction

The requirements of the new generation equipment are growing rapidly. High performance, better efficiency, reduced electromagnetic interference, higher flexibility and reduction of development time are some of the requirements that must be achieved while at the same time reducing the system costs. The BLDC motor technology combines lower cost with high reliability and high efficiency. Longer life, because of no brushes, high starting torque, high no load speed and smaller energy losses are some of the characteristics of the BLDC motor.
NEC offers the µPD780988x Subseries as a member of the low-cost, high performance 78K Family of 8-bit microcontrollers, designed specifically for midrange motor control applications. The device used in this application note is the µPD78F0988A from the 780988x subseries.
The purpose of this application note is to help users understand the dedicated motor control peripherals of the µPD780988 subseries by using a sample application. The software and hardware configurations published here are just examples and are not intended for mass production.

1.3 Overview of µPD78F0988A

Table 1-1: Functional Outline

Item		Function
Internal memory	Flash memory	60 KB[Note 1]
	High-speed RAM	1024 bytes
	Expansion RAM	1024 bytes [Note 2]
Memory space		64 KB
General-purpose register		8 bits x 32 register (8 bits x 8 register x 4 banks)
Instruction cycle		On-chip instruction execution time variable function system clock 8.38 MHz
Instruction set		16-bit operation
		Multiply/divide (8 bits x 8 bits, 16 bits / 8 bits)
		Bit manipulation (set, reset, test, Boolean operation)
		BCD adjust, etc.
I/O ports		Total: 47
		CMOS inputs: 8
		CMOS I/O: 39
Real-time output ports		8 bits x 1 or 4 bits x 2
		6 bits x 1 or 4 bits x 1
A/D converter		10-bit resolution x 8 channels
Serial interface		UART mode: 2 channels
		3-Wire serial I/O mode: 1channel
Timer		16-bit timer/event counter: 2 channels
		8-bit timer/event counter: 3 channels
		10-bit inverter control timer: 1 channel
		Watchdog timer: 1channel
Timer outputs		General-purpose outputs: 5
		Inverter control outputs: 6
Vectored interrupt sources	Maskable	Internal: 16, external 8
	Non-maskable	Internal:1
	Software	1
Power supply voltage		V_{DD} = 4.0 to 5.0 V
Operating ambient temperature		T_A = -40 to +85°C
Package		64-pin plastic SDIP[Note 3]
		64-pin plastic QFP and LQFP

Notes: 1. The capacity of the flash memory can be changed using the internal memory size select register.

2. The capacity of the internal expansion RAM can be changed using the internal expansion RAM size select register.

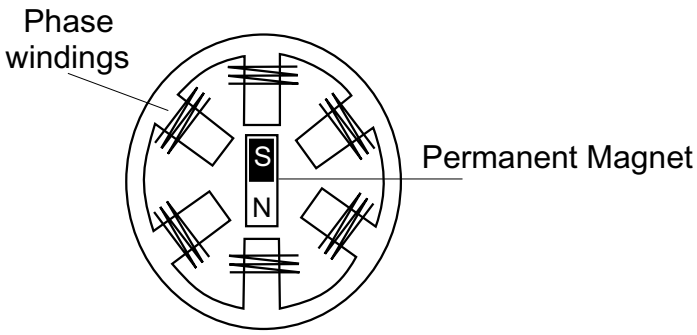
3. Standard quality grade products only.

Chapter 2 3-Phase BLDC Motor Basics

2.1 3-Phase BLDC Motor Basics

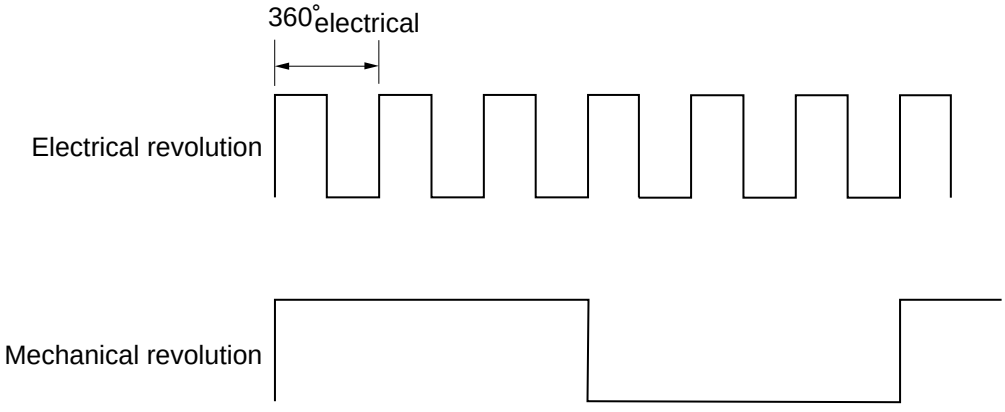
The properties and physical laws of the BLDC motor are almost similar to the DC machine. The structure of the BLDC motor is divided into two parts: The moving part is the rotor, represented by the permanent magnet, and the fixed part is the stator, represented by the phase windings of the magnetic circuit.

Figure 2-1: 3-Phase BLDC motor with one pole pair permanent magnet



The rotor is attracted to the electromagnetic field caused by the energized stator windings and starts to rotate. With an appropriate electromagnetic rotating field on the stator, the rotor follows the electromagnet poles and a rotation of the permanent magnet is created and maintained. There are two main characteristics of a BLDC motor: The first is the EMF (electromotive force) of the motor, which is proportional to its speed and the second is the synchronization between the stator flux and the permanent magnet rotor flux. This fundamental action used in the BLDC motor generates the highest torque of the motor. The synchronization of these two terms requires knowledge of the rotor position. The number of the permanent magnet poles can vary. A greater number of poles create a greater torque for the same current level. It defines also the ratio between electrical and mechanical revolution of the motor, which is described in Figure 2-2.

Figure 2-2: Connection between the electrical and mechanical revolution of the motor



The mathematical relationship between the electrical and mechanical revolution is given with the equation (1).

$$\omega_E = \frac{p}{2} \times \omega_M \qquad (1)$$

Where p is the pole number of the motor, ω_E the electrical speed and ω_M the motor speed. The BLDC motor used in this Application has six pole pairs, thus with 12 poles. So the calculation is

$$\omega_E = 6 \times \omega_M \qquad (2)$$

This means that the electrical speed is six times faster as the mechanical speed of the motor, as shown on Figure 2-2 above. In another words, the electrical supply to the motor has to rotate six times to produce one mechanical turn of the rotor.

The rotor position, which is needed for the synchronization between the rotor and the rotating field, can be estimated by two methods. The absolute position of the BLDC motor can be detected by implementing 3 Hall sensors (sensor mode), or Back-EMF detection (sensorless mode). The motor used in this application is equipped with the 3-Hall sensors, so that the absolute position can be determined to generate precise firing commands. Velocity feedback for the speed control loop is also provided.

The absence of the brushes in the BLDC motor requires electronic commutation to produce a rotating field on the stator.

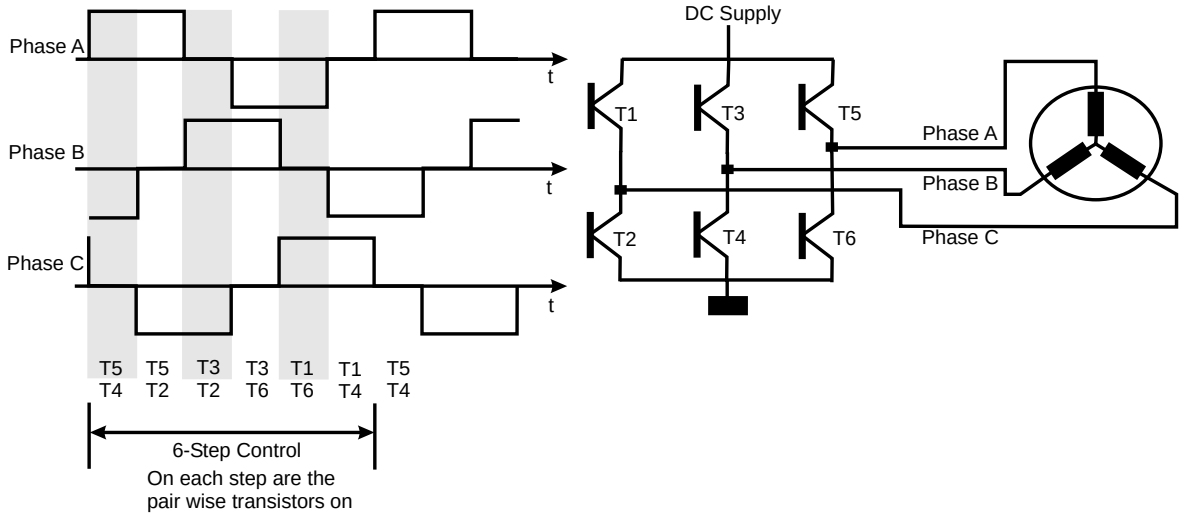
An external circuit, known as an Inverter circuit, provides this electronic commutation. The topology and the function of the Inverter will be described in the next chapter.

The 3-phase BLDC motor used in this application has 3-phase coil configuration **Y** –connected. The supply voltage is 24 V and max. current 5 A.

2.2 3-Phase BLDC Motor Control Requirements

As already mentioned, a 3-Phase Inverter performs the electronic commutation. The principal structure of the 3-Phase inverter is shown below, in Figure 2-3.

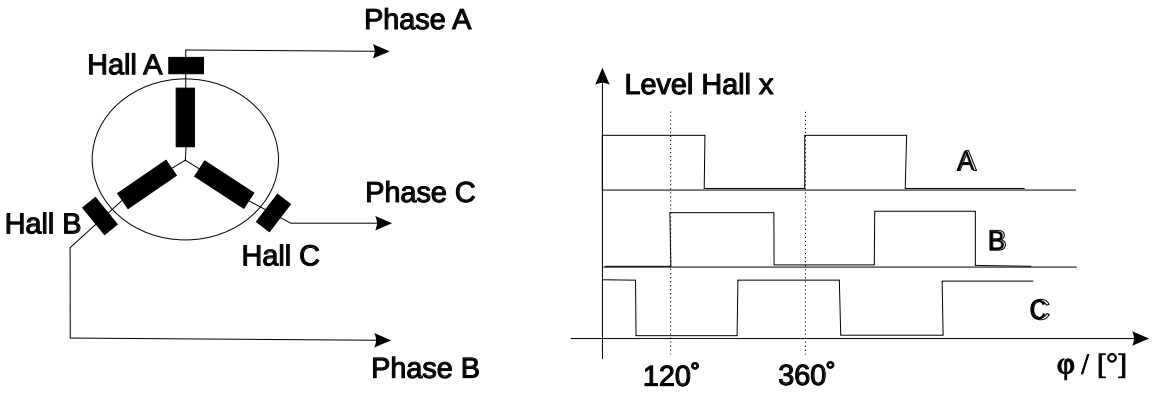
Figure 2-3: Three-Phase Inverter and the current flow



The circuit contains 3 half bridges divided into high side and low side transistors. T1, T3 and T5 are the high side and T2, T4 and T6 are the low side transistors of the circuit. The chosen current switching in this application is soft chopping, because it allows control of the current and the rate of change of the current with minimized current ripple. When using soft chopping two transistors in any one bridge remains in the complementary state. This means that the low side transistor (e.g. T4) is left ON during the phase supply to the motor and the high side transistor (e.g. T5 or T1) switches according to the pulsed control signal. During each supply step, two from three phases of the motor are controlled in this way. This method needs six PWM signals. The current flow during the six - step control is depicted in Figure 2-3 above.

The synchronization between the rotor and the rotating field requires knowledge of the rotor position. The BLDC motor used in this application has 3-Hall sensors implemented. The output signal flow of the sensors, which describes the electrical rotor position, is shown in Figure 2-4.

Figure 2-4: Output signal from the 3-Hall sensors included in the BLDC motor



The Hall sensor signals supply the information needed for the synchronization between the rotor and the rotating field and therefore the time points for the precise firing commands for the inverter to generate the rotating field.
With the three sensor signals from the motor, there are eight possible input code combinations as shown in Table 2-1 bellow. Two of them are not valid for the rotor estimation and are usually caused by an open or shorted sensor line.

The sensors are placed electrically 120^0 apart which equates to a 60^0 mechanical displacement for a six pole motor. The signal flow for each electrical rotation is shown in Figure 2-4.

Table 2-1: Hall sensor signal input codes

Hall A	Hall B	Hall C
0	0	0
1	0	1
1	0	0
1	1	0
0	1	0
0	1	1
0	0	1
1	1	1

BLDC motor control requirements are summarized below:

- Knowledge of the rotor position
- Mechanism to commutate the motor

For the closed-loop speed control of the motor there are two further requirements:

- Measurement of the motor speed and/or motor current
- PWM signal to control the motor speed and power.

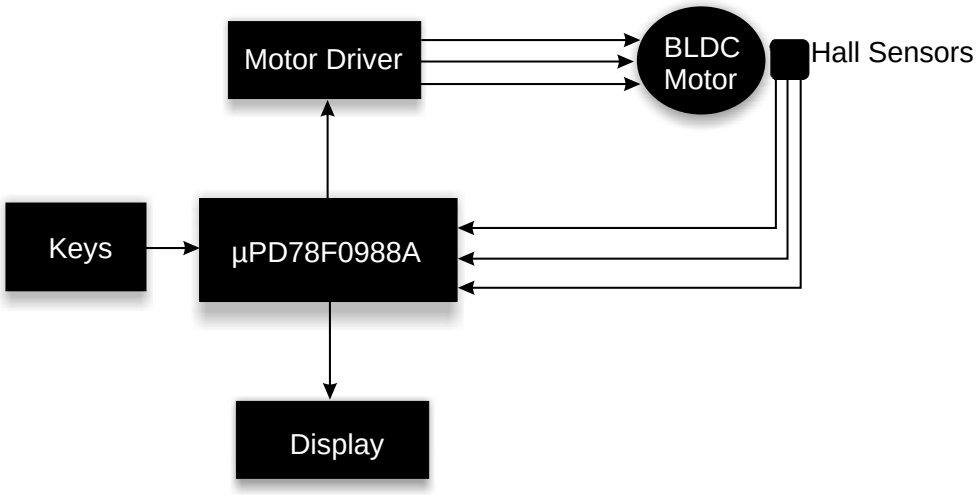
[MEMO]

Chapter 3 System Design Concept

3.1 System Concept

Figure 3-1 shows the principal block diagram of the system concept for the BLDC motor control.

Figure 3-1: Principal block diagram of the system configuration

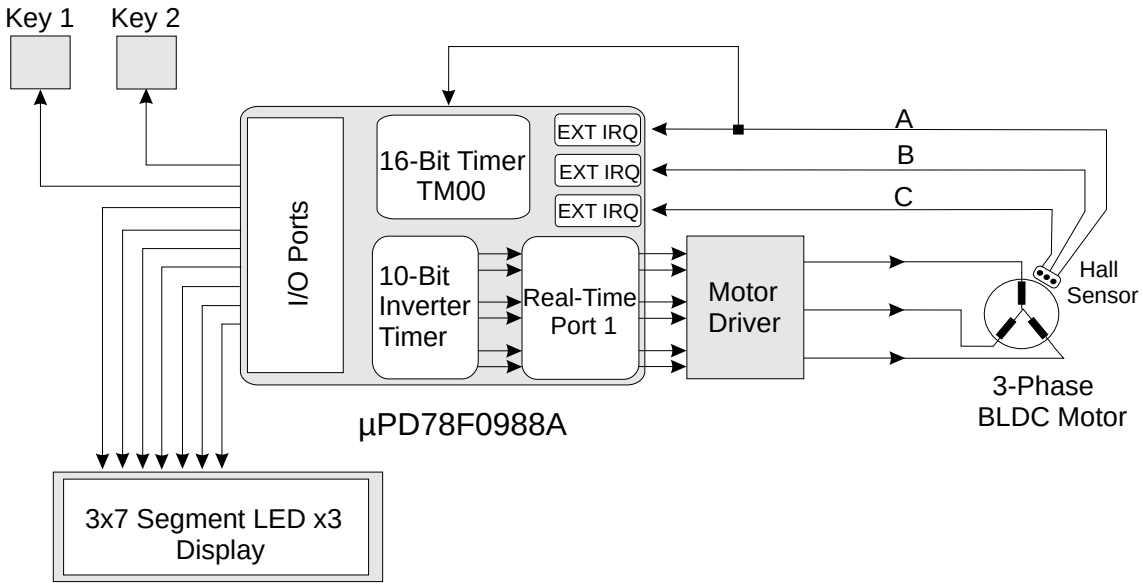


The µPD78F0988A processes the feedback from the sensor to control the motor driver that supplies the 3-phases of the BLDC motor. At the same time the speed of the BLDC motor is derived from the sensor signals and used to provide velocity feedback for the closed speed loop. The actual motor speed is indicated on the display.

3.2 System Configuration

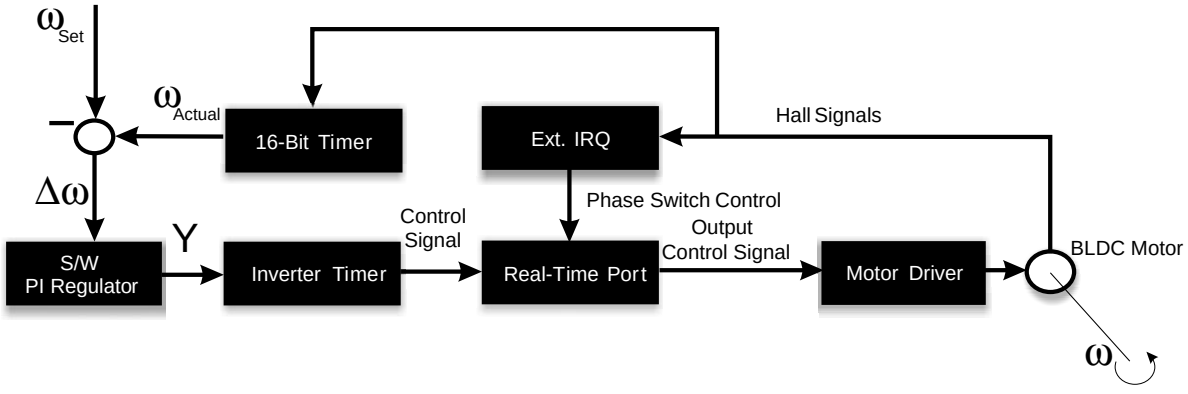
Figure 3-2 shows the system configuration and the peripherals of the µPD78F0988A device used for the BLDC motor control.

Figure 3-2: System Configuration with the peripherals of the µPD78F0988A



The rotor position information delivered from the Hall sensor of the BLDC motor is estimated with the 3-external interrupts of the device. The speed of the motor is measured with the 16-bit timer TM0 and the appropriate control signal for the BLDC motor is generated with the 10-bit inverter timer and the Real-Time Port 1of the device. The System configuration shows how the requirements of the BLDC motor control are fulfilled with the µPD78F0988A device. The function from each of the peripherals is described in the next chapter.
The system topology with the relationship between the hardware and software of the system is shown in Figure 3-3.

Figure 3-3: System topology and relationship between the control software and hardware of the system



Chapter 4 Hardware Configuration

4.1 µPD78F0988A Configuration

The µPD78F0988A device is a member of the high performance 78K Family 8-bit microcontrollers, designed specifically for mid-range motor control. The configuration of the device and the operating environment used in this application is listed below:

- CPU: µPD78F0988A
- Operating clock: System clock 8.38 MHz (incoming 12 MHz)
- Operating Voltage: 5 V
- Internal ROM: 60 Kbytes
- Internal RAM: 2 Kbytes
- External expansion memory: not used.

4.2 Peripherals I/O Assignments

Table 4-1 lists all pins of the µPD78F0988A device and the ones that are used in this application are described with their associated function.

Table 4-1: µPD78F0988A Peripheral I/O Assignment (1/2)

Pin No.	Pin Name	Mode setting	Function
1	P50	Input	Key Select
2	P51	Input	$\overline{SCK}$
3	P52	Input	SI
4	P53	Output	SO
5	P54	Input	Key Select
6	P55	Input	Speed measure
7	P56	Output	Common Display
8	P57	Output	Common Display
9	V_{SS0}		Ground
10	V_{DD0}		Supply Voltage
11	TO70	Output	Phase A
12	TO71	Output	Phase $\overline{A}$
13	TO72	Output	Phase B
14	TO73	Output	Phase $\overline{B}$
15	TO74	Output	Phase C
16	TO75	Output	Phase $\overline{C}$
17	P20	Output	Common Display
18	P21	Output	Not used
19	P22	Output	Not used
20	P23	Output	Not used
21	P24	Output	Not used
22	P25	Output	Not used
23	P26	Output	Not used
24	V_{DD1}		Supply Voltage
25	AV_{SS}		Connect to V_{SS}
26	P17	Output	Not used
27	P16	Output	Not used
28	P15	Output	Not used
29	P14	Output	Not used
30	P13	Output	Not used
31	P12	Output	Not used
32	P11	Output	Not used
33	P10	Output	Not used
34	AV_{REF}		Connect to V_{DD}
35	AV_{DD}		Connect to V_{DD}

Table 4-1: µPD78F0988A Peripheral I/O Assignment (2/2)

Pin No.	Pin Name	Mode setting	Function
36	$\overline{\text{RESET}}$	Input	Reset Input
37	P02	Input	Ext. IRQ Hall C
38	P03	Input	Ext. IRQ Hall A
39	V_{PP}		Connect to V_{SS}
40	X2		System Clock
41	X1	Input	System Clock
42	V_{SS1}	Ground	
43	P00	Output	Not used
44	P01	Output	Not used
45	P30	Output	Not used
46	P31	Output	Not used
47	P32	Output	Not used
48	P33	Output	Not used
49	P34	Output	Not used
50	P35	Output	Not used
51	P36	Output	Not used
52	P37	Output	Not used
53	P64	Output	Not used
54	P65	Output	Not used
55	P66	Output	Not used
56	P67	Output	Not used
57	P40	Output	Segment Display
58	P41	Output	Segment Display
59	P42	Output	Segment Display
60	P43	Output	Segment Display
61	P44	Output	Segment Display
62	P45	Output	Segment Display
63	P46	Output	Segment Display
64	P47	Output	Segment Display

4.3 16-Bit Timer Function

As shown in the explanation of the hardware, the speed of the BLDC motor is measured with using the 16-bit timer TM00 of the µPD78F0988A Device.
The timer has the following operating modes:

- Interval Timer
  - Generates interrupt request at the preset time interval

- PPG mode
  - Outputs a square wave who's frequency and output pulse can be set freely

- Pulse width measurement
  - Measures the pulse width of an external input signal

- External event counter
  - Measures the number of pulses of an external input signal

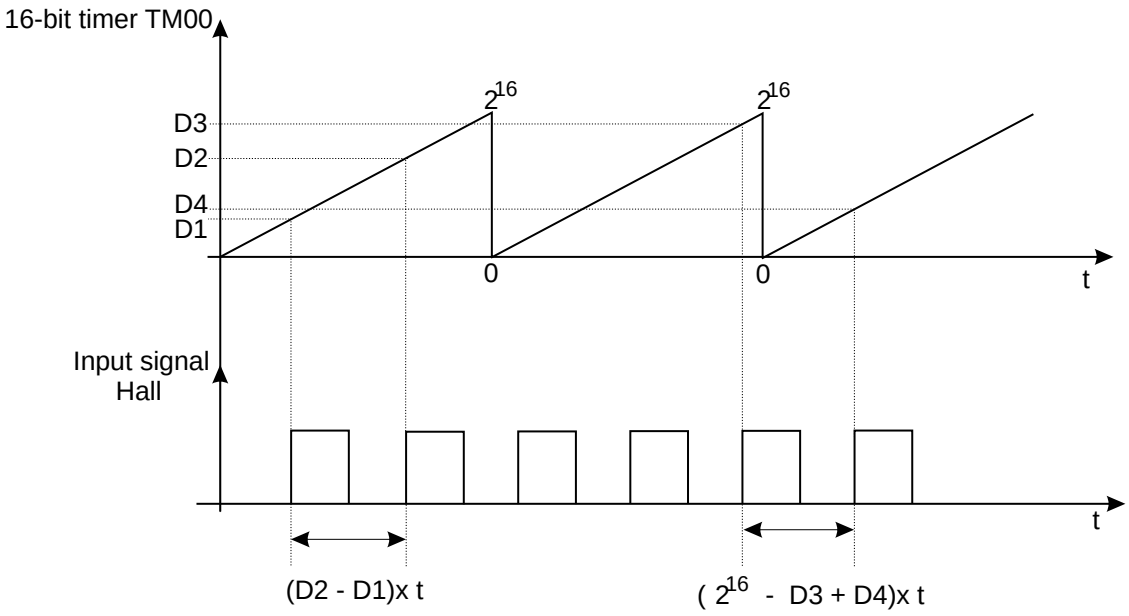
- Square-Wave output
  - Outputs a square wave with any selected frequency.

The pulse width measurement function of the timer was chosen to measure a motor speed that is in the range of 600 rpm…4800 rpm (= 10 Hz…80 Hz). Calculation using equation 2 produces the range of the hall sensor: 60 Hz…480 Hz. The count frequency of the timer, based on the hall sensor range, is set to 262 KHz.

The measurement process reacts to the rising edge of the input signal and calculates the time difference between the two rising edges of the input signal. The calculated difference is proportional to the frequency of the input signal.

Figure 4-1 describes the principal flow of the motor speed measurement with the 16-bit timer TM00. The values D1…D4 are the captured values in the CR000 capture register. The second register CR010 of the 16-bit timer is used as a compare register to detect the overflow of the timer during the measurement that has to be taken into consideration for the motor speed calculation.

Figure 4-1: Measurement Process of the 16-bit timer TM00

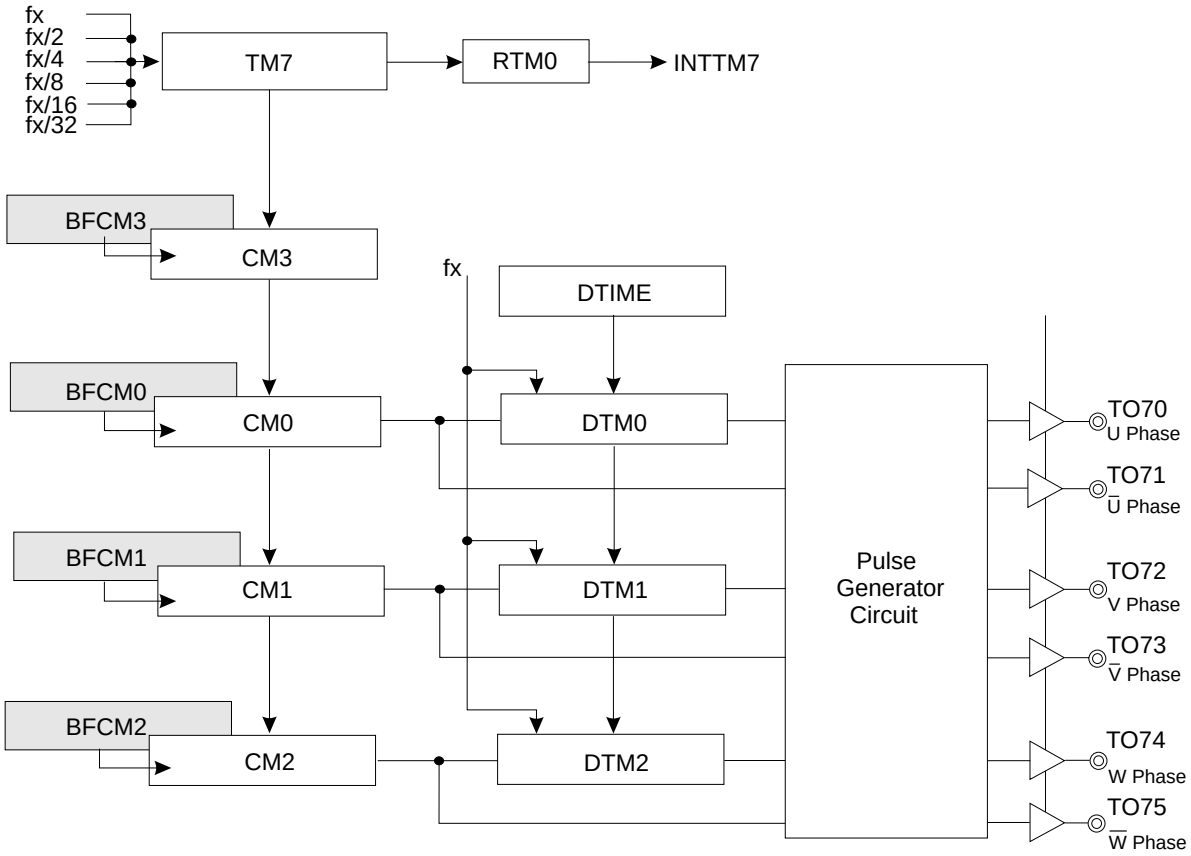


4.4 Inverter Control Timer Function

The inverter control timer TM7 is a 10-bit up/down counter that makes inverter control possible. It includes an 8-bit dead-time generation timer and allows generating of waveforms with non-overlapping active level. It generates six pair wise inverted PWM signals. Thus 3 positive and 3 negative output signals are generated. Figure 4-2 shows the configuration of the 10-bit inverter control timer (TM7).

The TM7 counts in synchronization with the rising edge of the count clock. The up/down count operation is defined with the value set in the CM3 compare register. Thus when the CM3 value matches the count value of the TM7, the timer TM7 is switched to count down operation until an underflow occurs (0000H value of the timer). The carrier frequency of the signal is set in the compare register CM3 and the dead time is set in the dead time setting register DTIME. The signal level conditions are defined with the set values in the three-compare register CM0, CM1 and CM2. The 10-bit buffer register BFCM0 to BFCM3 transfers the data into the compare registers (CM0 to CM3) with the timing of the interrupt request signal INTTM7.

Figure 4-2: 10-bit Inverter Control Timer Block Diagram



The buffer transfer control timer RTM0 is a 3-bit up counter and has the function of dividing the interrupt request signal INTTM7.

Figure 4-3 describes the operating timing of the inverter control timer TM7.

The calculation of the values needed for the definition of the desired output signal from the inverter control timer are performed with the following equations:

$$T_{TM7} = \frac{1}{f_X} \tag{3}$$

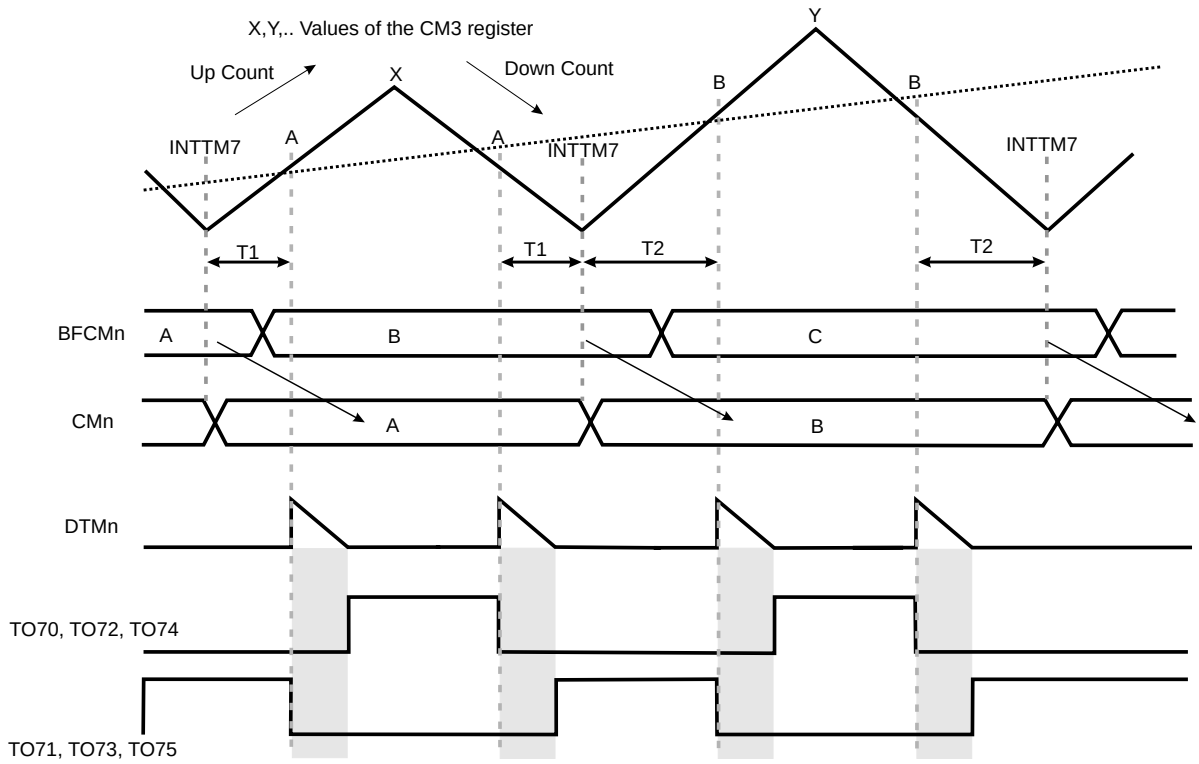
$$PWM_{CYCLE} = 2 \cdot T_{TM7} \tag{4}$$

$$DTM_{WIDTH} = (DTIME + 1) \cdot T_{TM7} \tag{5}$$

$$AWPP = [(CM3 - CM_{UP}) + (CM3 - CM_{DOWN})] \cdot T_{TM7} = 2 \cdot (CM3 - CM_X) \cdot T_{TM7} \tag{6}$$

$$AWNP = (CM_{DOWN} + CM_{UP}) \cdot T_{TM7} - T_{DTM} = 2 \cdot CM_X \cdot T_{TM7} - T_{DTM} \tag{7}$$

Figure 4-3: Operating timing of the inverter control timer TM7



f_X is the chosen system clock oscillation of the timer TM7. Equation (3) delivers the timer TM7 count clock and the equation (4) the PWM cycle of the signal.
The dead time between the active level of the signal pair, shown in Figure 4-3 with grey surface, is calculated with the equation (5). The equation (6) calculates the Active Width of the Positive Phase (AWPP) and the equation (7) the Active Width of the Negative Phase (AWNP) of the output signal.

4.5 Real- Time Output Port Function

The Real -Time output Port (RTP) transfers previously set data in the real-time buffer register to the output latch by hardware. The transfer is controlled with timer interrupts or external interrupt request generation. It is also possible to perform PWM modulation of a special pin with output pattern that can be specified in one bit unit.
The µPD780988 subseries has 2 channels of real-time output ports on chip. The RTP0 port is shared with Port 3 and RTP1 is shared with inverter control timer TM7. The real-time port used in this application is the RTP1 port. Therefore the function of the RTP1 port will be described in detail.

Figure 4-4: Block Diagram of the Real- Time Output Port RTP1

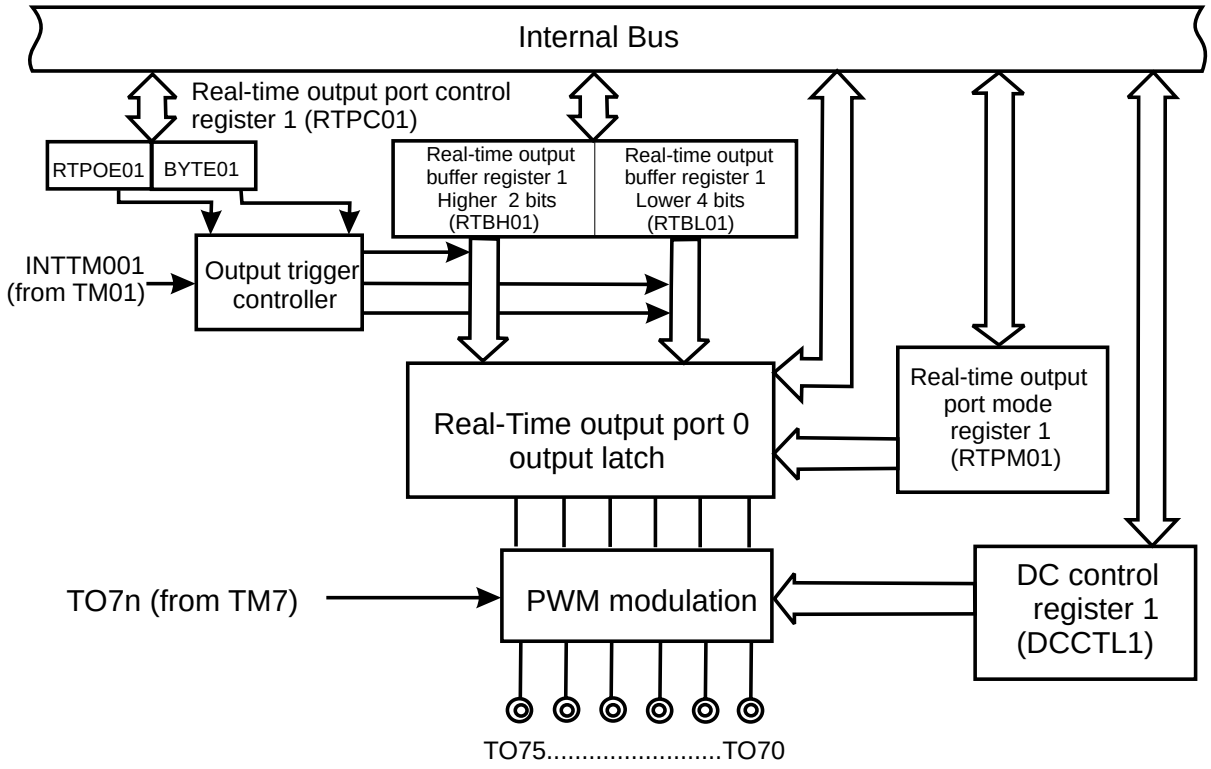


Figure 4-4 shows the block diagram of the real-time output port RTP1 that shares the output with the inverter control timer TM7.

The real-time output buffer register 1 (RTBH01, RTBL01) is the register that holds the data in advance. It is specified in entirely 6 bits that can be select either as 1 channel x 4 bits or 1 channel x 6 bits. The real time output mode is set with the port mode register RTPM01 that allows 1-bit units selection. The real-time output port control register RTPC01 sets the operating mode, enables/disables the operation of the real-time output port. The DC control register DCCTL1 controls the PWM modulation, enabling/ disabling of the output waveform inversion.

The relationship between the register settings of the real-time output port and the effects on the output is described in the Table 4-2 bellow:

Table 4-2: Relationship between registers settings and output effects

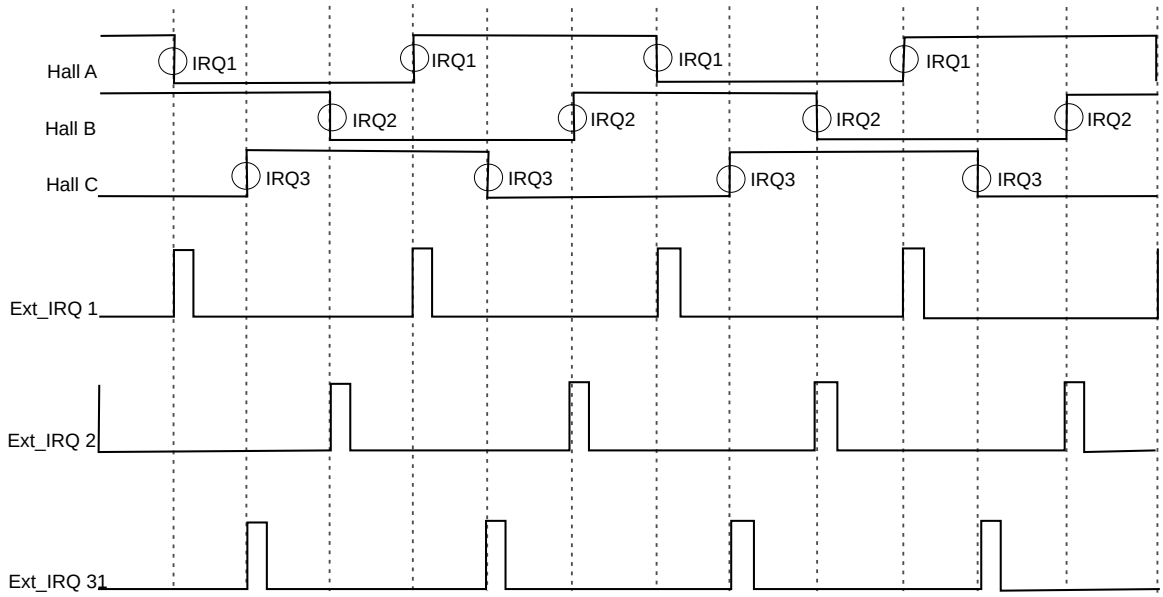
TMC7.7	DCCTL1.7	DCCTL1.4	DCCTL1.5 / DCCTL1.6	RTPC01.7	RTPM01.n	RTBH01 / RTBL01	Pin TO7n Status
0	x	x	x	x	x	x	Hi-Z
1	0	x	x	x	x	x	TO7n
	1	0	0	0	x	x	“low” output
				1	0	x	“low” output
					1	0	“low” output
						1	“high” output
			1	0	x	x	TO7n
				1	0	x	TO7n
					1	0	TO7n
						1	“high” output
		1	0	0	x	x	“high” output
				1	0	x	“high” output
					1	0	“high” output
						1	“low” output
			1	0	x	x	$\overline{\text{TO7n}}$
				1	0	x	$\overline{\text{TO7n}}$
					1	0	$\overline{\text{TO7n}}$
						1	“low” output

The interaction between the generated signal from the inverter control timer and the modulation of it with the real time output port makes generation of a wide range of signal wave forms possible. The solution of the signal generation for the control of the 3-phase inverter circuit that supplies the BLDC motor will be described in Chapter 5, where the software will also be introduced and described.

4.6 Interrupts Function

The µPD780988 Subseries includes several internal and external interrupt sources (see Table 1-1, “Functional Outline,” on page 12). In this application 3 external interrupts are used to detect the rotor position of the motor with the Hall sensors. The interrupts are maskable and are set to detect both edges of the input signal. The switching control of the real-time output port RTP1 that modulates the output signal from the inverter timer TM7 is than synchronized with the rotor position of the BLDC motor.
Figure 4-5 describes the generation of the external interrupt signal that is used to generate the control signal for the inverter circuit that supplies the BLDC motor synchronized with the Hall sensor signals.

Figure 4-5: External interrupts signal flow synchronized with the Hall sensor signals of the BLDC motor

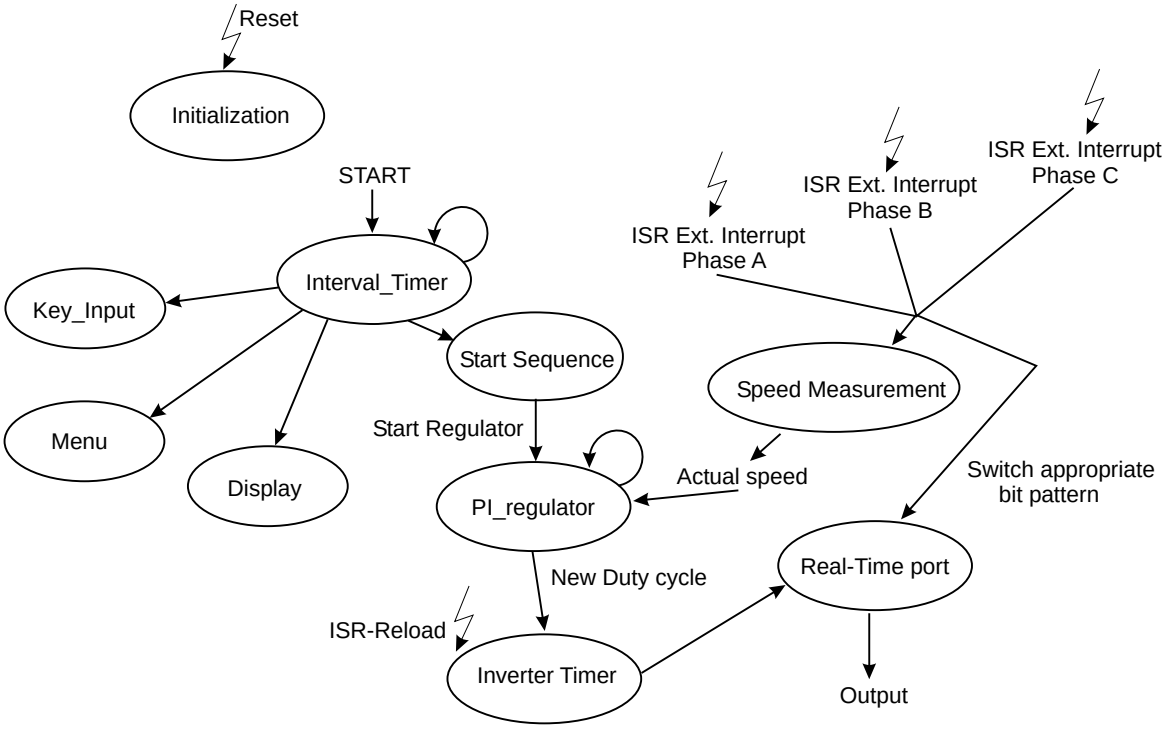


Chapter 5 Software Process Description

5.1 Data Flow Diagram

Figure 5-1 shows the principal data flow diagram and the relationship between the software modules and hardware peripherals that are involved in the control of the motor.

Figure 5-1: Principal Data Flow Diagram

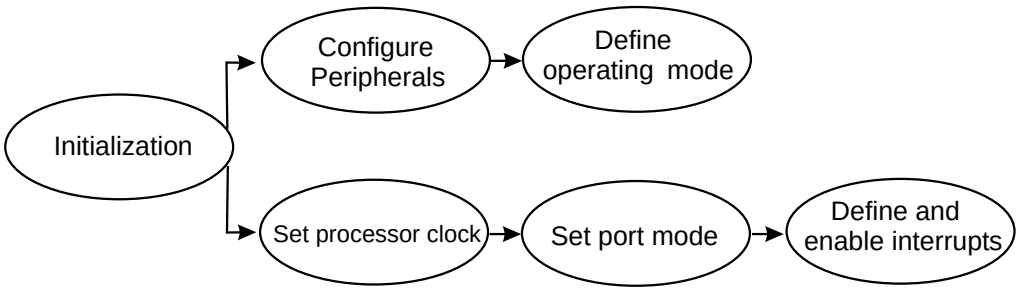


The external interrupts are the service routines with the highest priority in the system that can't be disturbed and controls the bit pattern of the real-time output port synchronised with input signals from the Hall sensor of the BLDC motor.
The other functions of the system shown in Figure 5-1 are sequential and implemented and executed in the main endless loop of the software.

5.2 Initialization

The initialization process is responsible for the initializing the µPD78F0988A device after a system reset. It configures the basic clock settings of the device, initializes the peripherals that are used for the motor control application and disables/ enables interrupts. The initialization contains two parts as shown in Figure 5-2, the first part that initializes the configuration of the device and the second part initialize the peripherals with their operating mode.

Figure 5-2: Initialization process



5.3 Interval_Timer

The watchdog timer is used to realize the interval timer function. It is used to generate an interrupt request at the preset time interval. The interval time length is set to the period of T = 977.6 µs. The interrupt request flag of the watchdog timer is polled and the function Interval_Timer is executed each time the interrupt request flag is detected high in the main loop.
The function Interval_Timer controls the execution of the key input function, the menu points function and the display function of the system. It's responsible also to start the BLDC motor running with the start sequence of the system at the start of running.

5.4 Key_Input, Menu, Display

These are the three functions that are responsible for getting the key inputs, to make the user able to make a choice between the menu points and to display the input and the actual rpm of the motor.

The Key_Input function is event controlled and it is executed only when a key entry is recognized. The sample time of the key entry is defined with the elapse time of the Interval_Timer function. The menu function is immediately executed when the key entry is recognized through the key input function.
The Display is refreshed every time with the executing of the interval timer function.

5.5 Start Sequence

The start sequence of the system is the function that allows a non-reflecting start of the motor. The concept of the function is to force an external interrupt that generates the appropriate switching of the real time output port. The generation of the external interrupt request depends on the start position of the rotor. As already described there are six possible Hall sensor positions that describe the rotor position. The start sequence derives three absolute start positions from these six positions that correspond to the external interrupts. The sequence proceeds upon the detection of the actual level of the input pins of the external interrupts, to force the next incoming external interrupt that switch the real time port. The table below describes the proceeding of the start sequence in both turn directions of the motor:

Right Direction

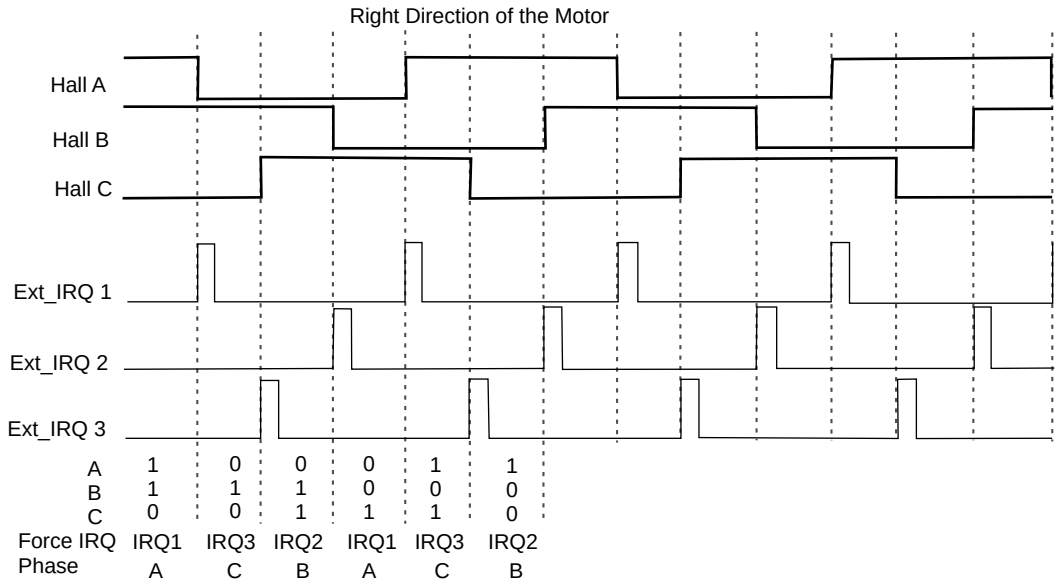
	Hall A	0	0	0	1	1	1
	Hall B	1	1	0	0	0	1
	Hall C	0	1	1	1	0	0
Start Phase		C	B	A	C	B	A

Left Direction

	Hall A	1	1	1	0	0	0
	Hall B	1	0	0	0	1	1
	Hall C	0	0	1	1	1	0
Start Phase		B	C	A	B	C	A

Figure 5-3 describes the connection between the six Hall sensor positions and the three derived results to force the appropriate external interrupt and describes also how the table described above has been used.

Figure 5-3: Start sequence



The left turn direction of the motor can be derived from the left turn direction table of the motor also described above.

5.6 Speed Measurement

The Hall_Measurement function has the task to measure the motor speed. The measured value is used as a feedback for the closed speed loop control.
The 16-bit Timer TM0 is used in the pulse width measurement mode and it's defined to react on the rising edge of the input signal from one chosen Hall sensor signal of the motor. The function is executed every time when one of the three interrupt service routines is active. Accordingly the elapsed time of the function, depending from the motor speed, is synchronized with the frequency of the three interrupt service routines which are modified, as already described in Chapter 4.6 "Interrupts Function" on page 30, to detect both edges of the input Hall signal from the motor. Thus the sample time of the function is equal with the actual motor speed.

5.7 PI-Regulator

The PI-Regulator used is the classical Proportional Integral (PI) control method in the closed loop control of the BLDC motor.

The regulator is based on the recursive PI algorithm known also as the speed algorithm and takes the form of:

$$G(s) = K_P + K_I \times \frac{1}{S}$$

Transformed into a discrete form:

$$K_P \times X_D + K_I \times (\Sigma X_D)$$

$$X_D = X(n) - X(n - 1)$$

where:

K_P	presents the proportional gain
K_I	presents the integral gain
X_P	presents the speed error
ΣX_D	presents the accumulated speed error

The coefficients K_P and K_I were derived empirically and optimized based on system behaviour produced by disturbances during the system testing.

The sample time of the regulator is set to 30 ms. The duration of the regulator execution time includes also two additional operations that have the task to normalise the value and to transform the calculated regulated quantity into the duty cycle of the PWM signal. The normalised value transforms the actual count (= motor speed) of the 16-bit timer TM00 into the range of the regulator which is defined as a 10-bit range.

[MEMO]

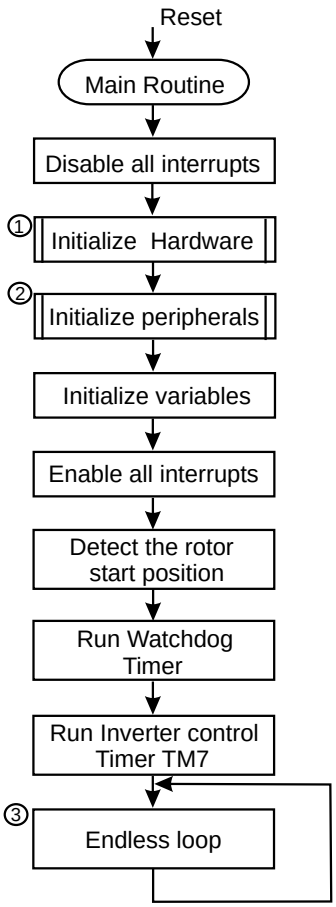
Chapter 6 Software Flow Charts

This chapter describes the important functions used in the system of the BLDC motor control application. The functions that are responsible for the key input, the display, the menu points, the normalising and transform values functions are not included in this chapter. Please refer in the software source codes if more information about these functions is needed.

6.1 Concept and Main Flow Diagram

Figure 6-1 shows the main program flow chart.

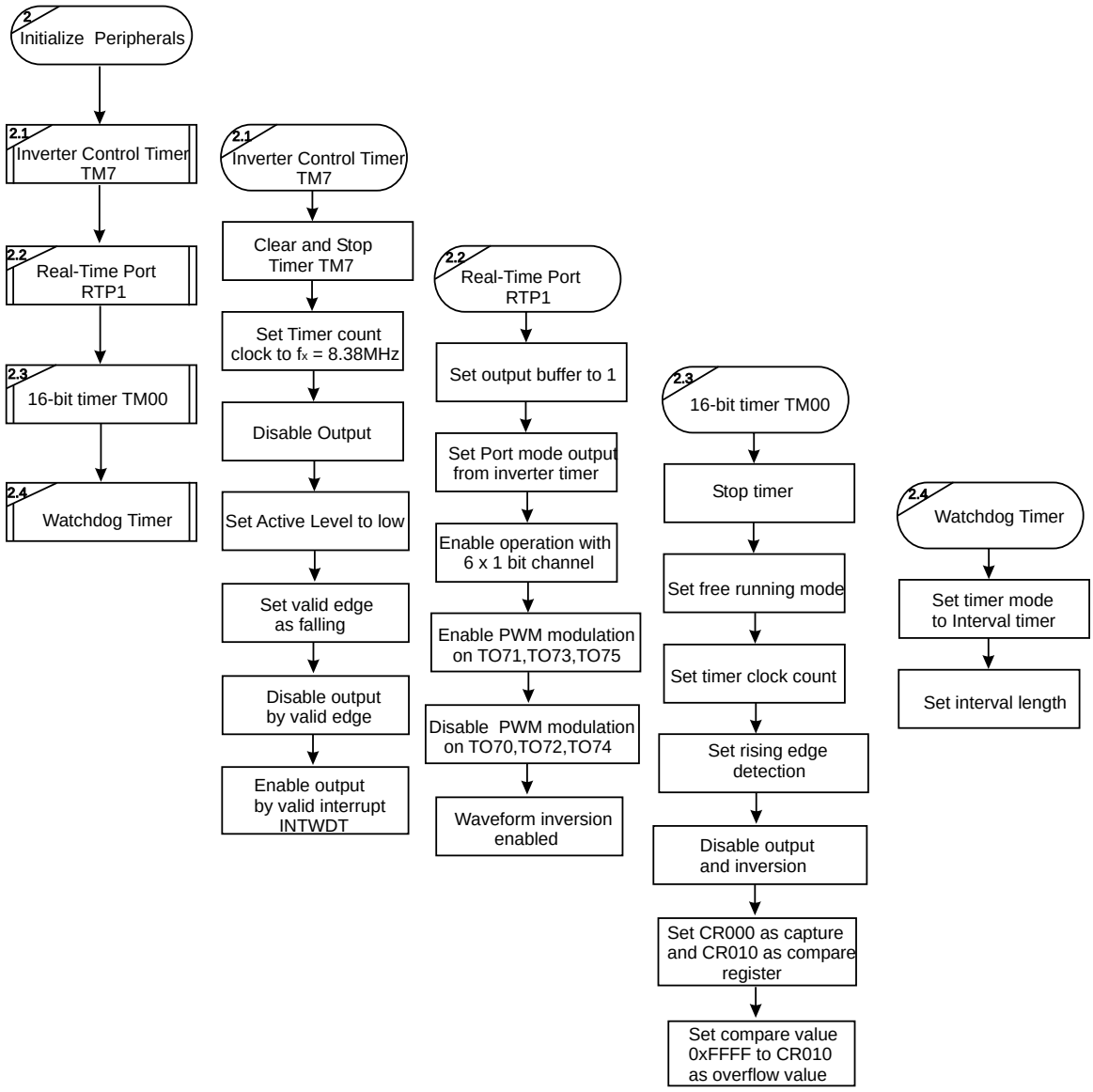
Figure 6-1: Main Program Flowchart



6.2 Peripherals initialization

Figure 6-2 shows the initialization of the used hardware peripherals of the µPD78F0988A device with their operation mode in this application.

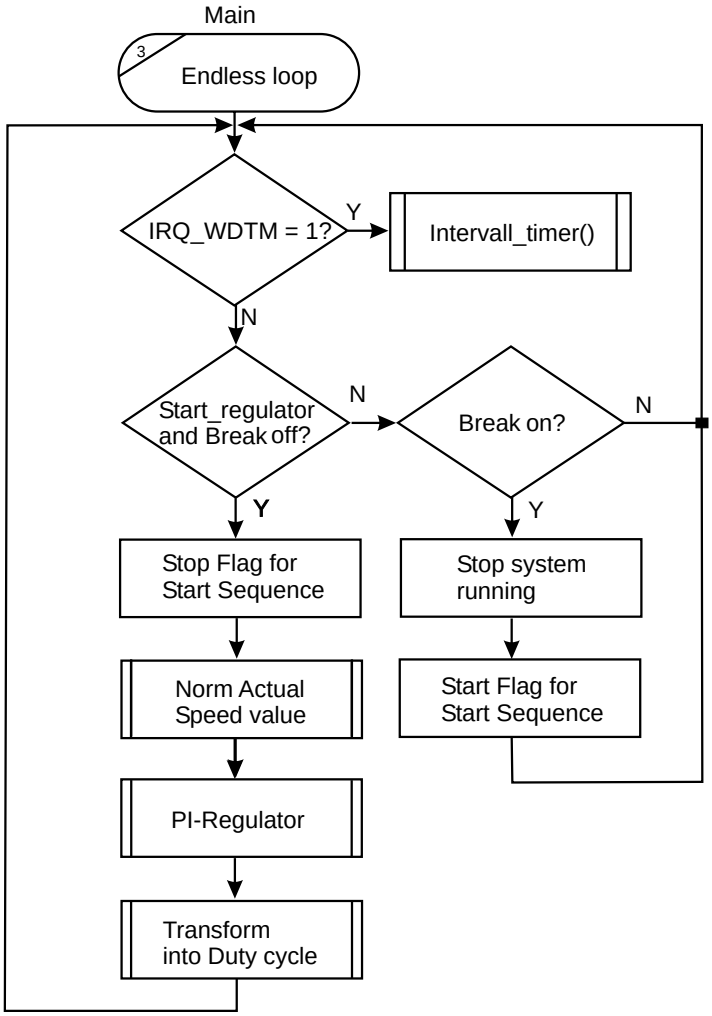
Figure 6-2: Peripherals initialization



6.3 Main Concept

Figure 6-3 shows the endless loop of the main program used in this application.

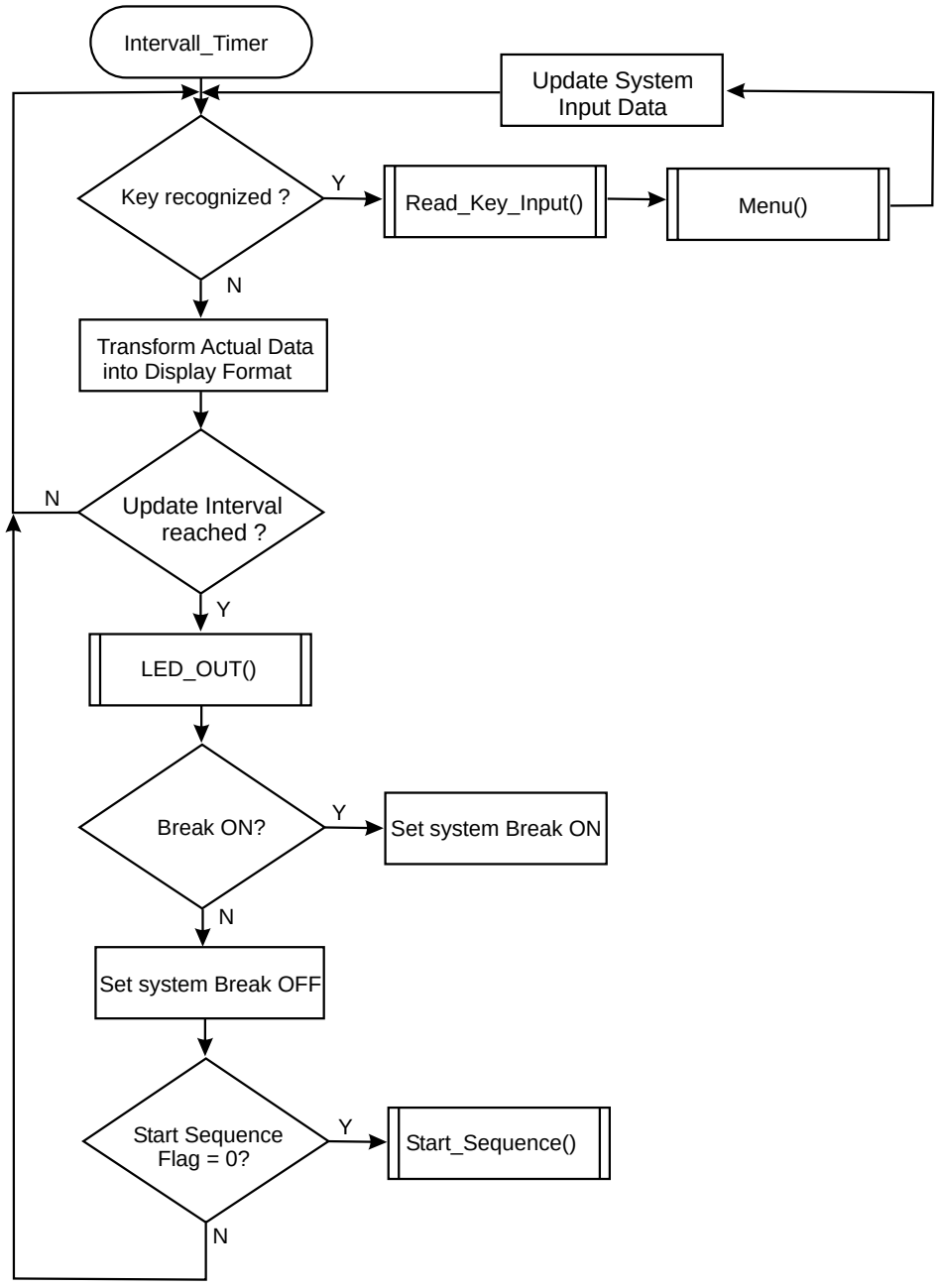
Figure 6-3: Endless Loop function flow



6.4 Interval Timer

Figure 6-4 shows the Interval timer function flow realized with the watchdog timer of the µPD78F0988A device in this application.

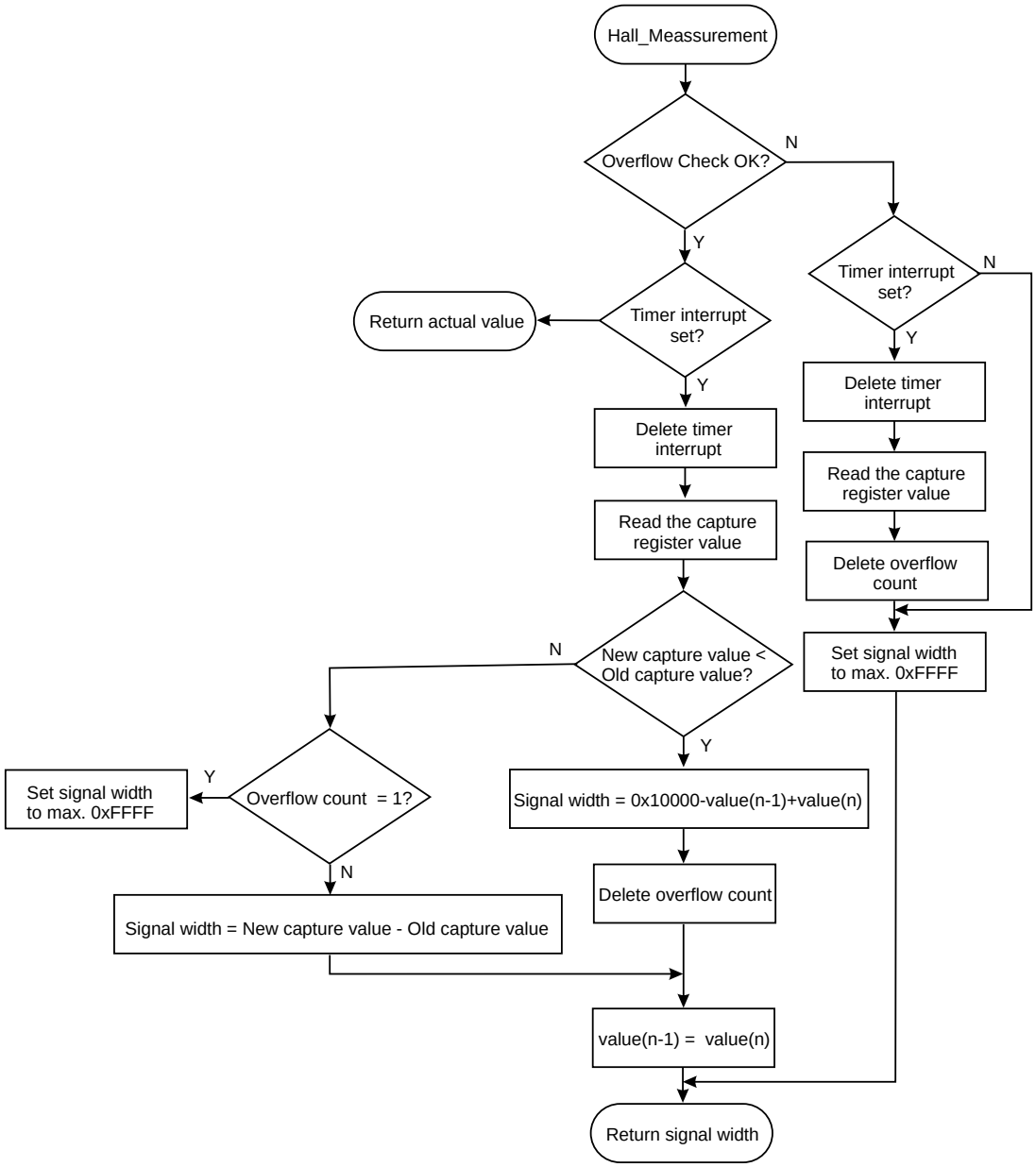
Figure 6-4: Interval timer function flow



6.5 Speed Measurement

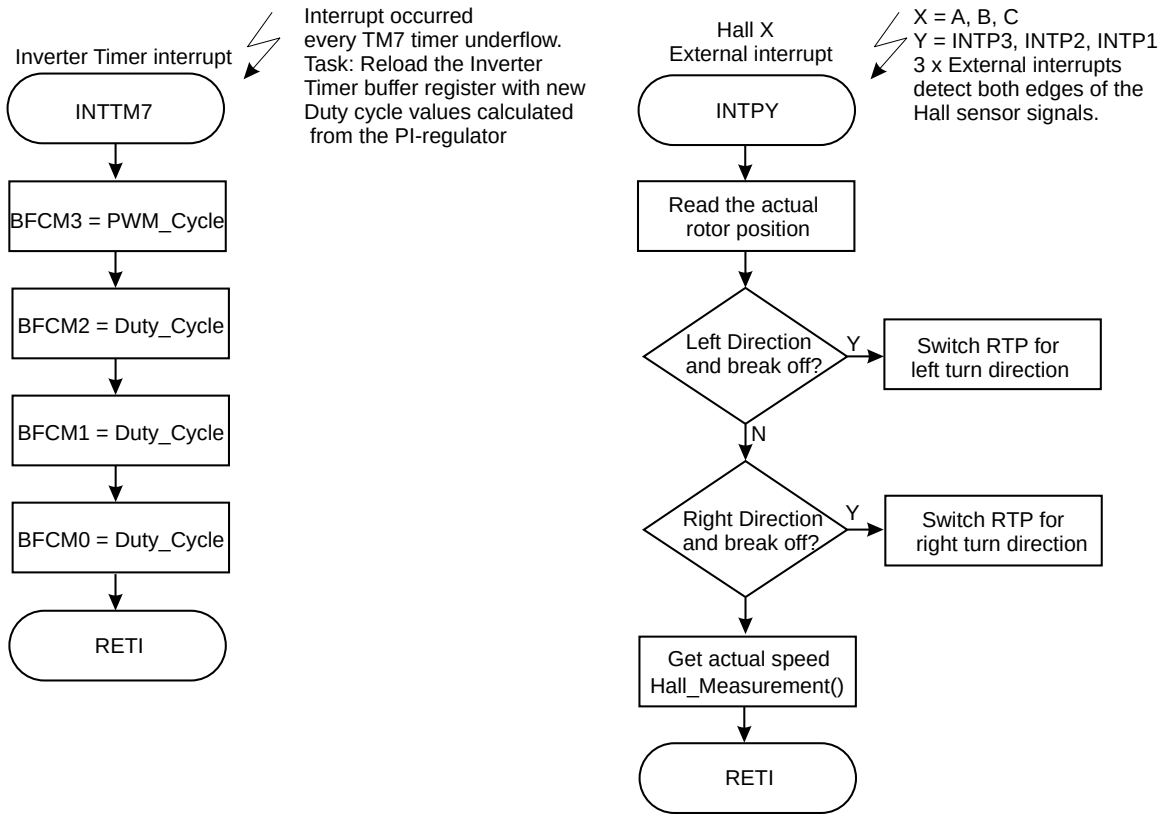
Figure 6-5 shows the speed measurement function flow realised with the 16-bit timer TM00.

Figure 6-5: Speed measurement flow



6.6 Control Signal Generation

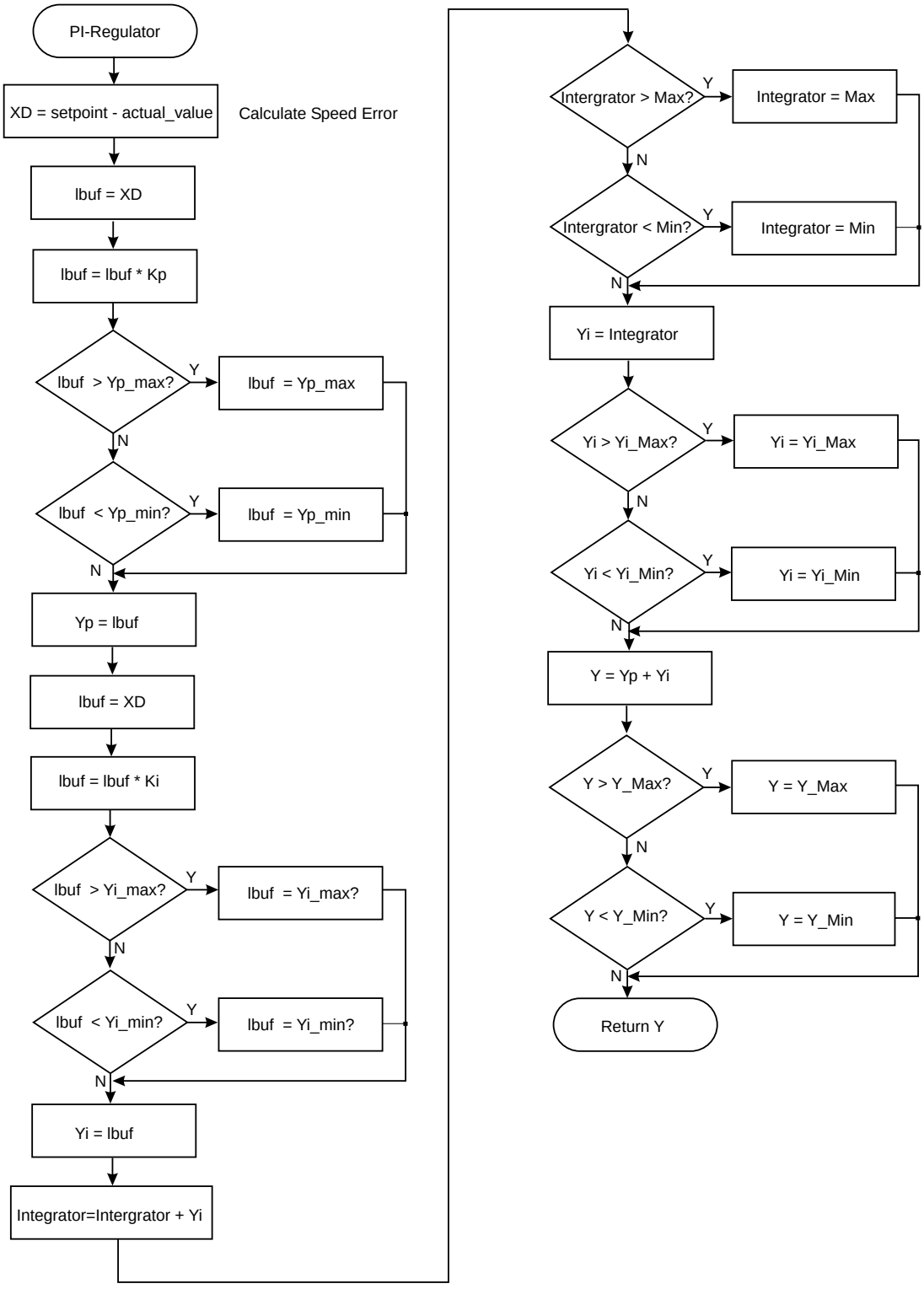
Figure 6-6: Signal generation flow in synchronization with the rotor position of the BLDC motor



6.7 PI-Regulator

Figure 6-7 shows the function flow of the implemented PI-Regulator in the system.

Figure 6-7: PI-Regulator function flow



[MEMO]

Chapter 7 Program Listing

```
/*============================================================================================
**          The listed example source code in this chapter was developed in cooperation with the Expert Assistant Manager
**          Mr. Ludger Lenzen.
** PROJECT     = Motor control
** MODULE      =  hdwinit()
** VERSION     = V0.1
** DATE        = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** ============================================================================================
** Description: Module initializes the UPD78F0988A Hardware
** ============================================================================================ */

void hdwinit (void)
  {
  // processor clock
        PCC  = 0x00;          // with speed
        OSTS = 0x04;

  // port latch

        P0=0x00;             // set output latch to 0
        P2=0x00;             // set output latch to 0
        P3=0x00;             // set output latch to 0
        P4=0x00;             // set output latch to 0
        P5=0x00;             // set output latch to 0
        P6=0x00;             // set output latch to 0

  // port mode
        PM0  = 0x0E;         // port 0 = input  ---> EXT-IRQ1, EXT-IRQ2, EXT-IRQ3 to detect the Rotor Pos
                             // port 1 is only input
        PM2  = 0x00;         // port 2 = output (P2.0 = DIG1 !!!!)
        PM3  = 0x00;         // port 3 = output
        PM4  = 0x00;         // port 4 = output
        PM5  = 0x31;         // port 5.5 = TM01 input and 5.0 = Key Up and 5.4 = Key Down input
        PM6  = 0x00;         // port 6 = output

  // pull up resistors
        PU0 = 0x00;          // no pull up-resistors
        PU2 = 0x00;          // no pull up-resistors
        PU3 = 0x00;          // no pull up-resistors
        PU4 = 0x00;          // no pull up-resistors
        PU5 = 0x11;          // pull up-resistors for the P5.0= Key Up and P5.4= Key Down Pin
        PU6 = 0x00;          // no pull up-resistors

  // interrupt definition
        IF0L = 0x00;         // INT request
        IF0H = 0x00;         // INT request
        IF1L = 0x00;         // INT request
                             // 7 6 5 4 3 2 1 0
        MK0L = 0xE3;         // 1 1 1 0 0 0 1 1  EXT 1,2,3 enable
        MK0H = 0xFD;         // 1 1 1 1 1 1 0 1  INT INTTM001 & TMMK7 enable
        MK1L = 0xFF;         // 1 1 1 1 1 1 1 1  INT not masked
```

```
    PR0L = 0xFF;          // INT low priority
    PR0H = 0xFF;          // INT low priority
    PR1L = 0xFF;          // INT low priority


    EGP = 0x0F;           // Rising  & Falling Edge
    EGN = 0x0F;           // Only Rising => EGN = 0x00
}
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  Inverter_INIT()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =

** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
**
** =============================================================================================
** Description: Module initializes the 10-bit Inverter timer of the UPD78F0988A
** ============================================================================================= */
void Inverter_INIT(void)
  {
  // Inverter timer control register format
  // ---------------------------------------------------------------------
                    // 7 6 5 4 3 2 1 0 Bit Nummber
        TMC7 = 0x00;      // 0 0 1 0 1 0 0 0
                    // 7-> 0/1 => Clear and Stop / Count enable
                    // 6-> 0   => Must be Zero
                    //           5 4 3
                    //         => 0 0 0 -> 0x00 => fclockcount = fx = 8,38 MHz
                    //         => 0 0 1 -> 0x08 => fclockcount = fx = 4.19 MHz
                    //         => 0 1 0 -> 0x10 => fclockcount = fx = 2.1  MHz
                    //         => 0 1 1 -> 0x18 => fclockcount = fx = 1.05 MHz
                    //         => 1 0 0 -> 0x20 => fclockcount = fx = 524  KHz
                    //         => 1 0 1 -> 0x28 => fclockcount = fx = 262  KHz
                    // 2,1,0   => INTTM7 occures every TM7 underflows

  // Inverter Timer mode Register
  //----------------------------------------------------------------------------

        TMM7 = 0x00;      // 7 6 5 4 3 2 1 0 Bit Nummber
                    // 0 0 0 1 0 0 0 0
                    // 7-> 0   => Must be Zero
                    // 6-> 0   => Must be Zero
                    // 5-> 0   => Must be Zero
                    // 4-> 0/1 => TM7 output disabled status/TM7 output enabled status
                    // 3-> 0/1 => Low/High TO70...TO75 output active level specification
                    // 2-> 0/1 => Falling/Rising TOFF7 valid edge
                    // 1-> 0/1 => Output not stoppet/Output stopped by valid EDGE
                    // 0-> 0/1 => Output not stoppet/Output stopped by valid INTWDT

        bALV = 0;         // ALV = Output active spec.
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  Real_Time_INIT()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
**
** =============================================================================================
** Description: Module initializes the Real-Time Otput port RTP1 of the UPD78F0988A
** ============================================================================================= */

void Real_Time_INIT(void)
  {
  // Real-Time Output Buffer Register
  //-------------------------------------------------------------------------
        RTBL01 = 0x0F;          // real-time output buffer register low

        RTBH01 = 0x30;          // real-time output buffer register high

  // Real-Time Output Port Mode Register 1
  //-------------------------------------------------------------------------

        RTPM01 = 0x3F;          // real-time output port mode register
                                // Output is TO70 to TO75 !!!!
                                // Bit 7 & 6 must be set to Zero !!!!

  // Real-Time Output Port Control Register 1
  //-------------------------------------------------------------------------
                                // 7 6 5 4 3 2 1 0
        RTPC01 = 0xA0;          // 1 0 1 0 0 0 0 0
                                // |___|_____Enables Operation and 6-Bits x 1 Channel
                                // real-time output port control register
                                // Output is TO70 to TO75 !!!!

  // DC control Resgister 1 (DCCTL1)
  //-------------------------------------------------------------------------
                        // 7 6 5 4 3 2 1 0 Bit Nummber
         DCCTL1 = 0x90;     // 1 0 0 1 0 0 0 0
                        // 7-> 0/1 => Inverter Timer Output/PWM Modulated RTP Output
                        // 6-> 0/1 => PWM Modulation disabled/enabled on TO70,TO72,TO74
                        // 5-> 0/1 => PWM Modulation disabled/enabled on TO71,TO73,TO75
                        // 4-> 0/1 => Inversion disabled/enabled => Output Waveform specification
         TM7_PWML = 1; // PWM Modulation enabled on TO71,TO73,TO75
         TM7_PWMH = 0; // PWM Modulation disabled on TO70,TO72,TO74

        //TM7_RTP = 0;

         bINV = 1;
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  TM00_INIT()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Module initializes the 16-bit timer TM00 of the UPD78F0988A
** ============================================================================================= */

void TM00_INIT(void)
  {
  TMC00 = 0x00;          // Stop operation

  PRM00 = 0x12;          // TI000 rising edge detection and count clock = 262KHz

  TOC00 = 0x00;          // inversion disabled, F/F no change, output disabled
                         // inversion enabled by match of TM00 and CR010, Timer output reset
                         // ->F/F set to zero & output disabled

  CRC00 = 0x01;          // CR000 capture on TIO10 valid edge and CR010 as compare register

  CR010 = 0xFFFF;        // compare value -> OVF detection!

  TMC00 = 0x06;          // Free-running mode; IRQ generated on match between TM00 and CR000,
                         // or match between TM00 and CR010
  IRQ_TM00 = 0;          // clear IRQ for security
  IRQ_OVF = 0;
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  WDTM_INITt()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Module initialize the Watch-Dog  timer of the UPD78F0988A as Interval Timer
** ============================================================================================= */

void WDTM_INIT(void)
  {
  // WDCS = 0x00 --> 488.8 microsec
  //           = 0x01 --> 977.6 microsec
  //           = 0x02 --> 3.91  msec
  //           = 0x03 --> 7.82  msec
  //           = 0x04 --> 15.6  msec
  //           = 0x05 --> 31.3  msec
  //           = 0x06 --> 125.1 msec

  WDCS = 0x01; // Intervall Timer
  WDTM = 0x00; // Intervall Timer mode
  }
```

```
/*============================================================================================
** PROJECT      = Motor control
** MODULE       =  Regulator()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**            NEC Electronics (Europe) GmbH
**            Technical Product Support
**            Semiconductors and Displays Business Unit
**            D-40472 Düsseldorf, Germany
** ============================================================================================
** Description: PI- Regulator
** ============================================================================================ */
int Regulator(void)
  {
  /*      PI Regulator
  Range of values:

  setpoint          =    0%...+100% = (integer) =          0...+4400
  actual_value   =    0%...+100% = (integer) =          0...+4400
  XD               = -100%...+100% = (integer) =  -4400...0...+4400

  Yp              = -100%...+100% = (integer) =  -4096...0...+4095
  Yi              = -100%...+100% = (integer) =  -4096...0...+4095
  Y               =    0%...+100% = (integer) =          0...+4095
  Integrator   = -400%...+400% = (integer) = -16384...0...+16383
  Kp              = -100%...+100% = (integer) =  -4096...0...+4095
  Ki              = -100%...+100% = (integer) =  -4096...0...+4095
  */

  /* local variables */
  static long lbuf; // local long type calculation buffer
  int XD;             // Delta_X
  long Yp;           // Y proportional part
  long Yi;           // Y integral part
  long Y;            // Y result

  /* value range limits */
  #define Yp_max          1048575l     // +4095 * 256
  #define Yp_min         -1048575l     // -4095 * 256
  #define Yi_max          4194303l     // +4095 * 1024
  #define Yi_min         -4194303l     // +4095 * 1024
  #define Integrator_max  4194303l  // +4095 * 1024
  #define Integrator_min -4194303l  // -4095 * 1024
  #define Y_max           4194303l     // +4095 * 1024
  #define Y_min                 0l            //         0 * 1024

  // calculate XD
  XD = setpoint - actual_value;

  // calculate Yp and limit
  // Note: Kp = kp * 256
  // calculate Yp = XD * Kp;
  lbuf = XD;
  lbuf = lbuf * Kp; // Yp (long)

  // limit Yp
  if (lbuf > Yp_max)
    {
    lbuf = Yp_max;
    }
  else
```

```
  {
  if (lbuf < Yp_min)
    {
    lbuf = Yp_min;
    }
  }
Yp = lbuf; // Yp = yp * 256

// calculate Yi and limit
// note: Ki = ki * 1024
// calculate Yi(t) = XD * Ki;
lbuf = XD;
lbuf = lbuf* Ki;// Yi(t) (long)

// limit Yi(t)
if (lbuf > Yi_max)
  {
  lbuf = Yi_max;
  }
else
  {
  if (lbuf < Yi_min)
    {
    lbuf = Yi_min;
    }
  }
Yi = lbuf;

// update integrator
Integrator = (Integrator + Yi);

// limit Integrator
if (Integrator > Integrator_max)
  {
  Integrator = Integrator_max;
  }
else
  {
  if (Integrator < Integrator_min)
    {
    Integrator = Integrator_min;
    }
  }
Yi = Integrator;

// limit Yi (Yi = yi * 1024)
if (Yi > Yi_max)
  {
  Yi = Yi_max;
  }
else
  {
  if (Yi < Yi_min)
    {
    Yi = Yi_min;
    }
  }
// calculate Y and limit */
// note; Yi = yi * 1024, Yp = yp * 256
// calculate Y = yp * 1024 + yi * 1024
Y = Yp + Yi;

// limit Y
if (Y > Y_max)
```

```
    {
    Y = Y_max;
    }
  else
    {
    if (Y < Y_min)
      {
      Y = Y_min;
      }
    }
return (int)(Y >> 10); // return y = Y / 1024
 }
```

```
/*============================================================================================
** PROJECT      = Motor control
** MODULE       =  Speed_Measurement()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
** ============================================================================================
** Description: Speed measurement of the motor
** ============================================================================================ */

unsigned int Speed_Measurement( unsigned int cnt )
  {

  // Overflow detection
  if (IRQ_OVF)
    {
    IRQ_OVF = 0;
    ovf_one++;
    if (ovf_one > 3)
      {
      ovf_one = 3;
      }
    }
  if (ovf_one > 1)
    {
    if (IRQ_TM00)
      {
      IRQ_TM00 = 0;
      value_old_one = CR000;// Read the value on the rising edge
      ovf_one = 0;
      }
    signalwidth_one = 0xFFFF;
    }
  else
    {
    // Check timer and Calculate signal width
      if (IRQ_TM00)
      {
      IRQ_TM00 = 0;
      value_new_one = CR000;// Read the value on the rising edge
      if (value_new_one < value_old_one)
        {
        signalwidth_one = (unsigned int)(0x10000 - value_old_one + value_new_one);
        }
      else
        {
        if (ovf_one)
          {
          signalwidth_one = 0xFFFF;
          }
        else
          {
          signalwidth_one = value_new_one - value_old_one;
          }
        }
      ovf_one = 0;
      value_old_one = value_new_one;
      }
```

```
    else
      {
      return cnt;
      }
    }
return signalwidth_one;   }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  ISR Phase_A()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**            NEC Electronics (Europe) GmbH
**            Technical Product Support
**            Semiconductors and Displays Business Unit
**            D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Detection of the Hall A sensor from the motor and switching of the bit pattern for the RTP port
** ============================================================================================= */

interrupt [INTP3_vect] void Phase_A(void) //P00 -> 38
  {
  Motor_Pos = P0 & 0x0E; //Detect Actuall Rotor Position

  //LEFT TURN
  if (!Motor_Dir && !BREAK)
    {
      RTPM01 = RTP_SWITCH_LEFT[Motor_Pos];

    //RIGHT TURN
    }
  else
    {
    if (Motor_Dir && !BREAK)
      {
      RTPM01 = RTP_SWITCH_RIGHT[Motor_Pos];
      }
    }
  actual_count = Speed_Measurement( actual_count );
  }
```

```
/*=========================================================================================
** PROJECT      = Motor control
** MODULE       =  ISR Phase_B()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** =========================================================================================
** Description: Detection of the Hall B sensor from the motor and switching of the bit pattern for the RTP port
** ========================================================================================= */

interrupt [INTP1_vect] void Phase_B(void) //P01 -> 37
  {
  Motor_Pos = P0 & 0x0E; //Detect Actuall Rotor Position

  //LEFT TURN
  if (!Motor_Dir && !BREAK)
    {
     RTPM01 = RTP_SWITCH_LEFT[Motor_Pos];
    //RIGHT TURN
    }
  else
    {
    if (Motor_Dir && !BREAK)
      {
      RTPM01 = RTP_SWITCH_RIGHT[Motor_Pos];
      }
    }
  actual_count = Speed_Measurement( actual_count );
  }
```

```
/*=========================================================================================
** PROJECT      = Motor control
** MODULE       =  ISR Phase_C()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =========================================================================================
** Description: Detection of the Hall C sensor from the motor and switching of the bit pattern for the RTP port
** ========================================================================================= */

interrupt [INTP2_vect] void Phase_C(void) // P01 -> 44
  {
  Motor_Pos = P0 & 0x0E; //Detect Actuall Rotor Position

  //LEFT TURN
  if (!Motor_Dir && !BREAK)
    {
       RTPM01 = RTP_SWITCH_LEFT[Motor_Pos];
    //RIGHT TURN
    }
  else
    {
    if (Motor_Dir && !BREAK)
      {
      RTPM01 = RTP_SWITCH_RIGHT[Motor_Pos];
      }
    }
  actual_count = Speed_Measurement( actual_count );
  }
```

```
/*=========================================================================================
** PROJECT      = Motor control
** MODULE       =  ISR TM7exception()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** =========================================================================================
** Description: Reload inverter timer TM7with new values
** ========================================================================================= */

interrupt [INTTM7_vect] void TM7exception (void)
  {
  BFCM3 = PWM_CYCLE;  //BFCM3_value = PWM Cycle;
  BFCM2 = DUTY_CYCLE; //BFCM2_value = Duty Cycle TO74, TO75
  BFCM1 = DUTY_CYCLE; //BFCM1_value = Duty Cycle TO72, TO73
  BFCM0 = DUTY_CYCLE; //BFCM0_value = Duty Cycle TO70, TO71
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  Start_Sequence()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Start of the motor
** ============================================================================================= */

void Start_Sequence(void)
  {
  //Start motor left or right */
  if (Motor_Dir  //Right turn
     )
     {
     if ((Motor_Pos == 0x04) || (Motor_Pos == 0x0A)  // 4 & 10
        )
        {
        IF0L.4 = 1; // Start Phase A
        }
     if ((Motor_Pos == 0x06) || (Motor_Pos == 0x08) // 6 & 8
        )
        {
        IF0L.2 = 1; // Start Phase B
        }
     if ((Motor_Pos == 0x02) || (Motor_Pos == 0x0C) // 2 & 12
        )
        {
        IF0L.2 = 3; // Start Phase C
        }
     }
  else
     {
     if (!Motor_Dir//Left turn
        )
        {
        if ((Motor_Pos == 0x02) || (Motor_Pos == 0x0C))
           {
           IF0L.4 = 1; // Start Phase A
           }
        if ((Motor_Pos == 0x04) || (Motor_Pos == 0x0A))
           {
           IF0L.2 = 1; // Start Phase B
           }
        if ((Motor_Pos == 0x06) || (Motor_Pos == 0x08))
           {
           IF0L.3 = 1; // Start Phase C
           }
        }
     }
  START_REGULATOR = 1;
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  Interval_Timer()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Key Input, Menu, Display and start sequence control
** ============================================================================================= */

void Interval_Timer (void)
  {
  // Display und Key_input run
  if (!P5.0 || !P5.4)
    {
    if ((delay_count > 5) && (!P5.0 || !P5.4))
      {
      input_key = Read_Key_Input();
      Menu(input_key);
      Motor_Dir = input_point.motor_dir;
      setpoint = motor_rpm;
      delay_count = 0;
      }
    else
      {
      delay_count++;
      }
    }
  // Adaptation of the RPM value for the Display
  // Transform the RPM valueinto number A.B.C. for the Display
  dummy_value = actual_value;
  i = 0;
  s[3] = s[2] = s[1] = s[0] = 0;
  do
    {
    s[i++] = dummy_value %10;
    }
  while ((dummy_value /= 10) > 0);
  if (loop_Menu == 5)
    {
    uc_num_A = s[3];
    uc_num_B = s[2];
    uc_num_C = s[1];
    loop_Menu = 0;
    }
  else
    {
    loop_Menu++;
    }
  LED_OUT();
  if (break_active.on_off)
    {
    BREAK = 1;
    }
  else
    {
    BREAK = 0;
    if (stop_start_sequence)
```

```
      {
      Start_Sequence();
      }
    }
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  LED_OUT()
** VERSION      = V0.1
** DATE          = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**            NEC Electronics (Europe) GmbH
**            Technical Product Support
**            Semiconductors and Displays Business Unit
**            D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Display control (3 x 7Segment Display)
** ============================================================================================= */
void LED_OUT(void)
  {
  void Loop_Pause(unsigned int);
  void LED_Fkt(unsigned char, unsigned char, unsigned char);
  // Make choice between Menu and Menu function */
  if (menu_flag)
    {
     switch (menu_choice) // switch(menu_choice)
      {
      case DIR_LEFT:
        // Display : d.-L */
        LED_Fkt(12,10,21); //0xDE, 0x40, 0x38
        break;
                    case DIR_RIGHT:
                    // Display : d.-r */
                    LED_Fkt(12,10,13); //0xDE, 0x40, 0x50
                    break;
                              case UP_RPM:
                              // Display: r.-u */
                              LED_Fkt(14,10,15);   //0xD0, 0x40, 0x1C
                              break;
                                        case DOWN_RPM:
                                        // Display : r.-d */
                                        LED_Fkt(14,10,11); //0xD0, 0x40, 0x5E
                                        break;
                                                  case STOP:
                                                  // Display : Stp. */
                                                  LED_Fkt(22,17,23); // 0x6D, 0x78, 0xF3
                                                  break;
                                                            case ACT_RPM:
                                                            // Display : Acr. */
                                                            LED_Fkt(26,25,13); //0x50, 0x1C, 0xD4
                                                            break; } }
  else
    {
    if (Menu_fkt)
      {
       switch (menu_choice) // switch(Menu_fkt)
        {
        case UP_RPM
          LED_Fkt(led_input.uc_a, led_input.uc_b, led_input.uc_c);
          break;
                    case DOWN_RPM:
                              LED_Fkt(led_input.uc_a, led_input.uc_b, led_input.uc_c);
                              break;
                              case STOP:
                                        LED_Fkt(0x0A, break_active.on_off_old, break_active.on_off);
                                        break;
                                        case ACT_RPM:
```

```
                                LED_Fkt(uc_num_A, uc_num_B, uc_num_C);
                                break;
}}}}
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  LED_Fkt() and Loop_Pause()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**           NEC Electronics (Europe) GmbH
**           Technical Product Support
**           Semiconductors and Displays Business Unit
**           D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Display control (3 x 7Segment Display)
** ============================================================================================= */

void LED_Fkt(unsigned char a, unsigned char b, unsigned char c)
  {
  DIG_2 = 0;
  DIG_3 = 0;
  LED_PORT = LED_CODE[a];
  DIG_1 = 1;

  Loop_Pause(500);

  DIG_1 = 0;
  DIG_3 = 0;
  LED_PORT = LED_CODE[b];
  DIG_2 = 1;

  Loop_Pause(400);

  DIG_1 = 0;
  DIG_2 = 0;
  LED_PORT = LED_CODE[c];
  DIG_3 = 1;

  Loop_Pause(300);
  }

void Loop_Pause(unsigned int loop)
  {
  for (loop_pause=0;loop_pause<loop;loop_pause++)
    {
    }
  }
```

```
/*============================================================================================
** PROJECT      = Motor control
** MODULE       =  Read_Key_Input()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** ============================================================================================
** Description: Detect the key input
** ============================================================================================ */

unsigned char Read_Key_Input(void)
  {
  // MENU:
  // Menu Choice:
  //            1.Turn Direction
  //            2.RPM
  //            3.STOP

  #define NO_KEY    0xFF
  unsigned char key_code;

  /* Intern Prototyp function */
  void delay(void);

  // Read Key 2 times with delay between */
  KEY_PORT = P5;

  KEY_PORT &= 0x11;

  delay();

  key_code = P5;

  key_code &= 0x11;

  // Check if key input ok and return key code value */
  if ((key_code == KEY_PORT) && (key_code != NO_KEY))
    {
    // Return Key Code */
    if (key_code == 0x01)
      {
      key_down_pushed = 1;
      }
    else
      {
      key_up_pushed = 1;
      }
    return key_code;
    }
  else
    {
     key_up_pushed   = 0;
    key_down_pushed = 0;
    return NO_KEY;
    }
  }
/*   delay */
```

```
void delay(void)
  {
  unsigned int loop_delay;
  for (loop_delay = 0; loop_delay < 1000; loop_delay++)
    {
    }  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       =  Menu()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**            NEC Electronics (Europe) GmbH
**            Technical Product Support
**            Semiconductors and Displays Business Unit
**            D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Detect and control the menu points
** ============================================================================================= */

void Menu(unsigned char input_key )
  {
  void rpm_input(void);
  void turn_dir(void);
  void break_input(void);
  if (input_key == 0x10 && Menu_flag)
    {
    // Main Menu Choice
    if (menu_choice <= 0xC0)
      {
      menu_choice = 0xFE;
      }
    else
      {
      menu_choice = menu_choice << 1;
      }
    }
  else
    {
    /* BLOCK MENU_CHOISE, GET THE INPUT & SELECT FUNCTION */
    switch (menu_choice)
      {
      case UP_RPM:
        //Drehzahl vergroessern
        input_point.rpm_motor_up = 1;
        input_point.rpm_motor_down = 0;
        rpm_input();
        break;
                    case DOWN_RPM:
                    input_point.rpm_motor_down = 1;
                    input_point.rpm_motor_up = 0;
                    rpm_input();
                    break;
                            case DIR_LEFT:
                            input_point.motor_dir_left = 1;
                            input_point.motor_dir_right = 0;
                            turn_direction();
                            break;
                                    case DIR_RIGHT:
                                    input_point.motor_dir_right = 1;
                                    input_point.motor_dir_left = 0;
                                    drehrichtung();
                                    break;
                                            case STOP:
                                            input_point.break = 1;
                                            break_input();
                                            break;
                                                    case ACT_RPM:
```

```
                                        input_point.act_rpm = 1;
                                        break;
    }
}
```

```
  // DOWN Key pushed: -> Go to Menu function and block the main Menu choice
  if (input_key == 0x01)
    {
    Menu_flag = 0;
    Menu_fkt = 1;
    }
  else
    {
    //  FREE MENU_CHOISE */
    if (input_key == 0x00)
      {
      Menu_flag = 1;
      Menu_fkt = 0;
     }
  }
}
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       = rpm_input()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**            NEC Electronics (Europe) GmbH
**            Technical Product Support
**            Semiconductors and Displays Business Unit
**            D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: calculate input up or down rpm of the motor and transform it for the display
** ============================================================================================= */

void rpm_input(void)
  {
  // setpoint:
  // Zahl = a.b.c
  // Aktuell: nur 100'er schritte
  // UP_DOWN_RPM
  if ((input_key == 0x10) && Menu_fkt)
    {
    if (input_point.rpm_motor_up)
      {
      // UP_RPM
      if (motor_rpm < 4500)
        {
        motor_rpm +=100;
        if ((led_input.uc_b < 9)&&(led_input.uc_a <=4))
          {
          led_input.uc_b++;
          }
        else
          {
          led_input.uc_b = 0;
          if (motor_rpm < 1000)
            {
            led_input.uc_c = 0;
            led_input.uc_a = 0;
            }
          else
            {
            if (led_input.uc_a <4)
              {
              led_input.uc_a++;
              }
            else
              {
              led_input.uc_a = 4;
              if (led_input.uc_b > 5)
                {
                led_input.uc_b = 5;
                }
              }
            }
          }
        }
      else
        {
        motor_rpm = 4500; //max rpm x 10
        led_input.uc_a = 4;
        led_input.uc_b = 5;
```

```
    led_input.uc_c = 0;
    }
}
```

```
    else
      {
      // DOWN_RPM */
      if (input_point.rpm_motor_down  && !input_point.break)
        {
        }
      if ((motor_rpm > 0) && (motor_rpm <= 4500))
        {
        motor_rpm -=100;
        if (led_input.uc_b > 0)
          {
          led_input.uc_b--;
          }
        else
          {
          led_input.uc_b = 9;
          if (led_input.uc_a > 0)
            {
            led_input.uc_a--;
            led_input.uc_c = 0;
            }
          else
            {
            led_input.uc_a = 0;
            led_input.uc_c = 0;
            }
          }
        }
      else
        {
        motor_rpm = 0; //min rpm x 10
        led_input.uc_a = 0;
        led_input.uc_b = 0;
        led_input.uc_c = 0;
        }
      }
    }
}
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       = turn_direction()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: detect the turn direction of the motor
** ============================================================================================= */
void turn_direction(void)
  {
  if ((input_key == 0x10) && Menu_fkt)
    {
    if (input_point.motor_dir_right == 1)
      {
      input_point.motor_dir = 1;
      }
    else
      {
      input_point.motor_dir = 0;
      }
    }}
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       = break_input()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: detect break input
** ============================================================================================= */

void break_input(void)
  {

  if (input_point.break &&(input_key == 0x10) && Menu_fkt)
    {
    break_active.on_off ^= input_point.break;
    input_point.break = 0;
    input_point.rpm_motor_up = 0;
    input_point.rpm_motor_down = 0;
    }
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       = norm_value()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: norm the value into the dregulator range
** ============================================================================================= */

unsigned int norm_value( unsigned int cnt )
  {
  // norm the input values
  unsigned long lbuf;
  unsigned int  ibuf;
  if (cnt > actual_count_min)
    {
    lbuf = norm_constant;
    lbuf = lbuf / (unsigned long)cnt;
    ibuf = (unsigned int)lbuf;
    ibuf++;
    ibuf = ibuf >> 1; // div by two
    if (ibuf > actual_value_ovr)
      {
      ibuf = actual_value_ovr;
      }
    return ibuf;
    }
  else
    {
    // if cnt below min -> max. value
    return actual_value_ovr;
    }
  }
```

```
/*=============================================================================================
** PROJECT      = Motor control
** MODULE       = transform_regulated_quantity()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =============================================================================================
** Description: Transform the calculated quantity into duty cycle
** ============================================================================================= */

unsigned int transfrom_regulated_quantity( unsigned int y_in, unsigned int period )
  {
  /*
        Convert Y (0...4095) into PWM set value.
        Limit of PWM set value = 0...period [0...512]
  */

  unsigned long lbuf;
  if (period)
    {
    lbuf = (unsigned long)(period << 1); // use (period * 2) to increase precision
    lbuf = lbuf * (unsigned long)y_in;
    lbuf = lbuf / (unsigned long)4096;
    lbuf++;
    lbuf = lbuf >> 1;                   // devide by two (precision)
    return (unsigned int)lbuf;
    }
  else
    {
    // result allways 0
    return 0;
    }
  }
```

```
/*=========================================================================================
** PROJECT      = Motor control
** MODULE       = main()
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** =======================================================================================
** Description: Main function of the system
** ======================================================================================= */
void main(void)
  {
  _DI();// Disable all interrupts

  actual_value   = 0;
  value_old_one    = 0x0;
  value_new_one    = 0xFFFF;
  signalwidth_one = 0xFFFF;
  break_active.on_off = 1;
  BREAK = 1;

  /* uPD init */
  hdwinit();

  /* Perihperals init*/

  //16-bit Timer TM00 (Speed measurement)
  TM00_INIT();

  //10-bit Inverter Timer TM7 (Six PWM signal generation)
  Inverter_INIT();

  //Real-Time output port (Signal modulation )
  Real_Time_INIT();

  //Watch-Dog Timer (Interval timer)
  WDTM_INIT();

  // Read the actual position of the motor
  Motor_Pos    = P0 & 0x0E;

  _EI();//Enable all interrupts

  RUN_WDTM = 1;  // Start counting with the WDTM as Intervall Timer
  DTIME = 0x14;  // delay time setting = 2,5Ýs
  TMC7.7 = 1;    // start inverter timer

  while (1)
    {
    if (IRQ_WDTM)
      {
      IRQ_WDTM = 0;
      Intervall_Timer();
      }
    if (START_REGULATOR & !BREAK)
      {
      stop_start_sequenc = OFF;
```

```
      // Transform timer count in actual_value used in the Regulator
      actual_value = norm_value( actual_count );
      //Calculate regulating quantity
      Y = Regelung();
      // Transform regulated quantity into Duty Cycle
      DUTY_CYCLE = transfrom_regulated_quantity( Y, PWM_CYCLE );
      }
    else
      {
      if (BREAK)
        {
        // STOP the SYSTEM RUNNING
        START_REGULATOR  = OFF;
        signalwidth_one        = OFF;
        actual_value           = OFF;
        DUTY_CYCLE             = OFF;
        Y                      = OFF;
        RTPM01                 = 0x3F;
        stop_start_sequence = ON;
        }
      }
    }
}
```

```
/*============================================================================================
** PROJECT      = Motor control
** MODULE       =
** VERSION      = V0.1
** DATE         = 20.06.2002
** LAST CHANGE  =
**
** Author:Jusuf Suad
**          NEC Electronics (Europe) GmbH
**          Technical Product Support
**          Semiconductors and Displays Business Unit
**          D-40472 Düsseldorf, Germany
** ============================================================================================
** Description: Variables used in the system
** ============================================================================================ */

#define OFF 0;
#define ON 1;

/* LED ANZEIGE: Display */
// SEGMENTE
#define LED_PORT P4

// COMMON
#define DIG_1  P2.0
#define DIG_2  P5.6
#define DIG_3  P5.7

/* Menu Points */
#define DIR_LEFT  0xF8
#define DIR_RIGHT 0xF0
#define UP_RPM    0xFE
#define DOWN_RPM  0xFC
#define STOP      0xE0
#define ACT_RPM   0xC0

/* mode register definitions */

/* interrupt register definitons */

/* constant definitions */

//                    0   1   2   3   4   5   6   7   8   9
unsigned int LED_CODE[27] ={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,
//                    -    d    d.   r    r.   u    u.   t    t.
                      0x40, 0x5E, 0xDE, 0x50, 0xD0, 0x1C, 0x9C, 0x78, 0xF8,
//                    n    n.   L    S    P.   E    C    A
                      0x54, 0xD4, 0x38, 0x6D, 0xF3, 0x79, 0x39, 0x77 };

/* Regulator constants: */

#define norm_constant       5237500l     // double of constant to norm the actual_count to actual_value
#define actual_count_min        524      // minimum valid count value
#define actual_value_ovr       5000      // limit if count exceeds measurment rang - to do
#define actual_value_max       4400
#define actual_value_min       0
#define setpoint_max           4400
#define setpoint_min           0

/*   Variables definition */
unsigned int count_irq = 0; //Event counter to recognize key
```

```
unsigned char delay_count = 0; //Event counter to recognize key

unsigned int count_sample_value = 0; //counter for the PI sample time

unsigned int sample_counter_value = 0; //var. for the PI sample time definition

unsigned char stop_start_sequenc = 0; //Flag for the start sequenc

unsigned char uc_num_A; // Display number
unsigned char uc_num_B;
unsigned char uc_num_C;

unsigned char s[4]; //Display nummber array
unsigned char i =0; //loop var for the s[] array

unsigned int dummy_value; //to get the actuall speed
                          //and transform for the display

unsigned char loop_Menu;

int ovf_one; //Overflow var. to count the ovf. from TM00
unsigned int signalwidth_one;//Actual count of the TM00
unsigned int wert_old_one = 0;//Count value (n-1) from TM00
unsigned int wert_new_one = 0;//Count value (n) from TM00
/* motor position control variables */
unsigned char Motor_Pos;

/* regulation variables */
unsigned int actual_count = 0xFFFF;    // actual timer count value for speed (16 Bit = [0 ... 65535])
int actual_value = 0;
long Integrator = 0;
int setpoint = 0;
int Y = 0;   // actual output value (12 bit [0...4095])
int Kp = 15; // Kp = kp * 256
int Ki = 50; // Ki = ki * 1024

/* LED variables */
static unsigned char key_down_pushed ;
static unsigned char key_up_pushed ;
static unsigned char KEY_PORT;

unsigned char loop_main = 1;
unsigned char uc_count = 0;
unsigned char uc_dir = 0;
unsigned char uc_rpm_count = 0;
unsigned char uc_rpm = 0;
unsigned char uc_stop = 0;
unsigned char menu_choice = 0xFF;
unsigned char menu_fkt = 0;
unsigned char menu_flag = 1;
unsigned char input_key;

unsigned int motor_rpm = 0;
unsigned int loop_pause;

/* PWM & DUTY CYCLE definition */
unsigned int PWM_CYCLE  = 125; //max 512;
unsigned int DUTY_CYCLE = 0;//20;
unsigned int DUTY_CYCLE_OLD;

//   RTP bit pattern tables
```

```
unsigned char RTP_SWITCH_RIGHT[15] = {0x00, 0x00, 0x06, 0x00, 0x18, 0x00, 0x0B, 0x00, 0x21, 0x00, 0x32, 0x00, 0x2C,
0x00, 0x07};

unsigned char RTP_SWITCH_LEFT[15]  = { 0x00, 0x00, 0x2C, 0x00, 0x32, 0x00, 0x21, 0x00, 0x0B, 0x00, 0x18, 0x00, 0x06,
0x00, 0x07};

/* bit variables */
bit Motor_Dir;    // Turn Direction of the Motor

bit START_REGLER; // Start flag for the regulato
bit BREAK;        // Break flag to stop the system running

/* TM7 bit definition */
bit TM7_RTP = DCCTL1.7;
bit TM7_PWMH = DCCTL1.6;
bit TM7_PWML = DCCTL1.5;
bit bINV = DCCTL1.4;
bit bALV = TMM7.3;

/* WDTM bit definition */
bit RUN_WDTM = WDTM.7; //Start counting with the watchdog timer
bit IRQ_WDTM = IF0L.0; //WDTM IRQ FLAG

/* TM00 IRQ-bit definitions */
bit INTTM000 = MK0H.2;
bit INTTM010 = MK0H.3;

bit TMIF000 = IF0H.2;
bit TMIF010 = IF0H.3;

bit IRQ_TM00 = IF0H.2; //Measure

/* TM01 IRQ-bit definitions */

bit INTTM001 = MK0H.4;
bit INTTM011 = MK0H.5;

bit TMIF001 = IF0H.4;
bit TMIF011 = IF0H.5;

bit IRQ_OVF  = IF0H.3; //OVF = compare TM010

// Menu
struct Menu
  {
  unsigned char dir_left;
  unsigned char dir_right;
  unsigned char up_rpm;
  unsigned char down_rpm;
  unsigned char stop;
  unsigned char rpm_act;
  } Menu_point;

// input

struct input
  {
  unsigned char motor_dir_right;
  unsigned char motor_dir_left;
  unsigned char motor_dir;
```

```
  unsigned char rpm_motor_up;
  unsigned char rpm_motor_down;
  unsigned char break;
  unsigned char run_in;
  unsigned char act_rpm;
  } input_point;

struct value
  {
  unsigned char uc_a;
  unsigned char uc_b;
  unsigned char uc_c;
  unsigned int led_value;
  } led_input;

struct break
  {
  unsigned char on_off;
  unsigned char on_off_old;
  } break_active;

struct input_return
  {
  unsigned char ready;
  unsigned char ready_old;
  } return_active;
```

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-6250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	☐	☐	☐	☐
Technical Accuracy	☐	☐	☐	☐
Organization	☐	☐	☐	☐

[MEMO]