



Automated Car By LineTracer MBed System

INDEX

1. Introduction.....	O-1
2. Goal Of Learning.....	2-3
4. Creating Program	4-6
5. BIOS & CLOCK	7-26
6. Software Writing...	27-30
7. LED & Switch Controll	31-41
8. AD Conversion.....	42-81
9. Motor Control	82-139
10. Fuzzy or PID	140-164
11. UART	165-188
12. Mbed System.....	168-END

Therefore, this kind of Robot should sense the line with its Infrared Ray (IR) sensors that installed under the robot. After that, the data is transmitted to the processor by specific transition buses. Hence, the processor is going to decide the proper commands and then it sends them to the driver and thus the path will be followed by the line follower robot. Micon Racer is a Automated robot designed and tested in order to attend at Tabrize line follower robots competition. But it encounter with some technical and mechanical problems. In this Paper, we have illustrated the process of design, implementation and testing MiconRacer, a small Robot designed for the line follower robots competition. The technical and mechanical issues and problems also have investigated.

This kind of robot can be used for military purposes, delivery services, transportation systems, blind assistive applications. Moreover, there are many annual line follower robots competitions organized by universities or industries around the world. They usually ask robotic teams for building a small robot with specific dimensions and weight according to the competition rules.

I. INTRODUCTION

Generally, the line follower robot is one of the selfoperating mobile machines that follows a line drawn on the floor. The path can be a visible black line on a white surface (reverse). The basic operations of the line follower are as follows:

- Capturing the line position with optical sensors mounted at the front end of the robot. Most are using several numbers of photo-reflectors. Therefore, the line sensing process requires high resolution and high robustness.
- Steering the robot to track the line with any steering mechanism. This is just a servo operation; actually, any phase compensation will be required to stabilize tracking motion by applying digital PID filter or any other servo algorithm.
- Controlling the speed according to the lane condition. The speed is limited during passing a curve due to the friction of the tire and the floor.

This robot can be divided into several parts:

- Sensors
- ADC (Analog to Digital Converter) and sensor circuit
- Processor
- Motor Driver
- Fuzzy/PID
- Time Read
- UART Method

ABOUT Renesas

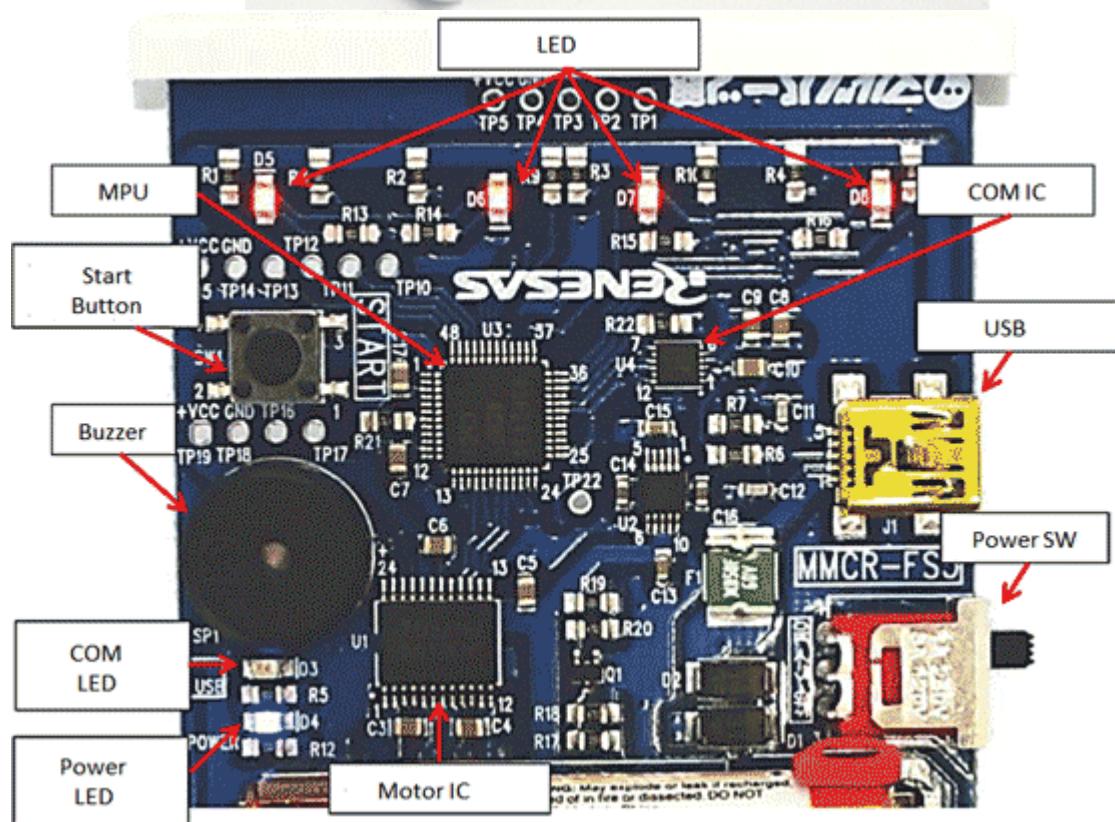
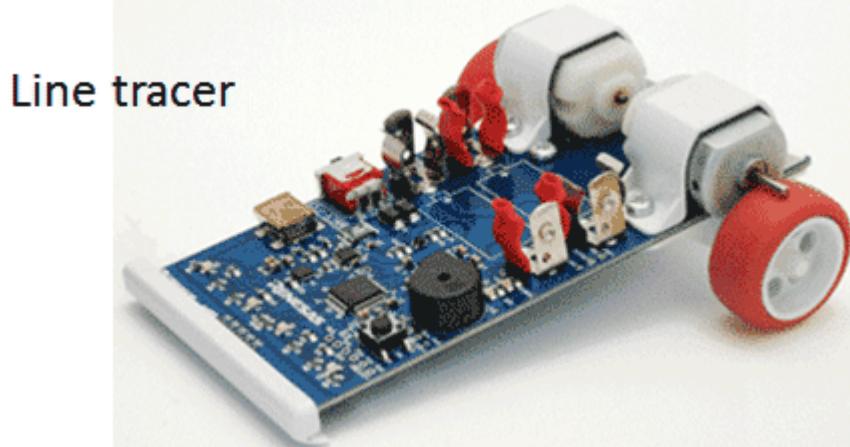
in response to user requirements that are rapidly expanding in scope, Renesas Electronics offers microcontroller (MCU) and microprocessor (MPU) products that provide excellent expandability while allowing customers to make full use of existing resources. Available in a wide array of memory and package options, Renesas microcontrollers, and microprocessors are fast, highly reliable, low in cost, and deliver eco-friendly performance. Incorporating the latest process technology, which enables integration of large-capacity flash memory, they are used in a wide array of applications, including demanding fields that require high quality and high reliability, such as the automotive industry. Additionally, there is a robust support system in place to help reduce development costs and reduce the time required for development. It consists of a variety of development tools, including products from other companies, backed by extensive technical documentation, software libraries, and active user communities. As the world's number one MCU/MPU vendor, Renesas Electronics provides the best and most powerful solutions based on a wide selection of microcontrollers and microprocessors.

[1] The goal of learning

“By making the control software of the line tracer, to learn the overall technology of the microcomputer system.”

[1.Features]

- (1)Use the C ++ language.
- (2)Control by Fuzzy logic.
- (3)Hardware to be used,



MCU=RENESAS R8C/34C (R5F21344CNFP)
ROM 16KB, RAM 1.5KB, 16bit CPU,
Max Frequency 20MHz

(4) The manufacturer's website:

<http://www.marutsu.co.jp/pc/i/237124/>

(5) Hardware Manual

Compiler Package for M16C Series and R8C Family

<http://www.renesas.eu/products/mpumcu/r8c/r8c3x/r8c34c/Documentation.jsp?typeVXNlcidzIE1hbnVhbDogSGFyZHdhcmU=>



(6) Software Manual

http://documentation.renesas.com/doc/products/mpumcu/rej09boo01_r8csm.pdf

(7) Coding Tool

Evaluation Software] C/C++ Compiler Package for M16C Series
and R8C Family M3T-NC30WA V.6.00 Release 00

http://am.renesas.com/support/downloads/download_results/C100000-C999999/tools/evaluation_m3t_nc30wa_v6.jsp

(8) Writing software to Flash memory

http://www.marutsu.co.jp/contents/shop/marutsu/include/large_order/img/MR2_img/dl/micon_racer2_dl.zip

(9) Other information

Online training for Renesas products

Renesas Interactive

<http://www.renesasinteractive.com/>

[6.Creating a Program]

Here we will create three new files.

(1) Create “LineTracer.h” file.

And add the following description.

```
#include "sfr_r834c.h" //Definition of the R8C/34C SFR
#include "Prototype.h"
```

(2) Create “Prototype.h” file.

Write the following description.

```
void Hwsetup (void);
```

(3) Create “Hwsetup.cpp” file.

And add the following description.

```
#include "sfr_r834c.h" //Definition of the R8C/34C SFR

void Hwsetup()
{
}
```

(4) Describe “LineTracer.cpp” file asfollows.

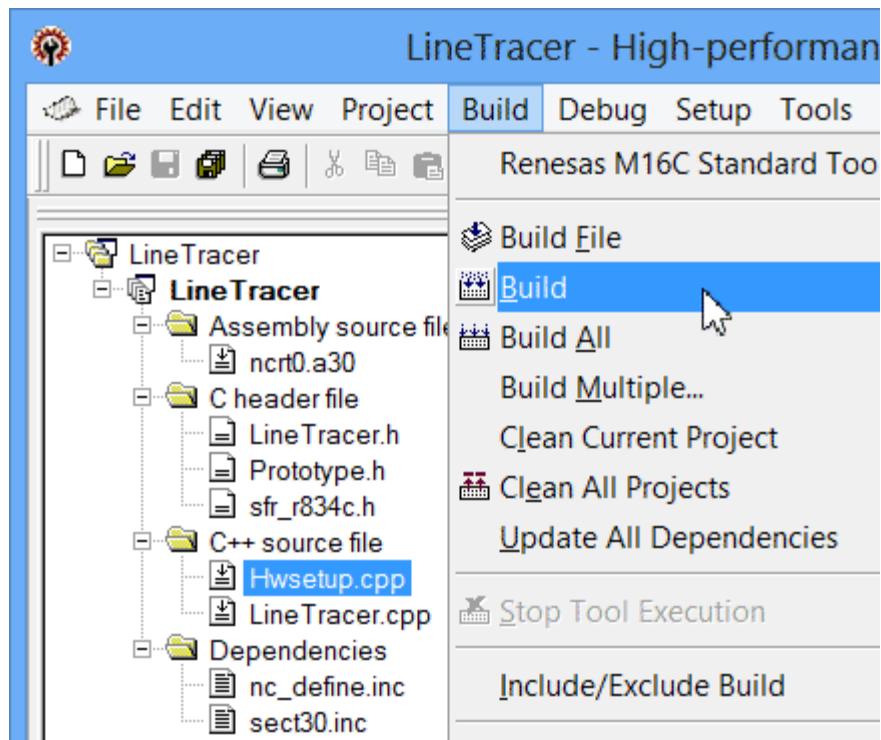
```
#include "LineTracer.h"

void main(void)
{
    Hwsetup();

    //Loop of main
    while (1)      /* Main processing */
    {
        }

}
```

(5) Build your project.



If there is no error, the results are displayed as shown below

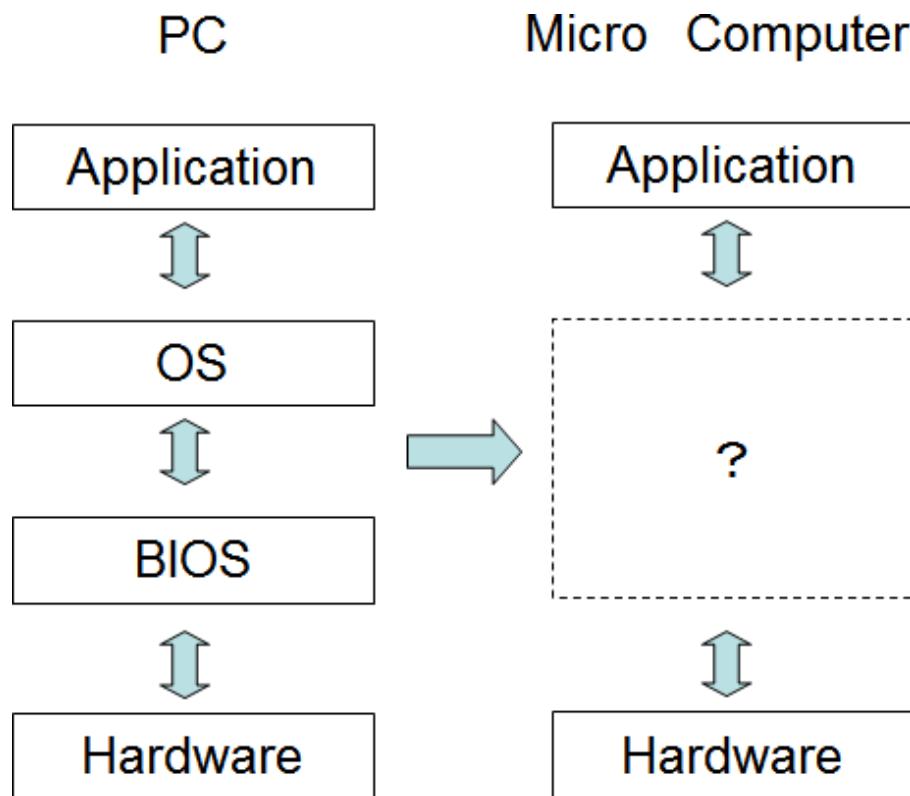
```
ROMDATA SECTION: 00000028 Byte(s)
PROGRAM SECTION: 00000095 Byte(s)
Optimizing Linkage Editor Completed
Phase OptLinker finished

Build Finished
0 Errors, 0 Warnings
```

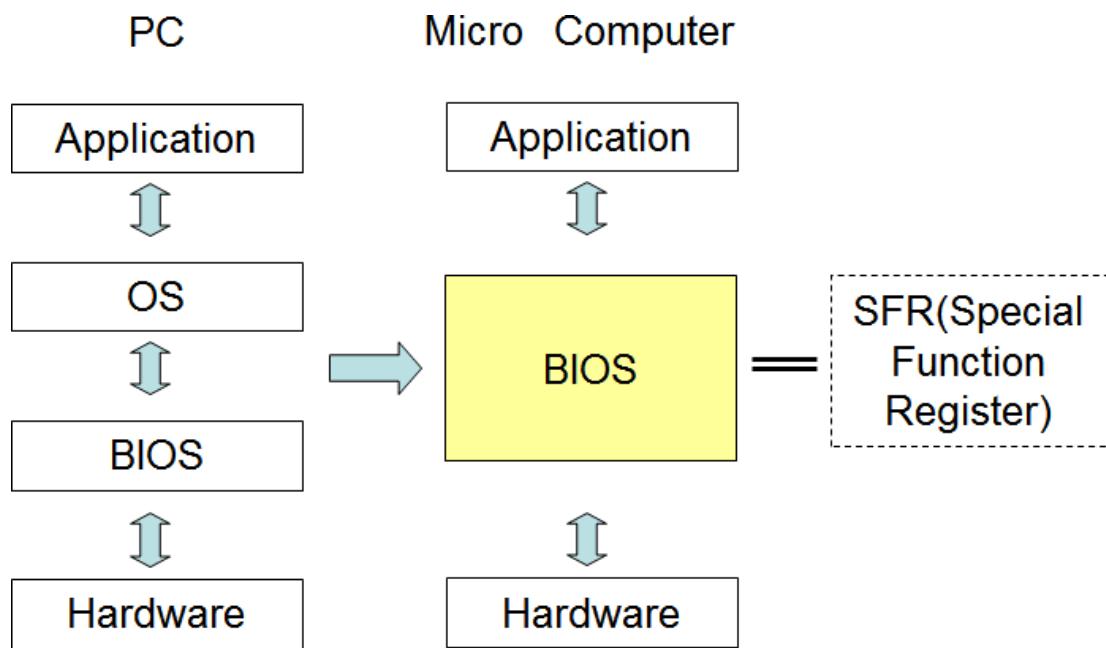
[7.BIOS]

With the PC, we operate hardware through the OS and BIOS from an application.

- (1) In the case of a microcomputer, what do you operate hardware through from your application?

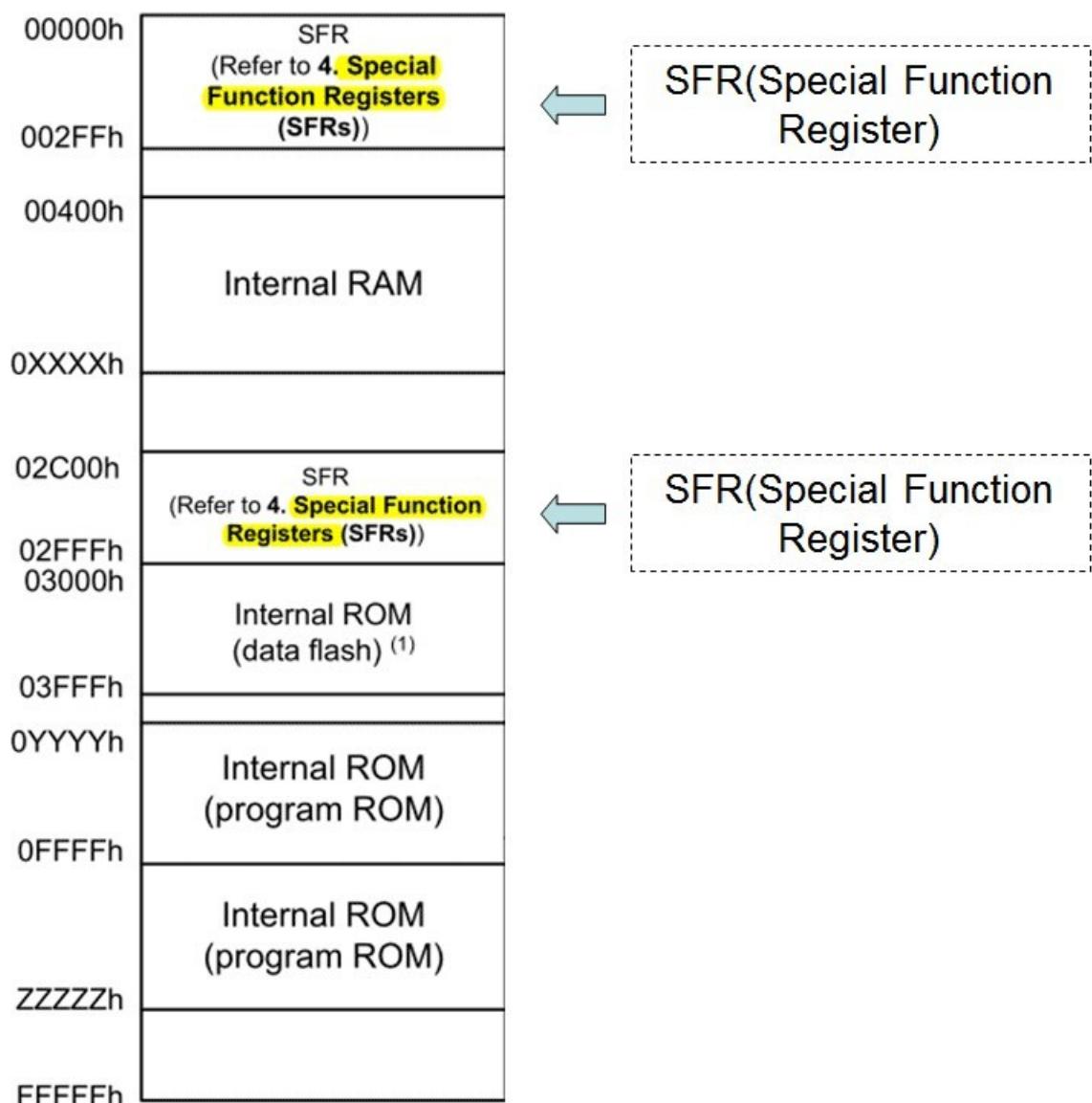


(2) For that, this CPU has special function registers to operate hardware.



(3) The software that manipulates SFR becomes BIOS.

(4) Look at the address map of this CPU.



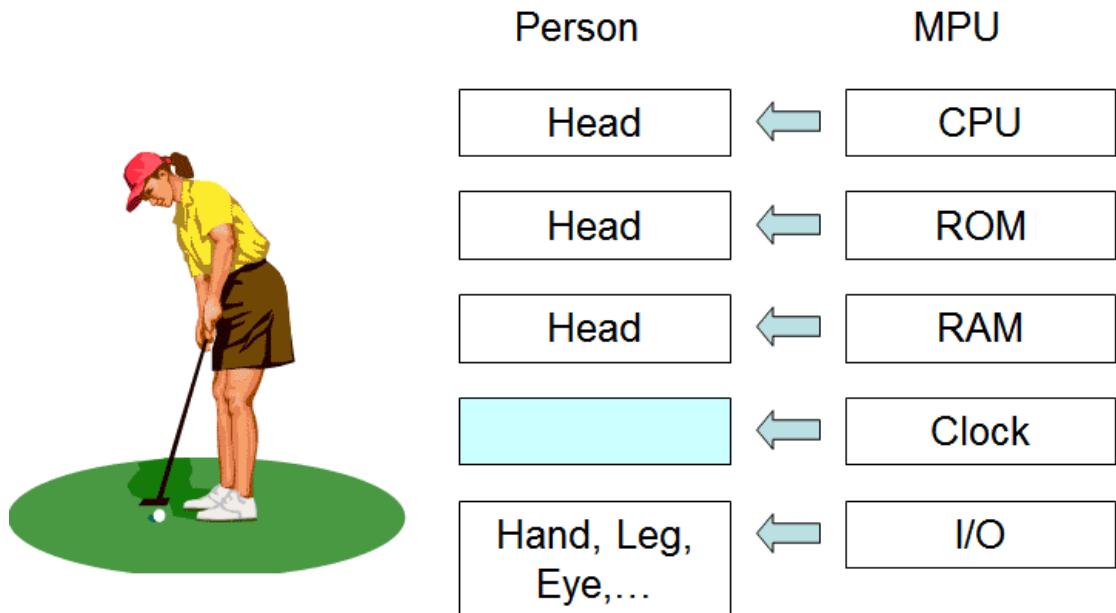
(5) The functions such as Digital I/O, AD/DA, I₂C, UART, LIN etc. are gathered in this SFR.

[8.Clock]

When we compare a human body with a computer, we can contrast like the chart below.

(1) What is equivalent in the blank square?

What is an important part of a body equivalent to a clock?

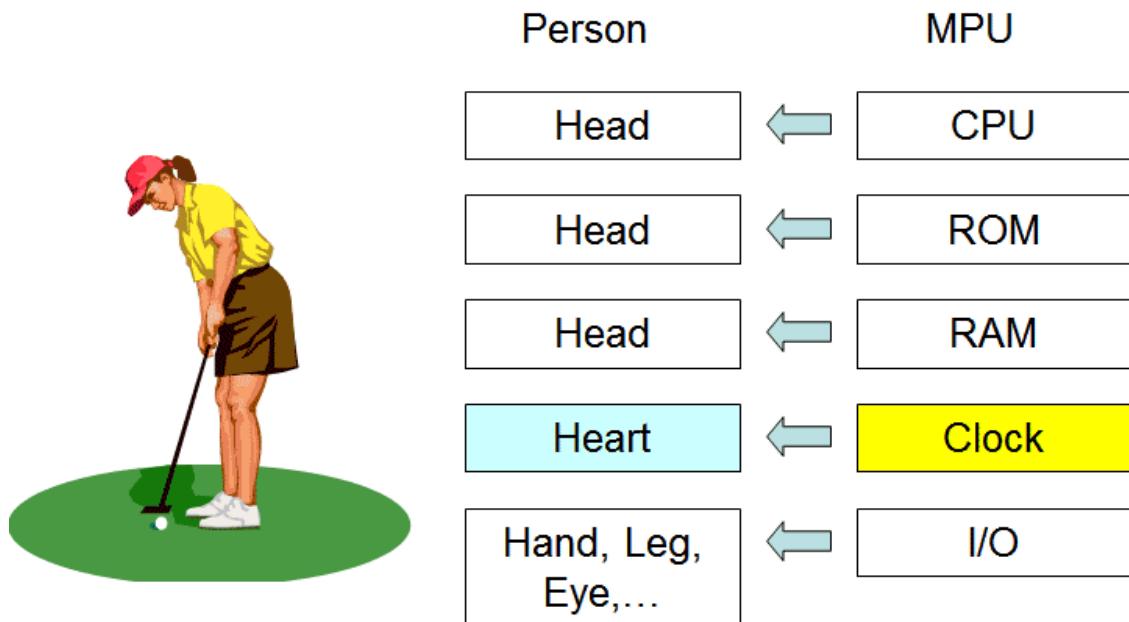


MPU=Micro Processing Unit

MCU=Micro Control Unit same as MPU

CPU=Central Processing Unit

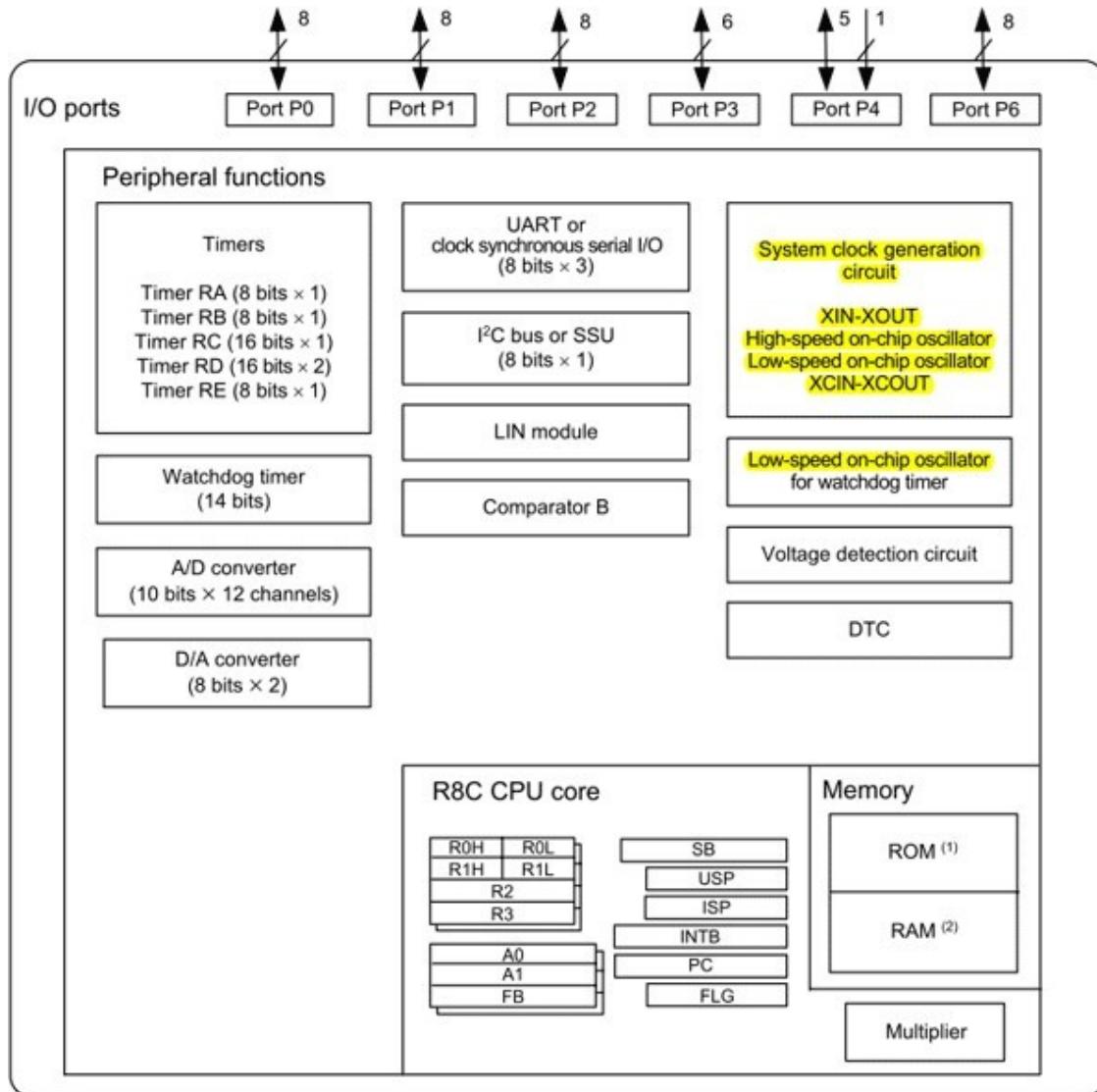
It is heart. Heart sends blood to the whole body.



(2) Let's start from an important clock equivalent to human heart.

[9.Oscillation]

(1) Let's look at the block diagram of the MPU.



Clock generator has been drawn in the upper right corner.

(2) Let's look at a little more detail.

Table 9.1 Specification Overview of Clock Generation Circuit

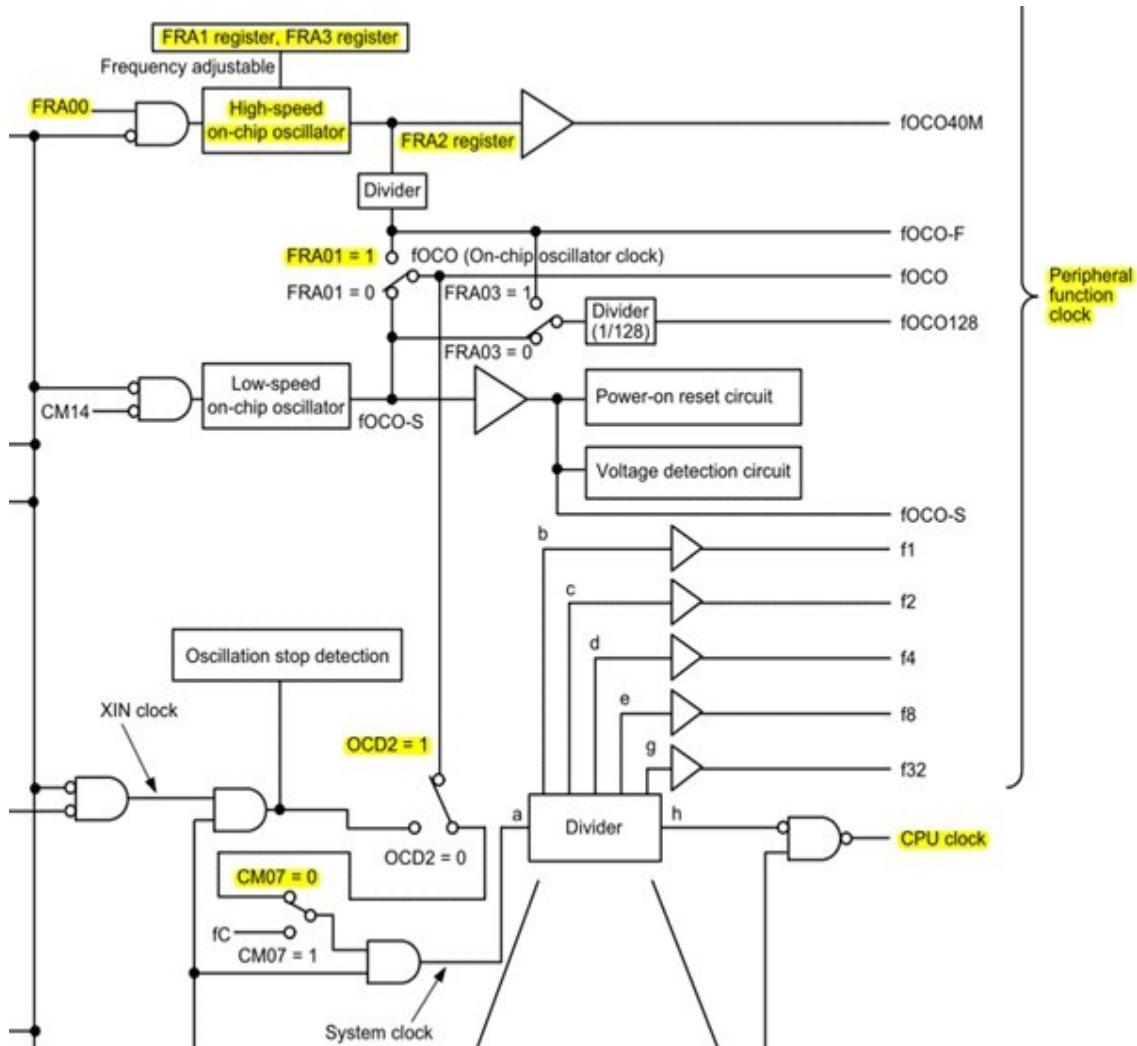
Item	XIN Clock Oscillation Circuit	XCIN Clock Oscillation Circuit	On-Chip Oscillator		Low-Speed On-Chip Oscillator for Watchdog Timer
			High-Speed On-Chip Oscillator	Low-Speed On-Chip Oscillator	
Applications	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source 	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source 	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source • CPU and peripheral function clock source when XIN clock stops oscillating 	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source • CPU and peripheral function clock source when XIN clock stops oscillating 	<ul style="list-style-type: none"> • Watchdog timer clock source
Clock frequency	0 to 20 MHz	32.768 kHz	Approx. 40 MHz ⁽⁴⁾	Approx. 125 kHz	Approx. 125 kHz

Review of Clock Generation Circuit

XCIN Clock Oscillation Circuit	On-Chip Oscillator		– (1)	–
	High-Speed On-Chip Oscillator	Low-Speed On-Chip Oscillator		
<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source 	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source • CPU and peripheral function clock source when XIN clock stops oscillating 	<ul style="list-style-type: none"> • CPU clock source • Peripheral function clock source • CPU and peripheral function clock source when XIN clock stops oscillating 	Usable	Usable
		Oscillate	Stop ⁽⁵⁾ Oscillate ⁽⁶⁾	

Since the line tracer required high-speed operation, we will use the High Speed On-Chip Oscillator.

(3) The figure below is the clock generator.



We will set a value in this SFR (Special Function Register).

(4) We add in “Hwsetup.cpp” file as described below.

```
#include "LineTracer.h"
static void mcu_init(void);

void Hwsetup(void)
{
    asm("FCLR I");      /* Interrupt disabled */
    mcu_init();
    asm("FSET I");      /* Interrupt enable */
}

static void mcu_init()
```

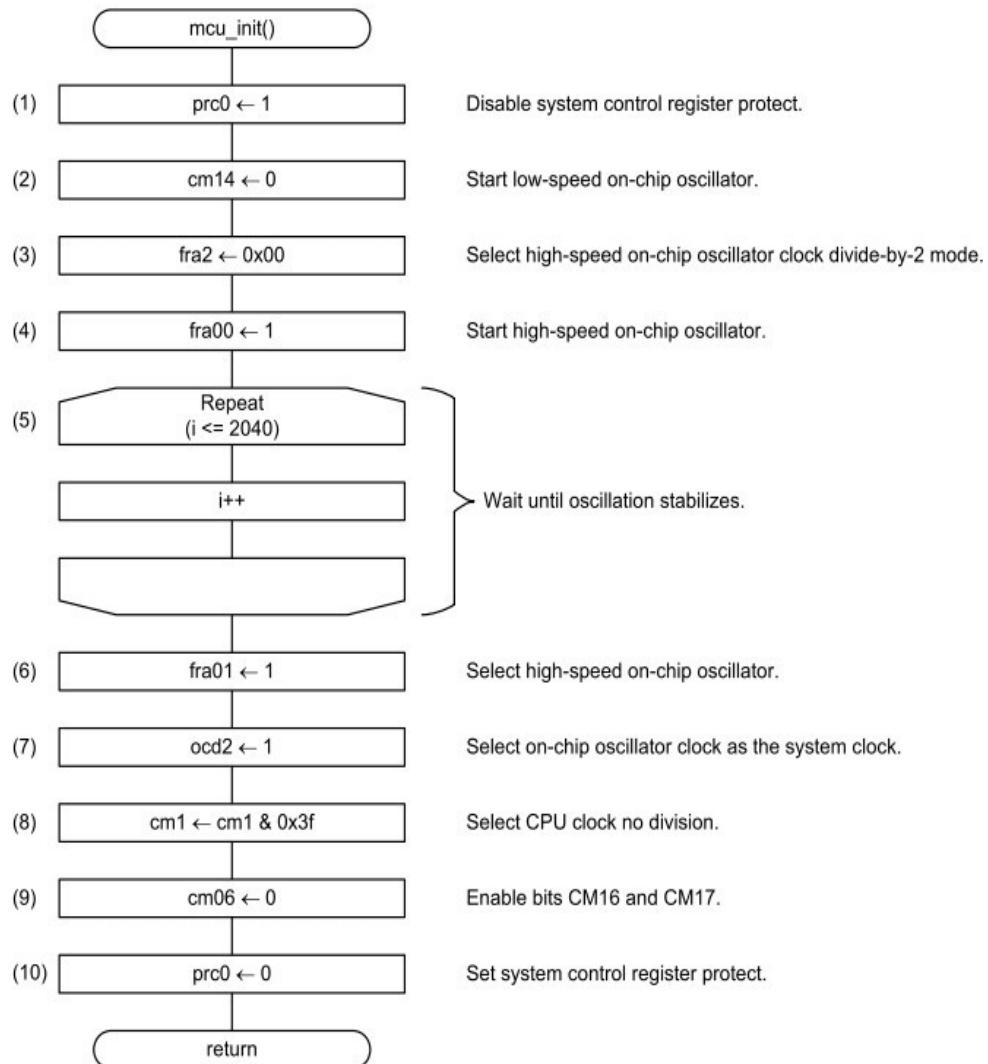
Now, we will write in this “mcu_init()” function.

[10.MCU_init]

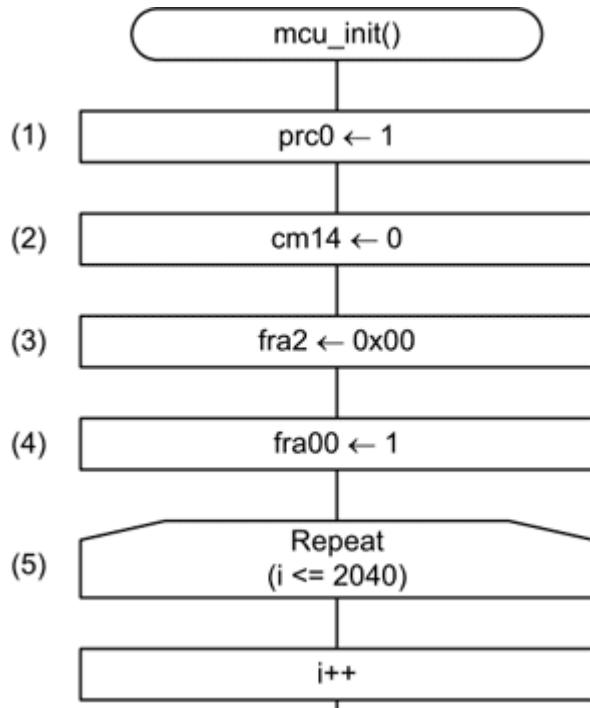
In Renesas manual, the set order of the clock generator is described as follows.

4.3 System Clock Setting

- Flowchart



Let's take a closer look.



- (1) First, since we operate the oscillator related registers, we have to remove the protection of “Protect Register” (PRCo).

```
prco = 1; /* Protect off */
```

10.1 Register

10.1.1 Protect Register (PRCR)

Address 000Ah								
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	PRC3	PRC2	PRC1	PRC0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PRC0	Protect bit 0	Enables writing to registers CM0, CM1, CM3, OCD, FRA0, FRA1, FRA2, and FRA3. 0: Write disabled 1: Write enabled (2)

When a system causes the runaway, we have to minimize damage, in particular in the industrial products. Therefore this protection was prepared for.

(2) Until high-speed oscillator is to stabilize oscillation, we use the low-speed on chip oscillator.

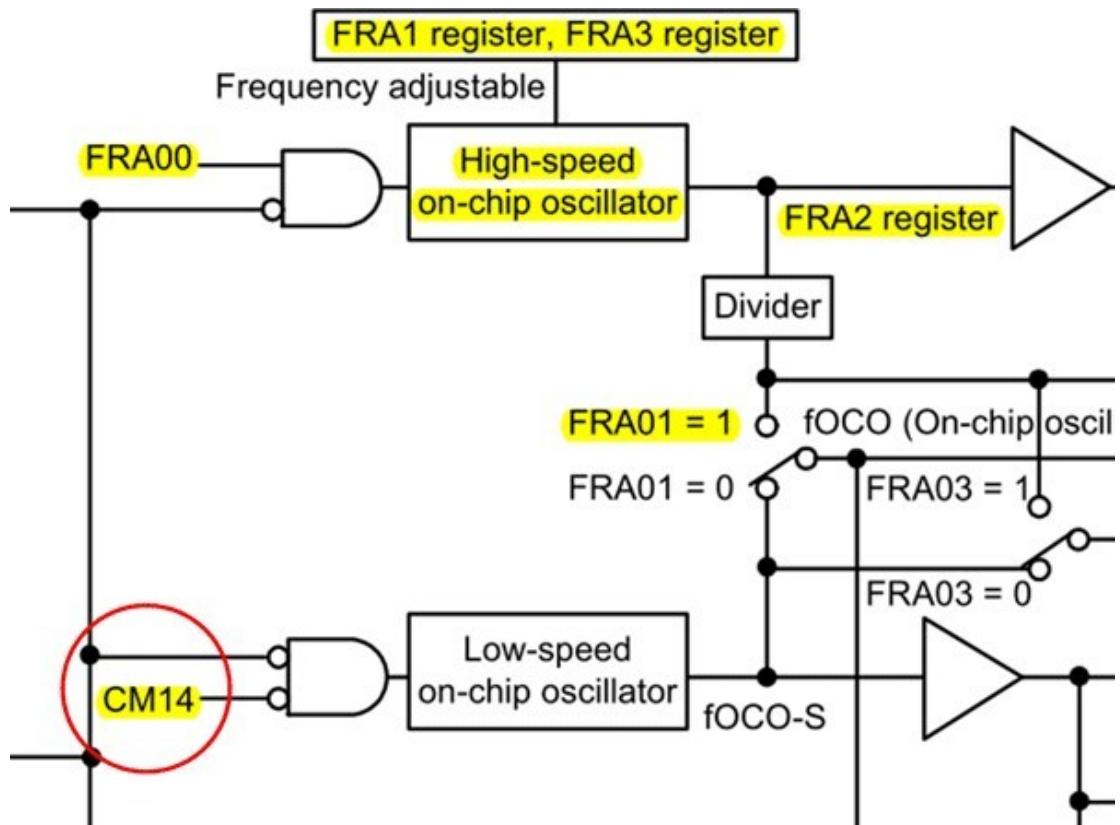
`CM14 = 0 ; /* Low-speed on-chip oscillator on */`

9.2.2 System Clock Control Register 1 (CM1)

Address 0007h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	CM17	CM16	—	CM14	CM13	CM12	CM11	CM10
After Reset	0	0	1	0	0	0	0	0

b4	CM14	Low-speed on-chip oscillator stop bit (3, 4)	0: Low-speed on-chip oscillator on 1: Low-speed on-chip oscillator off
b5	—	Reserved bit	Set to 1.
b6	CM16	CPU clock division select bit 1 (1)	b7 b6 0 0: No division mode 0 1: Divide-by-2 mode 1 0: Divide-by-4 mode 1 1: Divide-by-16 mode
b7	CM17		



(3) Then set the divider to most fast division.

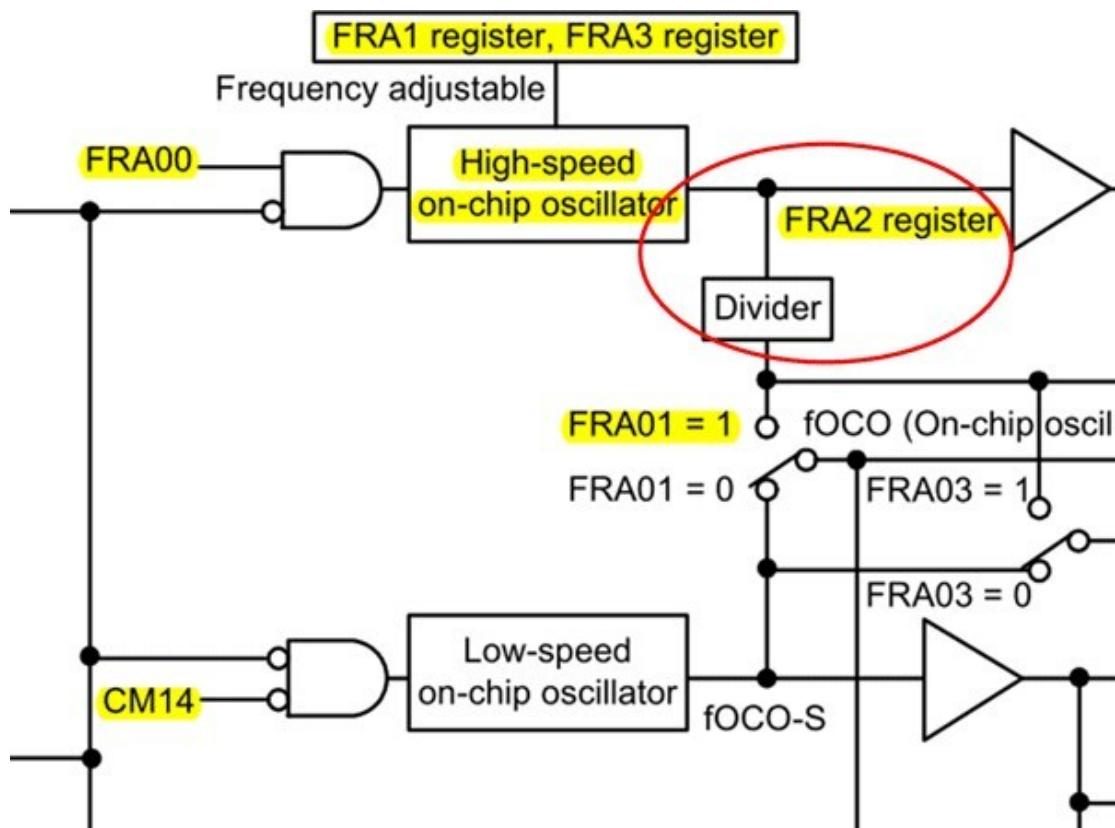
FRA2 = 0x00; /* Select Divide-by-2 mode */

9.2.8 High-Speed On-Chip Oscillator Control Register 2 (FRA2)

Address 0025h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	FRA22	FRA21	FRA20
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	FRA20	High-speed on-chip oscillator frequency switching bit	Division selection
b1	FRA21		These bits select the division ratio for the speed on-chip oscillator clock. b2 b1 b0
b2	FRA22		0 0 0: Divide-by-2 mode 0 0 1: Divide-by-3 mode 0 1 0: Divide-by-4 mode



(4) The High-Speed Oscillator to ON.

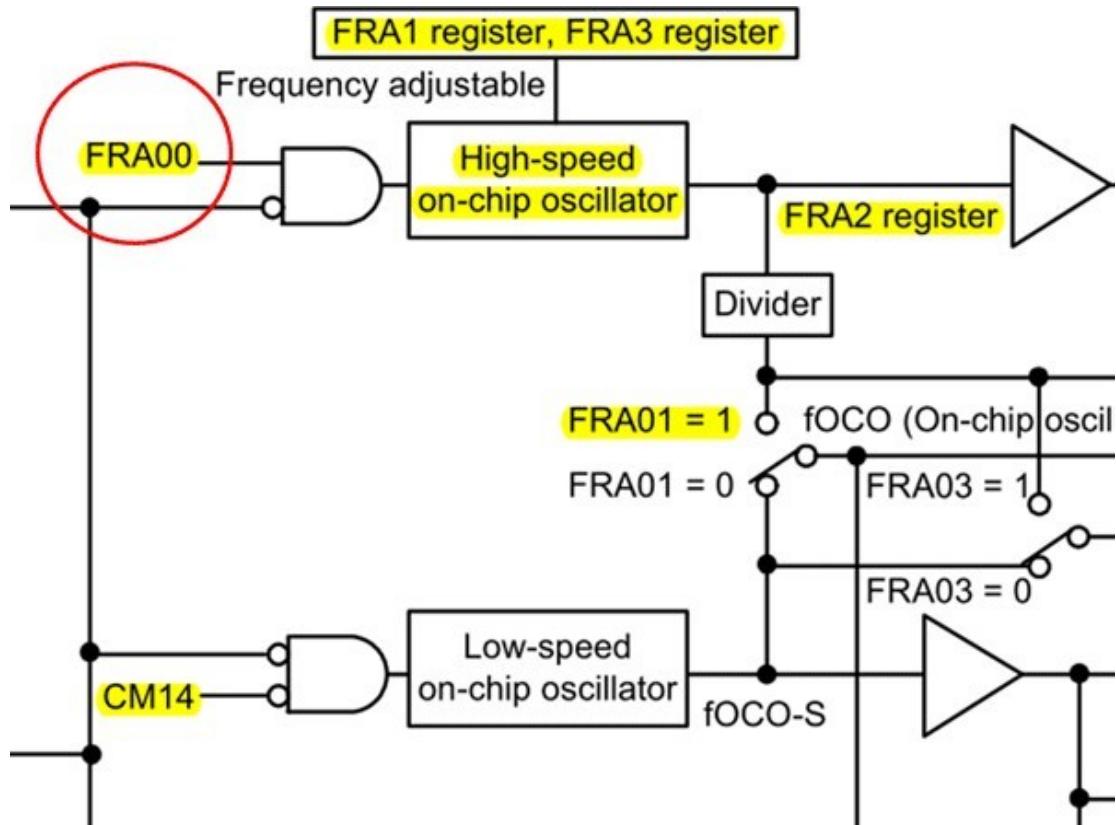
```
fra00 = 1; /* High-speed on-chip oscillator on */
```

9.2.6 High-Speed On-Chip Oscillator Control Register 0 (FRA0)

Address 0023h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	FRA03	—	FRA01	FRA00
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	FRA00	High-speed on-chip oscillator enable bit	0: High-speed on-chip oscillator off 1: High-speed on-chip oscillator on
b1	FRA01	High-speed on-chip oscillator select bit (1)	0: Low-speed on-chip oscillator selected 1: High-speed on-chip oscillator selected

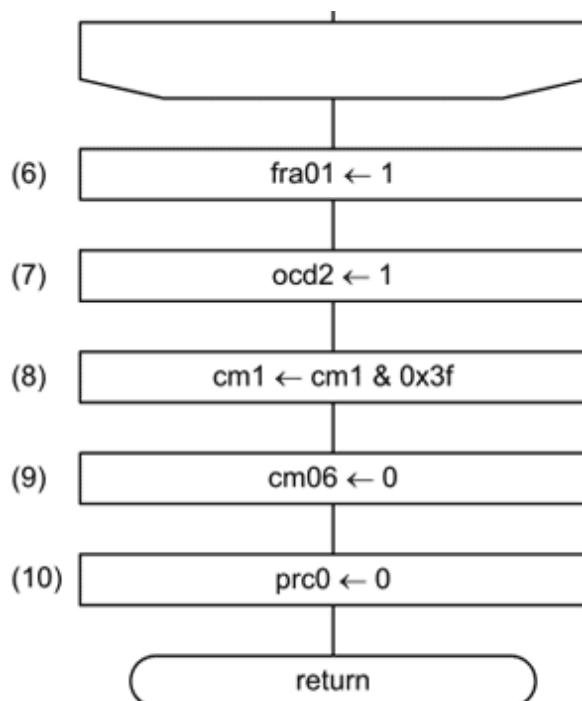


(5) Until the high-speed oscillator is stable, wait a little.

```
// ---- Wait until oscillation stabilizes ----  
// This setting is an example of waiting time for the  
// oscillation stabilization. Please evaluate the time  
// for oscillation stabilization by a user.
```

```
for ( int i = 0 ; i <= 2040; i++ ) ;
```

Flowchart of the second half



(6) Switch to the High-Speed Oscillator.

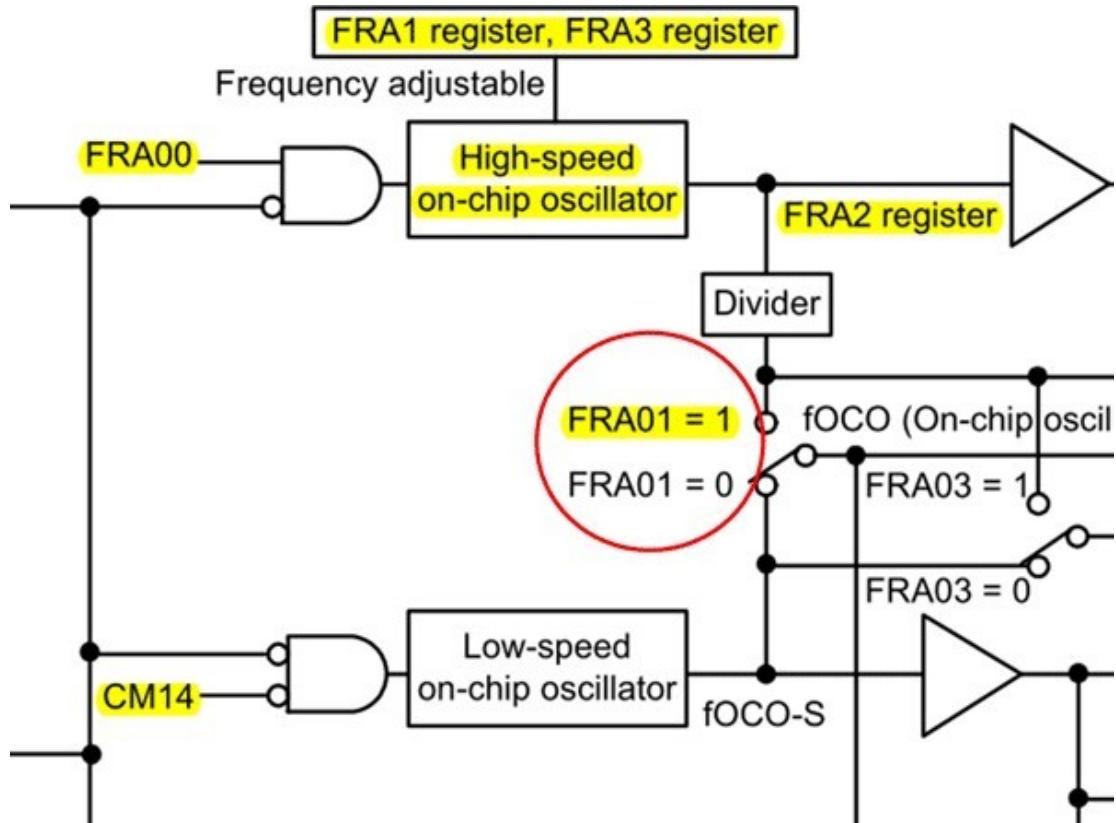
```
FRA01 = 1; /* Select High-speed on-chip oscillator*/
```

9.2.6 High-Speed On-Chip Oscillator Control Register 0 (FRA0)

Address 0023h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	FRA03	—	FRA01	FRA00
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	FRA00	High-speed on-chip oscillator enable bit	0: High-speed on-chip oscillator off 1: High-speed on-chip oscillator on
b1	FRA01	High-speed on-chip oscillator select bit (1)	0: Low-speed on-chip oscillator selected 1: High-speed on-chip oscillator selected



(7) Use the On-chip oscillator as system clock.

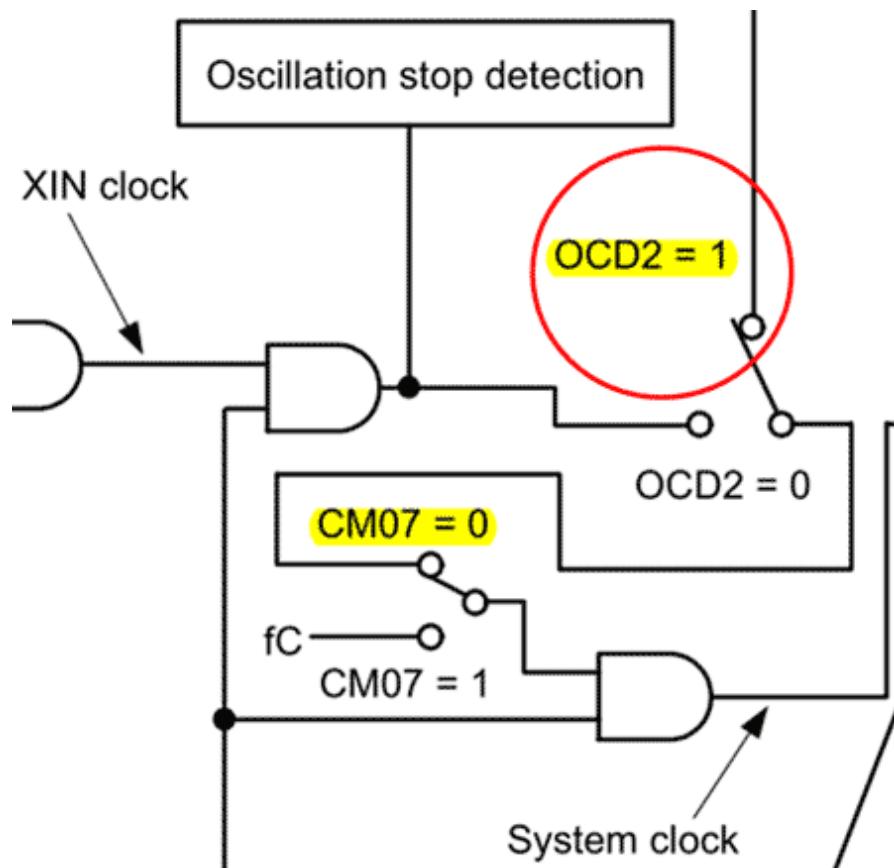
`OCD2 = 1; //System clock : On-chip oscillator clock selected`

9.2.4 Oscillation Stop Detection Register (OCD)

Address 000Ch

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	OCD3	OCD2	OCD1	OCD0
After Reset	0	0	0	0	0	1	0	0

Bit	Symbol	Bit Name	Function
b0	OCD0	Oscillation stop detection enable bit (6)	0: Oscillation stop detection function disabled 1: Oscillation stop detection function enabled
b1	OCD1	Oscillation stop detection interrupt enable bit	0: Disabled (1) 1: Enabled
b2	OCD2	System clock select bit (3)	0: XIN clock selected (6) 1: On-chip oscillator clock selected (2)



(8) No division of the system clock.

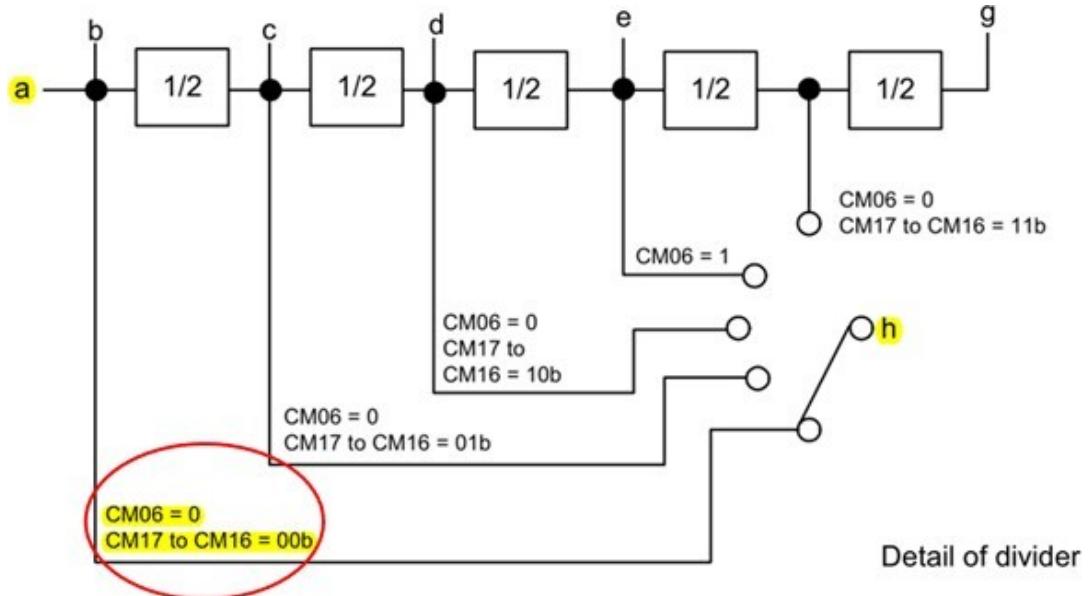
`CM1 &= 0x3f;` //No division mode in CPU clock division

9.2.2 System Clock Control Register 1 (CM1)

Address 0007h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	CM17	CM16	—	CM14	CM13	CM12	CM11	CM10
After Reset	0	0	1	0	0	0	0	0

b4	CM14	Low-speed on-chip oscillator stop bit (3, 4)	0: Low-speed on-chip oscillator on 1: Low-speed on-chip oscillator off
b5	—	Reserved bit	Set to 1.
b6	CM16	CPU clock division select bit 1 (1)	b7 b6 0 0: No division mode 0 1: Divide-by-2 mode 1 0: Divide-by-4 mode 1 1: Divide-by-16 mode
b7	CM17		



(9) Enable the CM16, CM17 bit.

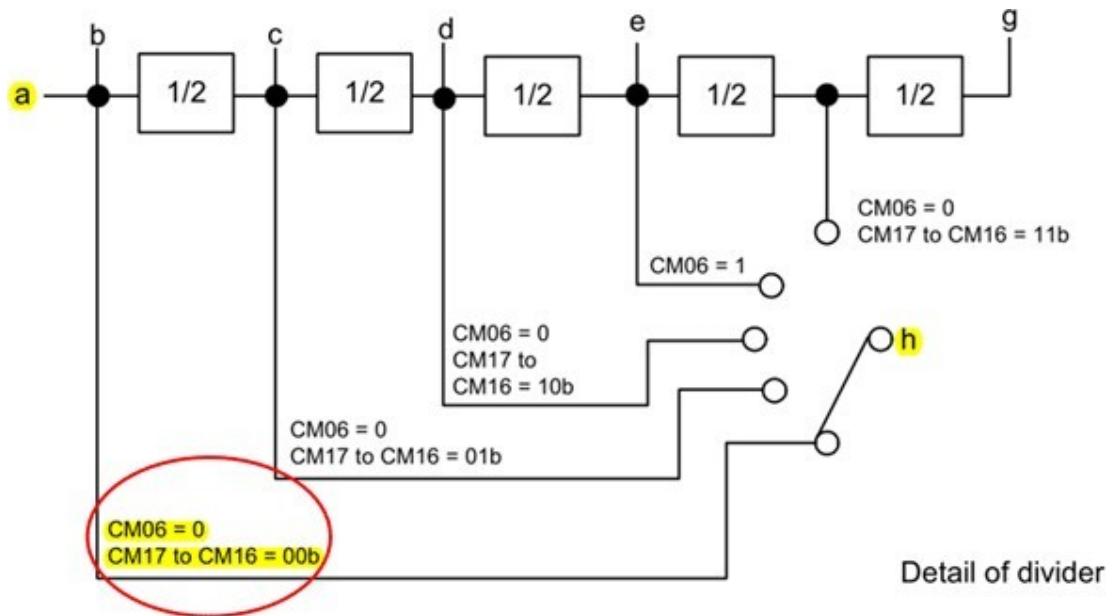
```
cmo6 = 0; /* CM16 and CM17 enabled */
```

9.2.1 System Clock Control Register 0 (CM0)

Address 0006h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	CM07	CM06	CM05	CM04	CM03	CM02	—	—
After Reset	0	0	1	0	1	0	0	0

b6	CM06	CPU clock division select bit 0 (4) 0: Bits CM16 and CM17 in CM1 register enabled 1: Divide-by-8 mode
b7	CM07	XIN, XCIN clock select bit (7) 0: XIN clock 1: XCIN clock



Reduce the power consumption by stopping the Low-speed on-chip oscillator.

```
CM14 = 1; /* Low-speed on-chip oscillator off */
```

(10) Since the register setting is finished, protect the register.

```
prco = 0; /* Protect on */
```

10.1 Register

10.1.1 Protect Register (PRCR)

Address 000Ah

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	PRC3	PRC2	PRC1	PRC0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PRC0	Protect bit 0	Enables writing to registers CM0, CM1, CM3, OCD, FRA0, FRA1, FRA2, and FRA3. 0: Write disabled 1: Write enabled (2)

Incidentally, let's reduce the power consumption by stopping the function that does not use.

Timer B is not used.

18.2.4 Timer RB Mode Register (TRBMR)

Address 010Bh

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	TCKCUT	—	TCK1	TCK0	TWRC	—	TMOD1	TMOD0
After Reset	0	0	0	0	0	0	0	0

b7	TCKCUT	Timer RB count source cutoff bit (1)	0: Provides count source 1: Cuts off count source
----	--------	--------------------------------------	--

```
trbmr = 0x80; /* Timer RB count source cutoff */
```

In addition, timer C and SSU, I₂C are not used. (See next page)

```
mstcr = 0x28; // SSU, I2C and Timer RC to Standby mode
```

20.6.1 Module Standby Control Register (MSTCR)

Address 0008h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	MSTTRC	MSTTRD	MSTIIC	—	—	—
After Reset	0	0	0	0	0	0	0	0
b3	MSTIIC	SSU, I ² C bus standby bit						0: Active 1: Standby ⁽¹⁾
b4	MSTTRD	Timer RD standby bit						0: Active 1: Standby ^(2, 3)
b5	MSTTRC	Timer RC standby bit						0: Active 1: Standby ⁽⁴⁾

(11) In summary, program looks like this.

```
static void mcu_init()
{
    //Set High-speed on-chip oscillator clock to System clock
    prco = 1;      /* Protect off */      /*CM0,CM1,CM3,*/
    /* OCD,FRAo,FRA1,FRA2,FRA3*/
    CM14 = 0;      /* Low-speed on-chip oscillator on */
    FRA2 = 0x00;   /* Selects Divide-by-2 mode */
    fra00 = 1;     /* High-speed on-chip oscillator on */
    for ( int n = 0; n <= 2040; n++);
        // This setting is an example of waiting time for the
        // oscillation stabilization. Please evaluate the time
        // for oscillation stabilization by a user.
    FRAO1 = 1; // Selects High-speed on-chip oscillator
```

```

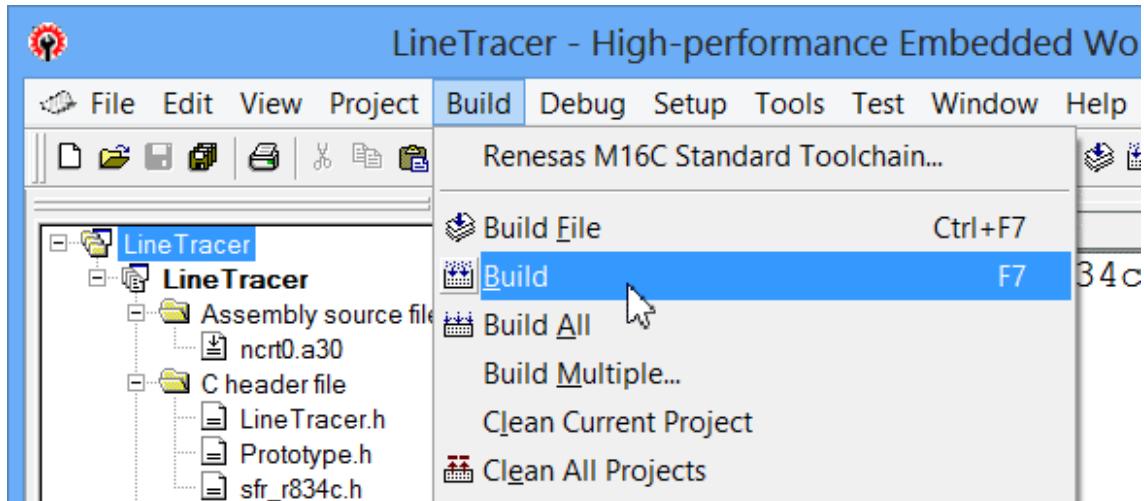
OCD2 = 1;
    //System clock : On-chip oscillator clock selected
CM1 &= 0x3F;
    //No division mode in system clock division
cm06 = 0;      /* CM16 and CM17 enable */
CM14 = 1;      /* Low-speed on-chip oscillator off */
prco = 0;      /* Protect on */
trbmr = 0x80;  /*Timer RB count source cutoff*/
mstcr = 0x28;  /* SSU, I2C and Timer RC*/
                /*to Standby mode*/
}

```

Function	
Timer RA	Use for A/D
Timer RB, RC, RE	Not use
Timer RD	Use for Motor
Watch dog	Not use
A/D	Use for Sensor
D/A	Not use
UART	Use for PC
SSU, I2C, LIN	Not use
DTC	Not use
Voltage Detection	Not use

[11. Build project]

Let's build your project.



If there is an error, please check the spelling mistake.

```
ROMDATA SECTION: 00000028 Byte(s)
PROGRAM SECTION: 00000009d Byte(s)
Optimizing Linkage Editor Completed
Phase OptLinker finished

Build Finished
0 Errors, 0 Warnings
```

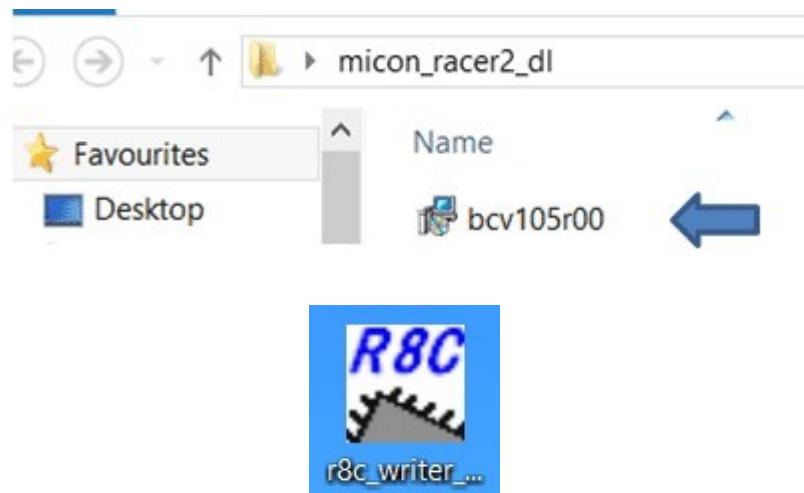
The screenshot shows the "Build" output window. It displays the results of a successful build process. The output text includes information about ROMDATA and PROGRAM sections, the completion of the Optimizing Linkage Editor phase, and the final message "Build Finished" with "0 Errors, 0 Warnings". Below the text, there is a navigation bar with tabs for Build, Debug, Find in Files 1, Find in Files 2, Macro, Test, and Version. The "Build" tab is currently active. At the bottom, it says "Ready" and shows icons for desktop and network connections, with "Default1 desktop" selected.

[12. Software for writing]

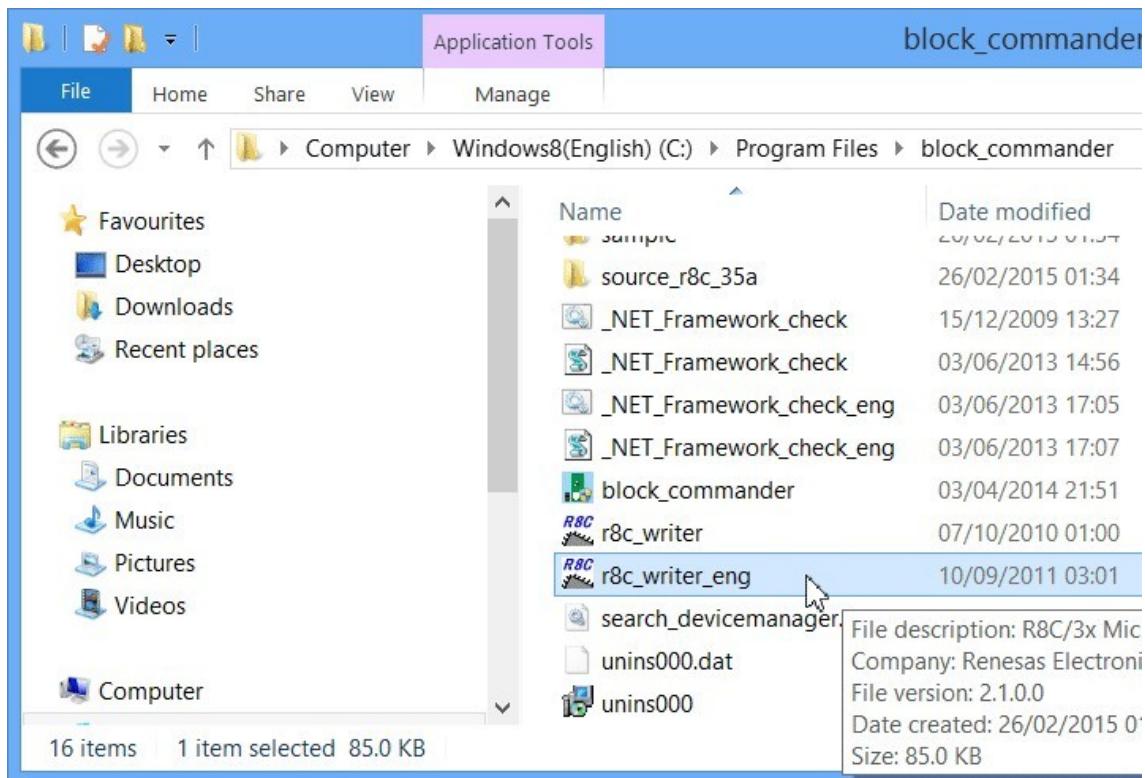
Get the writing software from the following site

http://www.marutsu.co.jp/contents/shop/marutsu/include/large_order/img/MR2_img/dl/micon_racer2_dl.zip

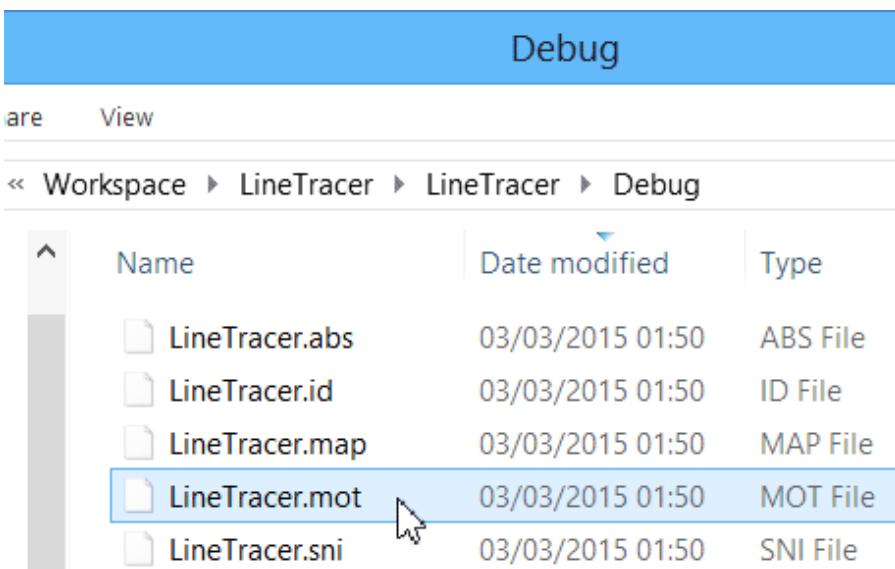
After unzipping, execute “bcv105r00.exe”.

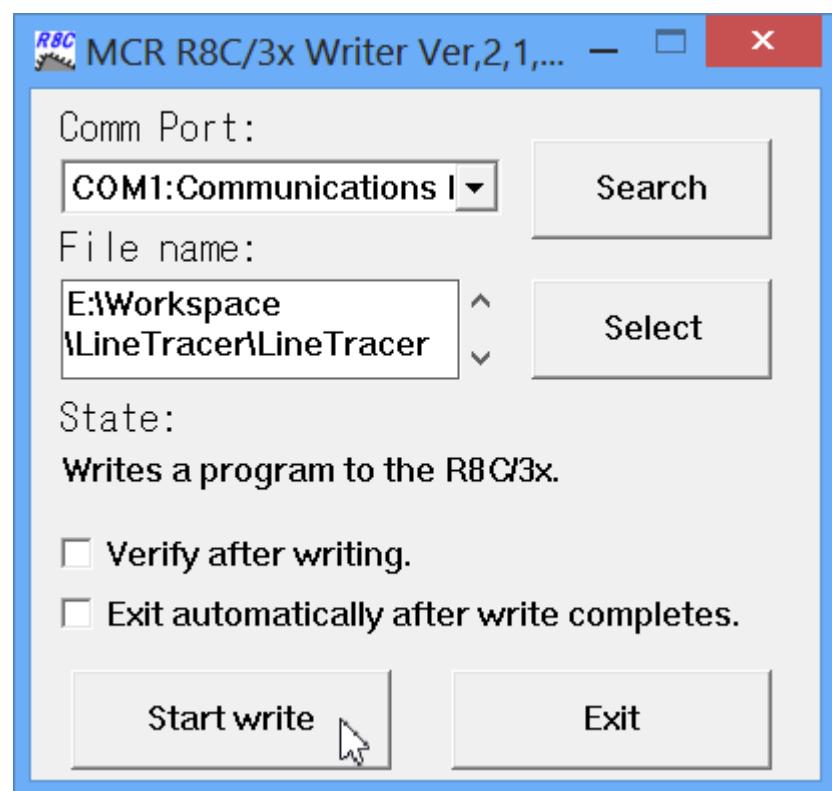
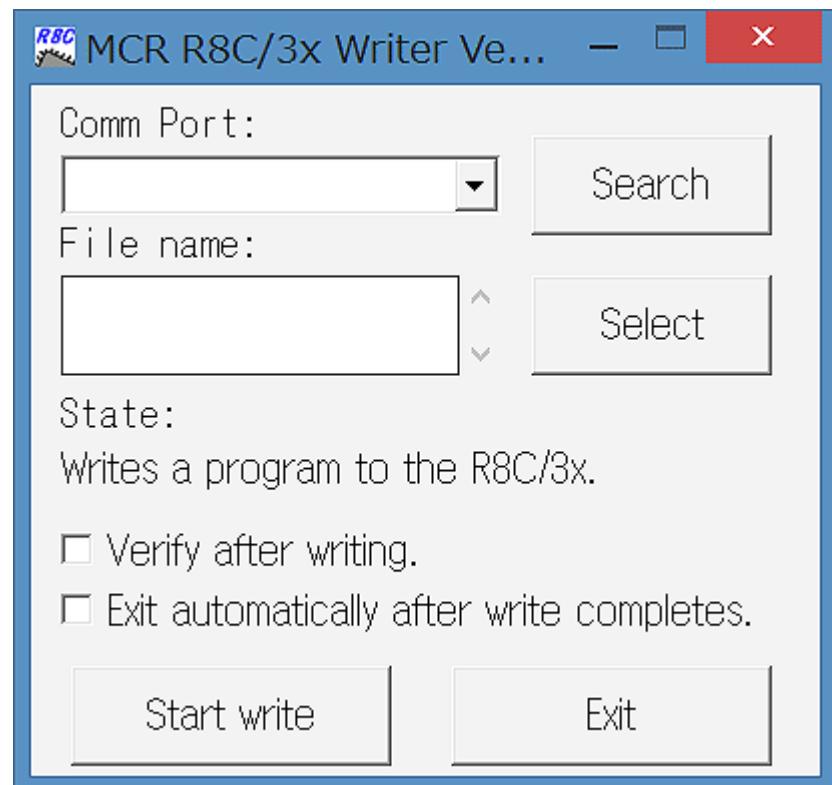


Move “r8c_writer_eng.exe” in installation folder to the desktop.



Writes “LineTracer.mot” file by running
“r8c_writer_eng.exe”.





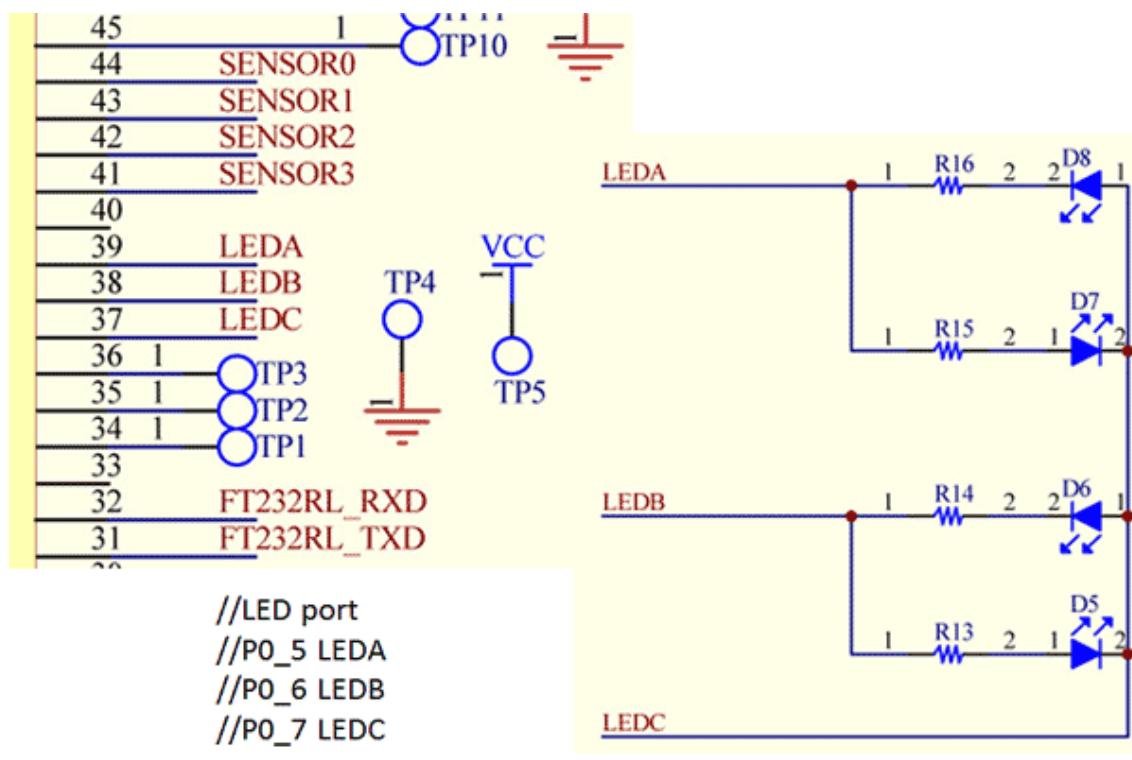
{Question 1}

Why is a low-speed oscillator prepared?

[2] LED and Switch

[1.Circuit of LED]

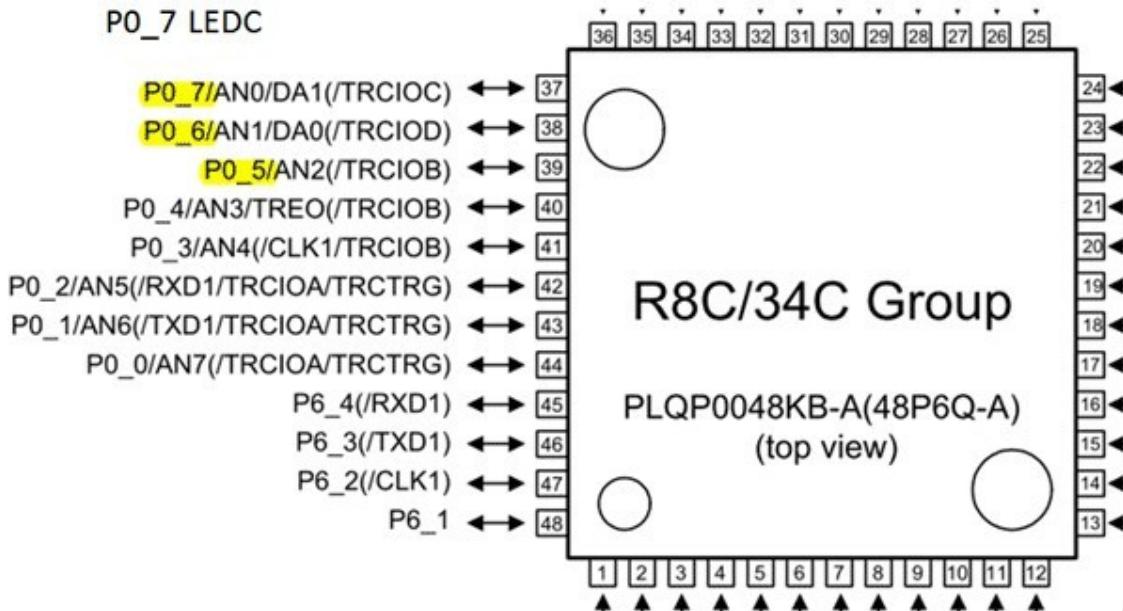
I am sure you want know whether your program is running.
Then, let's create the program to turn on the LED.
First, let's look at the circuit diagram.
The part of the LED is not a little ordinary.



LEDC=0 LEDC=1

LEDA=D7 data LEDA=~D8 data
LEDB=D5 data LEDB=~D6 data

LED port
 P0_5 LEDA
 P0_6 LEDB
 P0_7 LEDC



[2.PORT_init]

(1) Add to “Hwsetup.cpp” file.

```

static void mcu_init(void);
static void port_init(void);

void Hwsetup(void)
{
  asm("FCLR I");      /* Interrupt disabled */

  mcu_init();
  port_init();

  asm("FSET I");      /* Interrupt enable */
}

```

```
static void port_init()
{
}
```

Add in “port_init()” function.

(2) Digital input and output are accomplished by reading and writing to the Pi register.

7.4.2 Port Pi Register (Pi) (i = 0 to 4 or 6)

Address 00E0h(P0), 00E1h(P1), 00E4h(P2), 00E5h(P3 (1)), 00E8h(P4 (2)), 00ECh(P6),

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	Pi_7	Pi_6	Pi_5	Pi_4	Pi_3	Pi_2	Pi_1	Pi_0
After Reset	X	X	X	X	X	X	X	X

Bit	Symbol	Bit Name	Function
b0	Pi_0	Port Pi_0 bit	0: "L" level 1: "H" level
b1	Pi_1	Port Pi_1 bit	
b2	Pi_2	Port Pi_2 bit	
b3	Pi_3	Port Pi_3 bit	
b4	Pi_4	Port Pi_4 bit	
b5	Pi_5	Port Pi_5 bit	
b6	Pi_6	Port Pi_6 bit	
b7	Pi_7	Port Pi_7 bit	

```
po = 0; /* LED port off */
```

(3) The PDi register selects whether I/O ports are used for input or output. Each bit in the PDi register corresponds to one port.

7.4.1 Port Pi Direction Register (PDi) ($i = 0$ to 4 or 6)

Address 00E2h (PD0 (1)), 00E3h (PD1), 00E6h (PD2), 00E7h (PD3 (2)), 00EAh (PD4 (3)), 00EE

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	PDi_7	PDi_6	PDi_5	PDi_4	PDi_3	PDi_2	PDi_1	PDi_0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PDi_0	Port Pi_0 direction bit	0: Input mode (functions as an input port) 1: Output mode (functions as an output port)
b1	PDi_1	Port Pi_1 direction bit	
b2	PDi_2	Port Pi_2 direction bit	
b3	PDi_3	Port Pi_3 direction bit	
b4	PDi_4	Port Pi_4 direction bit	
b5	PDi_5	Port Pi_5 direction bit	
b6	PDi_6	Port Pi_6 direction bit	
b7	PDi_7	Port Pi_7 direction bit	

Po_5 = LEDA, Po_6 = LEDB, Po_7 = LEDC

(4) The protection function protects important registers from being easily overwritten if a program runs out of control.

10.1.1 Protect Register (PRCR)

Address 000Ah

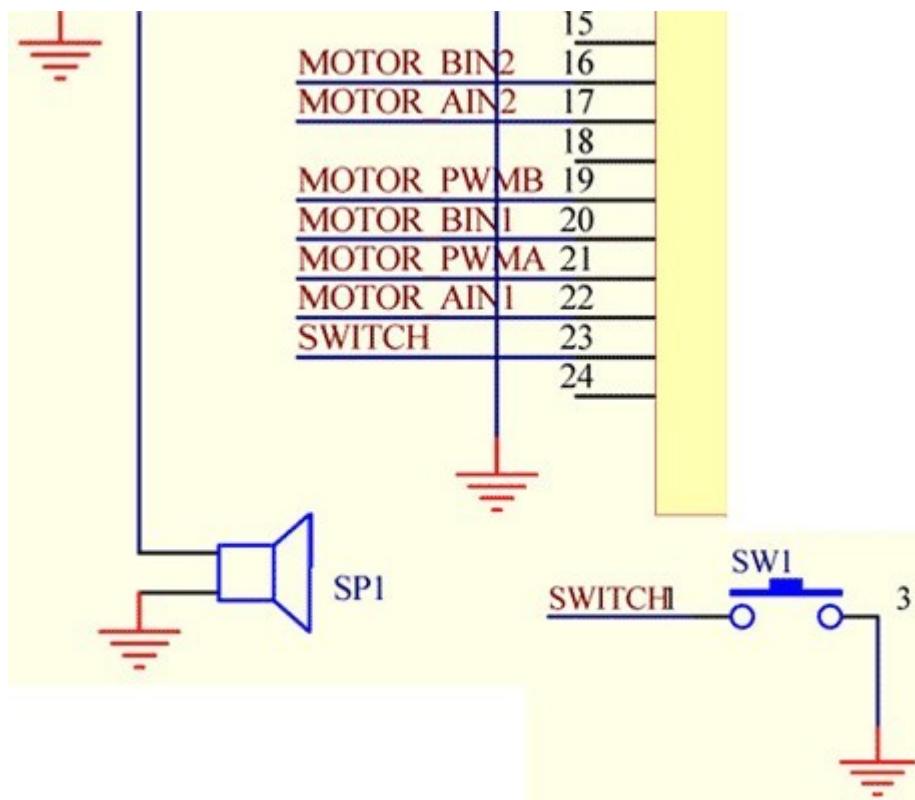
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	PRC3	PRC2	PRC1	PRC0
After Reset	0	0	0	0	0	0	0	0
b2	PRC2	Protect bit 2	Enables writing to the PD0 register. 0: Write disabled 1: Write enabled (1)					

The PRC2 bit is set to 0 after setting it to 1 (write enabled) and writing to the SFR area. Change the register protected by the PRC2 bit with the next instruction after that used to set the PRC2 bit to 1. Do not allow interrupts or DTC activation between the instruction to set to the PRC2 bit to 1 and the next instruction.

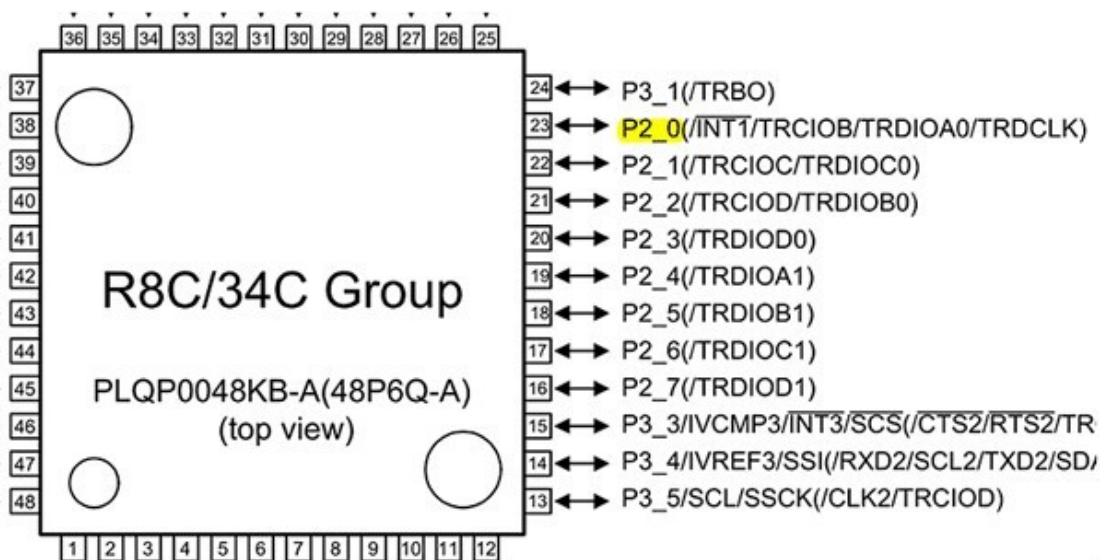
```
PRC2 = 1;          /* Protect off */
pdo |= 0xE0;      /* LED port direction = output */
```

[3.Digital Input]

Incidentally let's add digital input program.



A switch is connected at Pin23 (P2_0).



(1) P2_o set to input port.

7.4.1 Port Pi Direction Register (PDi) (i = 0 to 4 or 6)

Address 00E2h (PD0 (1)), 00E3h (PD1), 00E6h (PD2), 00E7h (PD3 (2)), 00EAh (PD4 (3)), 00EEh (PD6),

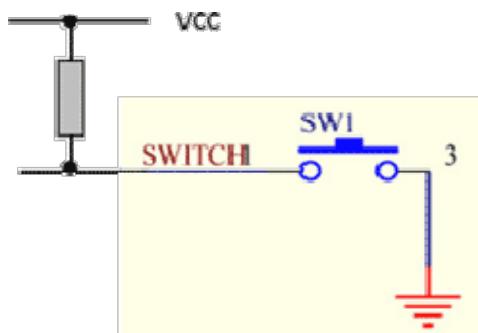
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	PDi_7	PDi_6	PDi_5	PDi_4	PDi_3	PDi_2	PDi_1	PDi_0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PDi_0	Port Pi_0 direction bit	0: Input mode (functions as an input port) 1: Output mode (functions as an output port)
b1	PDi_1	Port Pi_1 direction bit	

```
P2 = 0 ;      //P2_o=Switch Input,  
             //P2_1...P2_7=Motor Output
```

(2) Usually, we use the pulling up resistance to stabilize the input voltage.

However, this MPU has built-in pulling up resistance. We use it.



7.4.17 Pull-Up Control Register 0 (PUR0)

Address 01E0h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	PU07	PU06	PU05	PU04	PU03	PU02	PU01	PU00
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PU00	P0_0 to P0_3 pull-up	0: Not pulled up 1: Pulled up (1)
b1	PU01	P0_4 to P0_7 pull-up	
b2	PU02	P1_0 to P1_3 pull-up	
b3	PU03	P1_4 to P1_7 pull-up	
b4	PU04	P2_0 to P2_3 pull-up	

puo4 = 1 ; //Pullup for Switch Input

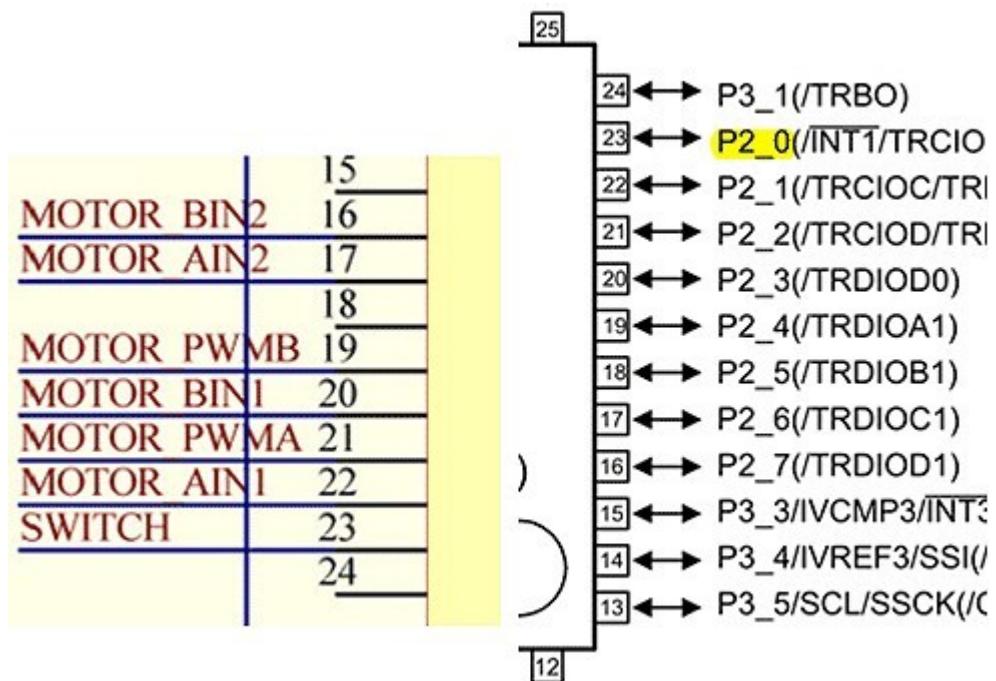
(3) We can understand the state of the switch when readthis register bit (p2_o).

7.4.2 Port Pi Register (Pi) (i = 0 to 4 or 6)

Address 00E0h(P0), 00E1h(P1), 00E4h(P2), 00E5h(P3 (1)), 00E8h(P4 (2)), 00ECh(P6),

Symbol	Pi_7	Pi_6	Pi_5	Pi_4	Pi_3	Pi_2	Pi_1	Pi_0	b0
After Reset	X	X	X	X	X	X	X	X	

Bit	Symbol	Bit Name	Function
b0	Pi_0	Port Pi_0 bit	0: "L" level 1: "H" level
b1	Pi_1	Port Pi_1 bit	



Since from P2_1 to P2_7 are assigned to the output of the motor, set to 0.

PD2 = 0xFE ; //P2_o Input, P2_1...P2_7 Output

(4) So far the code is ...

```
static void port_init()
{
    //Po_5  LEDA
    //Po_6  LEDB
    //Po_7  LEDC
    // Setting port registers */
    po = 0;      /* LED port off */
    // Setting port direction registers */
    PRC2 = 1;    /* Protect off */
    pdo |= 0xE0; /* LED port direction = output */

    // SWITCH Button & Motor Output
    P2 = 0;      /*P2_0=Switch Input,
                    //P2_1...P2_7=Motor Output
    puo4 = 1;    /*Pullup for Switch Input
    PD2 = 0xFE;  /*P2_0 Input, P2_1...P2_7 Output
}
```

[4.Test Program]

Let's make the Main program for test.
It is OK if LED changes on/off by the switch.

```
void main(void)
{
    Hwsetup();

    while(1)          /* Main processing */
    {
        //if SWITCH == ON
        if( P2_O == 0 )
        {
            po_5 = 1 ; //LED D7 ON
            po_6 = 1 ; //LED D5 ON
            po_7 = 0 ;
        }
        else
        {
            po_5 = 0 ; //LED D8 ON
            po_6 = 0 ; //LED D6 ON
            po_7 = 1 ;
        }
    }
}
```

{Question 2}

The output port has something to spare, but why did they make the output port a complicated circuit?

{Question 3}

LEDC=0	LEDC=1
LEDA=D7 data LEDB=D5 data	LEDA= \sim D8 data LEDB= \sim D6 data

LEDC=0

LEDC=1

LEDA=D7 data

LEDB=D5 data

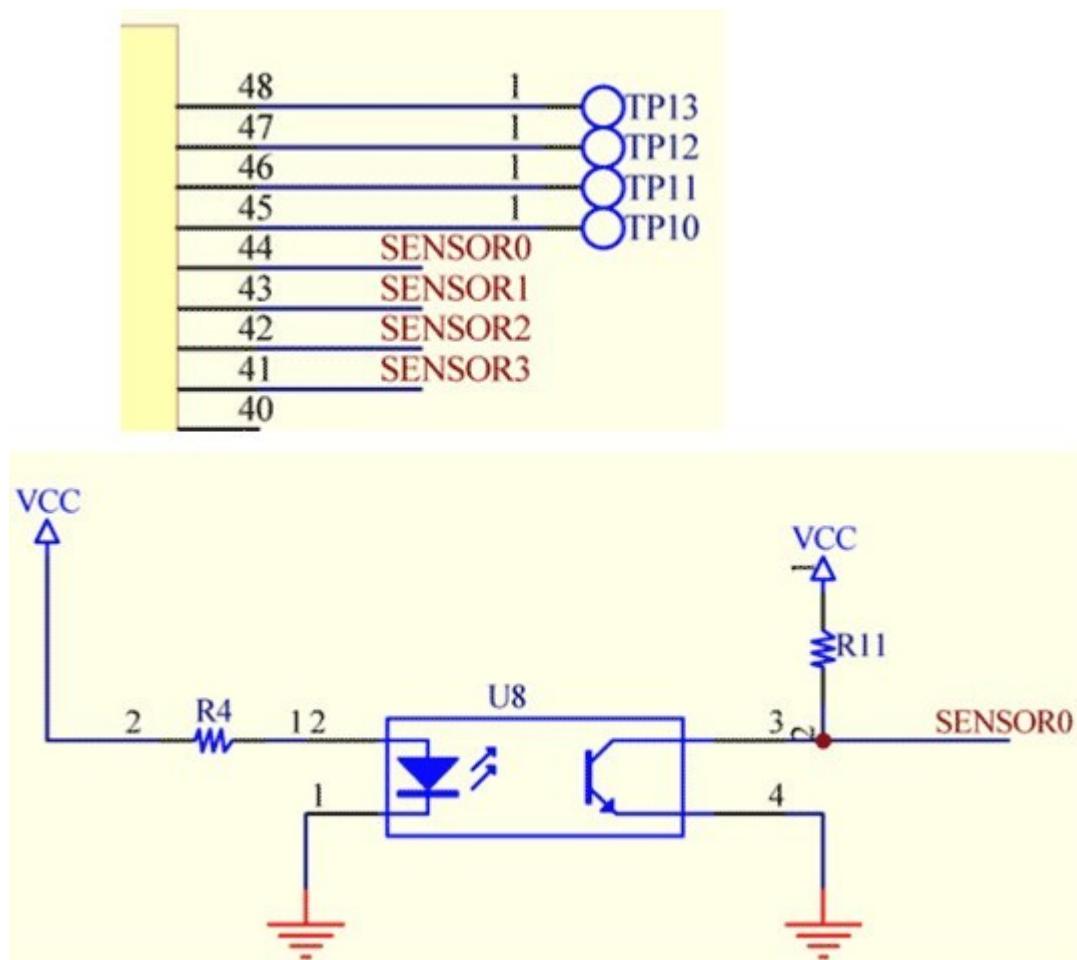
LEDA= \sim D8 data

LEDB= \sim D6 data

As mentioned above, why tilde is used?

[3] A/D conversion

Reading sensor values by AD converter.



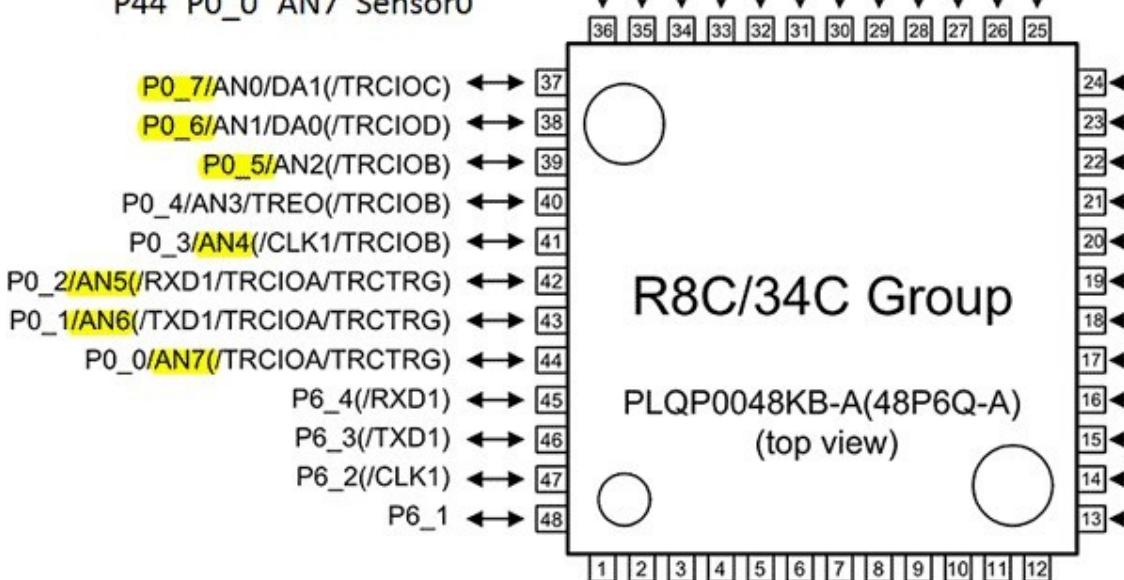
Type of the sensor : ITR8307-S18-TR8

From the manual of the maker...

ITR8307/L24/TR8 is a light reflection switch which includes a GaAs IR-LED transmitter and a NPN photo-transistor with a high photosensitive receiver for short distance, operating in the infrared range.

Both components are mounted side- by- side in a plastic package.

P41 P0_3 AN4 Sensor3
 P42 P0_2 AN5 Sensor2
 P43 P0_1 AN6 Sensor1
 P44 P0_0 AN7 Sensor0



[1.Hwsetup]

(1) Add to “Hwsetup.cpp” file as follows.

```

static void mcu_init(void);
static void port_init(void);
static void ad_init(void);

void Hwsetup(void)
{
  asm("FCLR I");      /* Interrupt disabled */
  mcu_init();
  port_init();
  ad_init();
  asm("FSET I");      /* Interrupt enable */
}

```

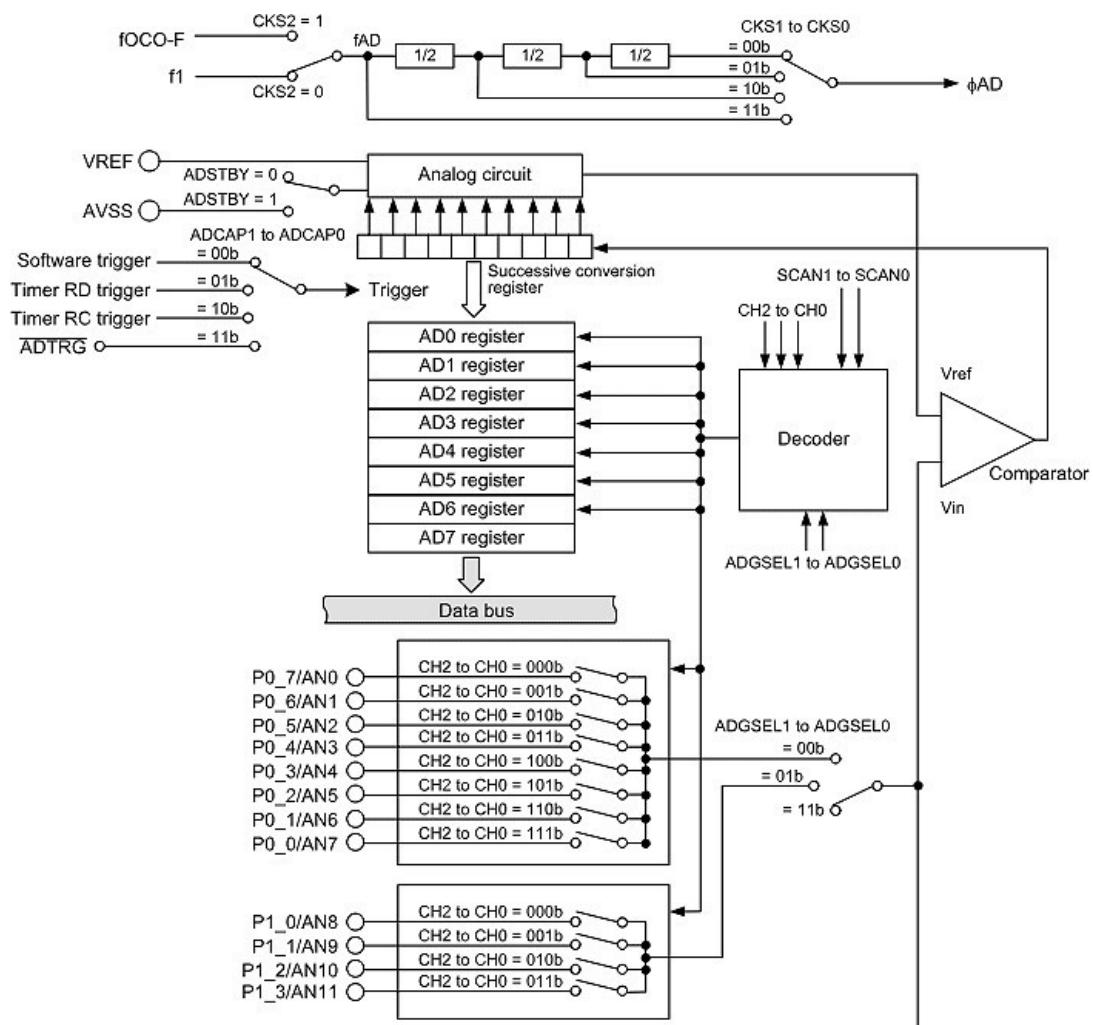
```
static void ad_init()
```

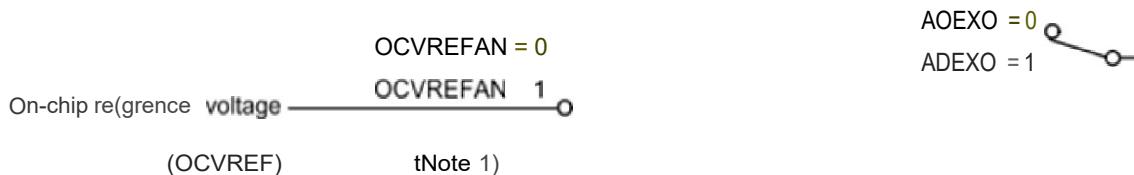
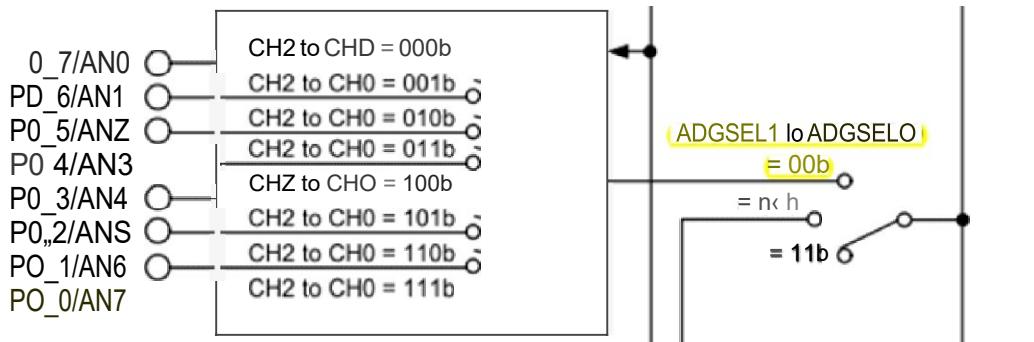
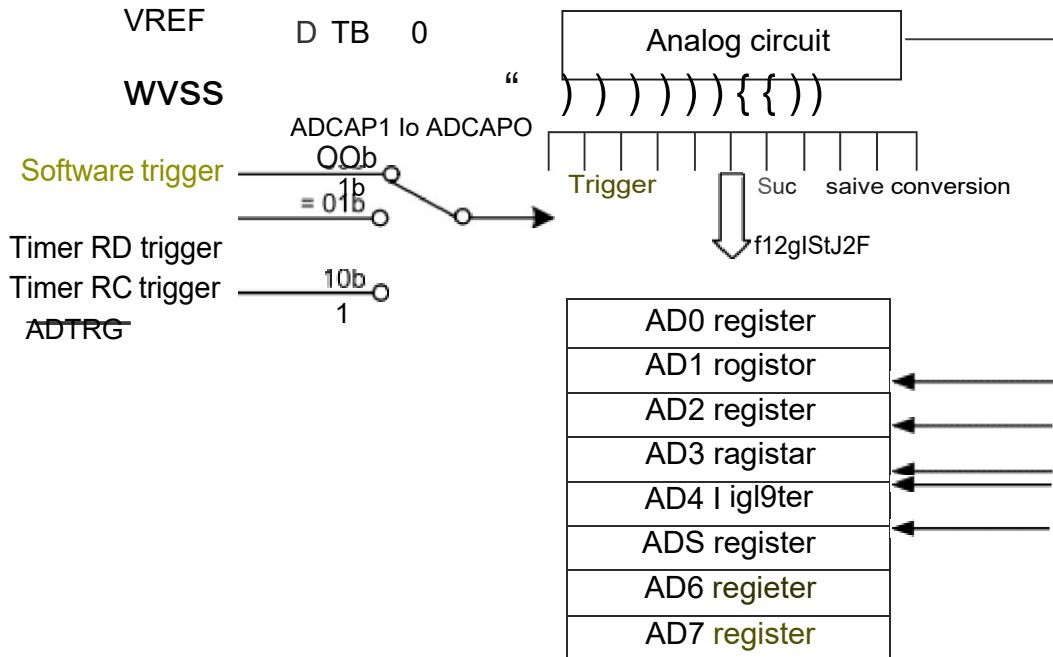
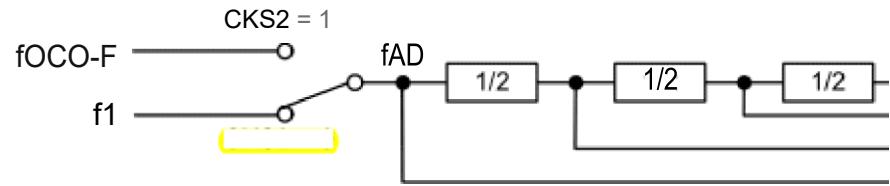
```
{
```

```
}
```

We describe in “ad_init()” function.

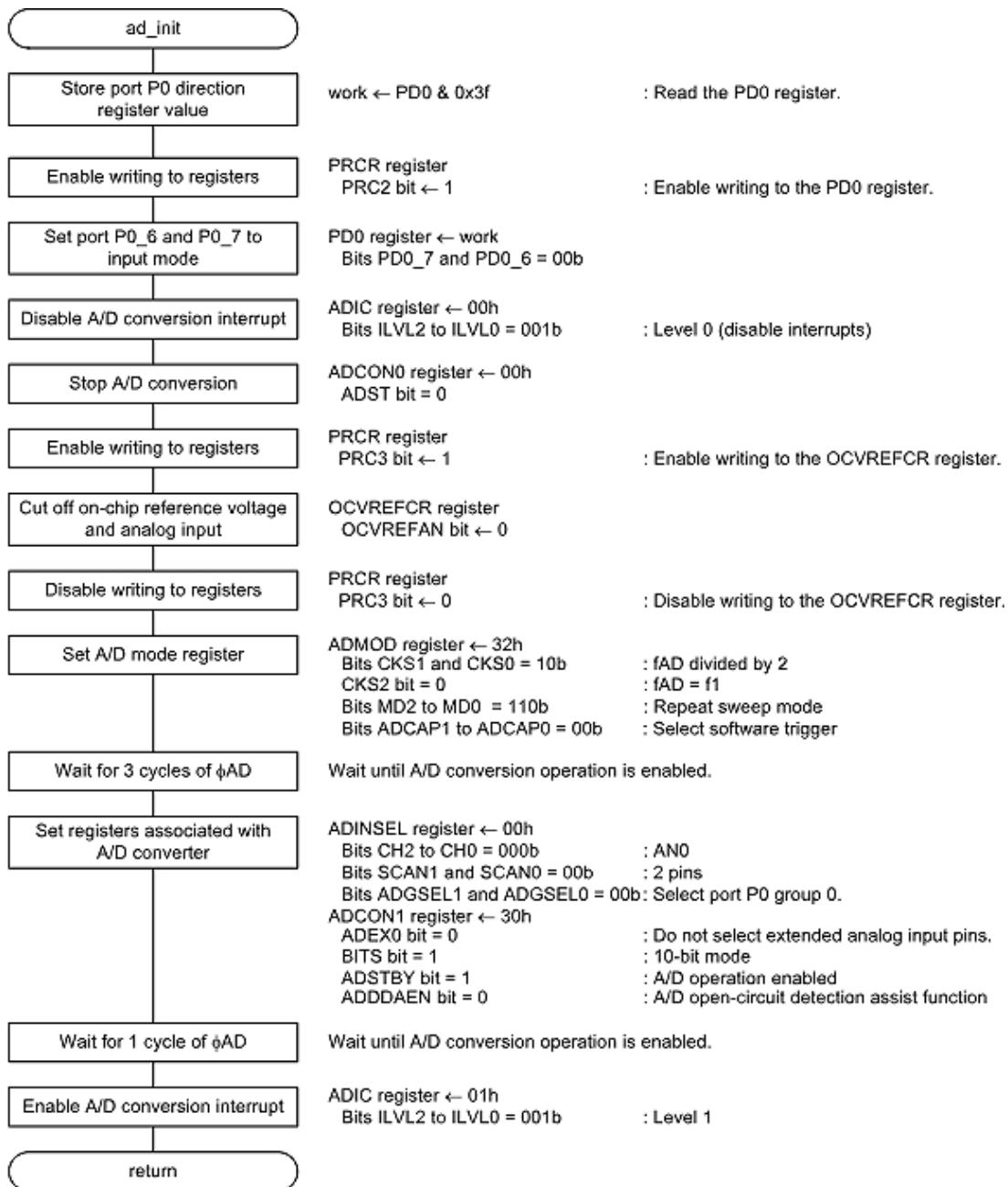
[2. Block Diagram]

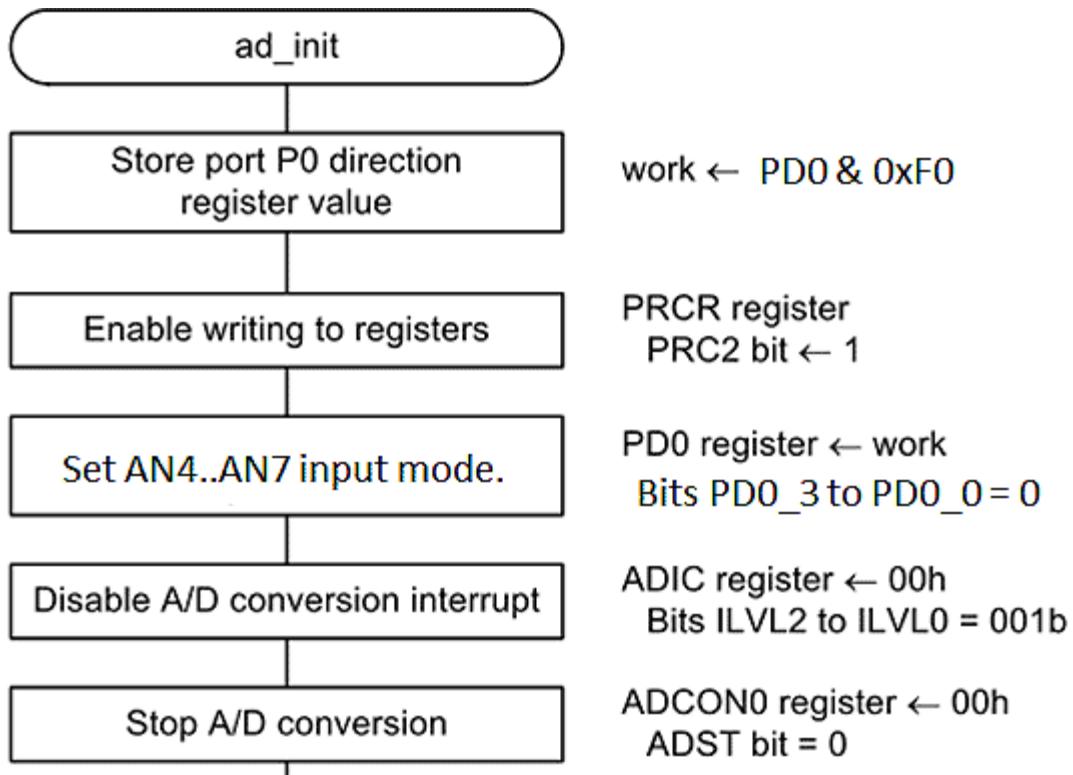




[3.ad_init]

This is a sample program provided by Renesas Electronics Corp.
We follow this procedure.





(1) Port Direction

7.4.1 Port Pi Direction Register (PDi) (i = 0 to 4 or 6)

Address 00E2h (PD0 (1)), 00E3h (PD1), 00E6h (PD2), 00E7h (PD3 (2)), 00EAh (PD4 (3)), 00EEh (PD6 (4))

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	PDi_7	PDi_6	PDi_5	PDi_4	PDi_3	PDi_2	PDi_1	PDi_0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PDi_0	Port Pi_0 direction bit	0: Input mode (functions as an input port) 1: Output mode (functions as an output port)
b1	PDi_1	Port Pi_1 direction bit	
b2	PDi_2	Port Pi_2 direction bit	
b3	PDi_3	Port Pi_3 direction bit	
b4	PDi_4	Port Pi_4 direction bit	
b5	PDi_5	Port Pi_5 direction bit	
b6	PDi_6	Port Pi_6 direction bit	
b7	PDi_7	Port Pi_7 direction bit	

1. Write to the PD0 register with the next instruction after that used to set the PRC2 bit in the PRCR register to 1 (write enabled).

```

unsigned char work; /* Variable for PDo register */

work = pdo & oxfo; /* Read PDo register */
PRC2 = 1; /* Protect off */
pdo = work;
/* Po_3/AN4...Po_o/AN7 port direction = input */

```

(2)INTERRUPT

11.2.1 Interrupt Control Register

(TREIC, S2TIC, S2RIC, KUPIC, ADIC, S0TIC, S0RIC, S1TIC, S1TRBIC, U2BCNIC, VCMP1IC, VCMP2IC)

Address 004Ah (TREIC), 004Bh (S2TIC), 004Ch (S2RIC), 004Dh (KUPIC), 004Eh (ADIC),
 0051h (S0TIC), 0052h (S0RIC), 0053h (S1TIC), 0054h (S1RIC), 0056h (TRAIC),
 0058h (TRBIC), 005Eh (U2BCNIC), 0072h (VCMP1IC), 0073h (VCMP2IC),

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	IR	ILVL2	ILVL1	ILVL0
After Reset	X	X	X	X	X	0	0	0

Bit	Symbol	Bit Name	Function
b0	ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0: Level 0 (interrupt disabled) 0 0 1: Level 1 0 1 0: Level 2 0 1 1: Level 3 1 0 0: Level 4 1 0 1: Level 5 1 1 0: Level 6 1 1 1: Level 7
b1	ILVL1		
b2	ILVL2		

28.2.5 A/D Control Register 0 (ADCON0)

Address 00D6h

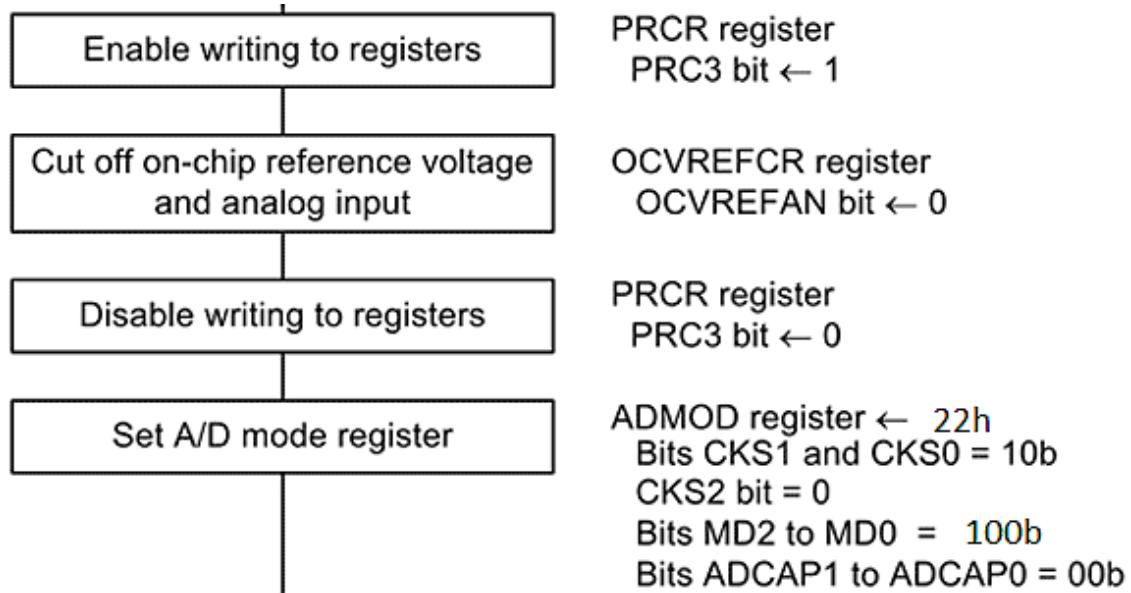
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	ADST
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	ADST	A/D conversion start flag	0: Stop A/D conversion 1: Start A/D conversion

```

adic = ox00; /* Disable A/D conversion interrupt */
adcono = ox00; /* Stop A/D conversion */

```



(3) Reference Voltage

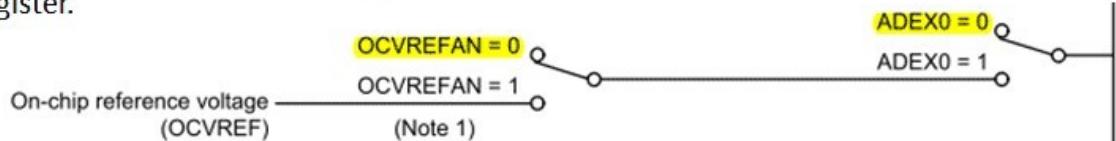
28.2.1 On-Chip Reference Voltage Control Register (OCVREFCR)

Address 0026h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	OCVREFAN
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	OCVREFAN	On-chip reference voltage to analog input connect bit (1)	0: On-chip reference voltage and analog input are cut off 1: On-chip reference voltage and analog input are connected

Set the PRC3 bit in the PRCR register to 1 (write enabled) before rewriting the OCVREFCR register.



```

//Cut off on-chip reference voltage and analog input
prc3 = 1;      /* Protect off */
ocvrefan = 0;
prc3 = 0;      /* Protect on */

```

(4) A/D Mode

28.2.3 A/D Mode Register (ADMOD)

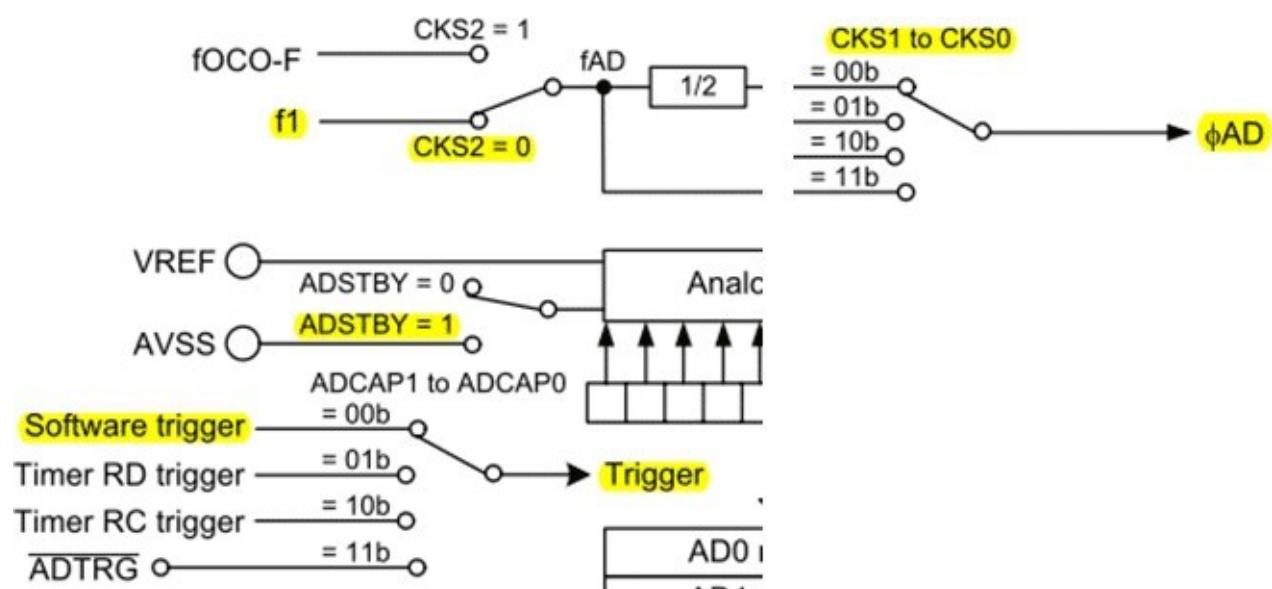
Address 00D4h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	ADCAP1	ADCAP0	MD2	MD1	MD0	CKS2	CKS1	CKS0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	CKS0	Division select bit	b1 b0 0 0: fAD divided by 8 0 1: fAD divided by 4 1 0: fAD divided by 2 1 1: fAD divided by 1 (no division)
b1	CKS1		
b2	CKS2	Clock source select bit (1)	0: Selects f1 1: Selects fOCO-F
b3	MD0	A/D operating mode select bit	b5 b4 b3 0 0 0: One-shot mode 0 0 1: Do not set. 0 1 0: Repeat mode 0 0 1 1: Repeat mode 1
b4	MD1		
b5	MD2		1 0 0: Single sweep mode 1 0 1: Do not set. 1 1 0: Repeat sweep mode 1 1 1: Do not set.
b6	ADCAP0	A/D conversion trigger select	b7 b6 0 0: A/D conversion starts by software trigger (ADCON0 register)
b7	ADCAP1	bit	

Note:

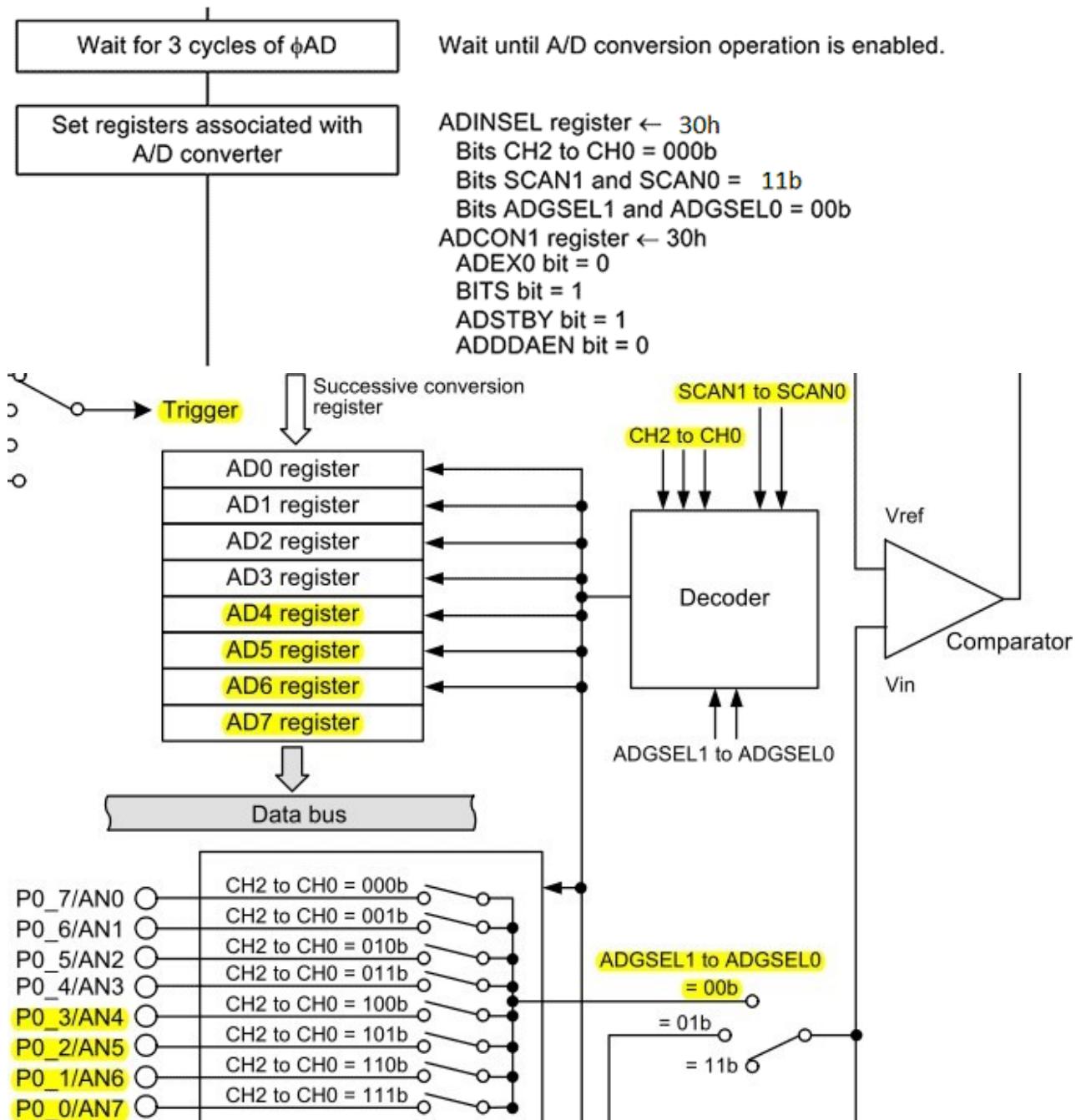
- When the CKS2 bit is changed, wait for 3 ϕ_{AD} cycles or more before starting A/D conversion.



```

/* ---- Set A/D mode --- */
admod = 0x22; //Division select : fAD divided by 2
                //Clock source : F1
//A/D operating mode : single sweep mode
// Start A/D conversion by software trigger

```



(5) A/D Input Select

28.2.4 A/D Input Select Register (ADINSEL)

Address 00D5h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	ADGSEL1	ADGSEL0	SCAN1	SCAN0	—	CH2	CH1	CH0
After Reset	1	1	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	CH0	Analog input pin select bit	Refer to Table 28.2 Analog Input Pin Selection
b1	CH1		
b2	CH2		
b3	—	Reserved bit	Set to 0.
b4	SCAN0	A/D sweep pin count select bit	b5 b4 0 0: 2 pins 0 1: 4 pins 1 0: 6 pins 1 1: 8 pins
b5	SCAN1		
b6	ADGSEL0	A/D input group select bit	b7 b6 0 0: Port P0 group selected 0 1: Port P1 group selected
b7	ADGSEL1		

Table 28.2 Analog Input Pin Selection

Bits CH2 to CH0	Bits ADGSEL1, ADGSEL0 = 00b
000b	AN0
001b	AN1
010b	AN2
011b	AN3
100b	AN4
101b	AN5
110b	AN6
111b	AN7

```
//When the CKS2 bit is changed, wait for 3 AD cycles
//or more before starting A/D conversion
for (int i = 0; i < 10; i++){
}
/* ---- Set A/D inputs ---- */
adinsel = 0x30; /* Analog input pin : AN0 to AN7 */
/* A/D sweep pin count : 8 pins */
/* A/D input group : Port P0 */
```

(6) A/D Control

28.2.6 A/D Control Register 1 (ADCON1)

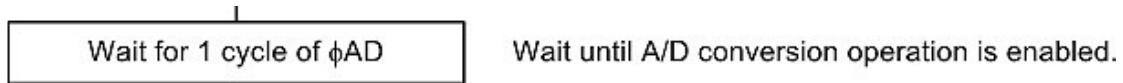
Address 00D7h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	ADDDAEL	ADDDAEN	ADSTBY	BITS	—	—	—	ADEX0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	ADEX0	Extended analog input pin select bit (1)	0: Extended analog input pin not selected 1: On-chip reference voltage selected (2)
b1	—	Reserved bits	Set to 0.
b2	—		
b3	—		
b4	BITS	8/10-bit mode select bit	0: 8-bit mode 1: 10-bit mode
b5	ADSTBY	A/D standby bit (3)	0: A/D operation stops (standby) 1: A/D operation enabled
b6	ADDDAEN	A/D open-circuit detection assist function enable bit (4)	0: Disabled 1: Enabled
b7	ADDDAEL	A/D open-circuit detection assist method select bit (4)	0: Discharge before conversion 1: Precharge before conversion

```
/* ---- Set A/D control register ---- */
adcon1 = 0x30;
    /* Not select extended analog input pin */
    /* 10-bit mode */
    /* Enable A/D operation */
    /* A/D open-circuit detection
       /* assist function disabled */
```





```
/* When the ADSTBY bit is changed */
/* from 0 (A/D operation stops) to 1 (A/D operation*)
/* enabled), wait for 1 AD cycle */

for (int i = 0; i < 10; i++){
```

(7) Summary

Initialization program is as follows.

```
void ad_init(void)
{
    unsigned char work; //Variable for PDo register

    /* === Read PDo register === */
    work = pdo & 0x0f;

    /* === Set Port === */
    PRC2 = 1; /* Protect off */
    pdo = work;
    //Po_3/AN4...Po_0/AN7 port direction = input
```

```

/* ---- Disable A/D conversion interrupt ---- */
adic = 0x00;

/* ---- Stop A/D conversion---- */
adcono = 0x00;

//Cut off on-chip reference voltage and analog input
prc3 = 1;           /* Protect off */
ocvrefan = 0;
prc3 = 0;           /* Protect on */

/* ---- Set A/D mode---- */
admod = 0x22;//Division select : fAD divided by 2
/* Clock source : F1 */
/* A/D operating mode : single sweep mode */
/* Start A/D conversion by software trigger */

//When the CKS2 bit is changed, wait for 3 fAD cycles
//or more before starting A/D conversion
for(int i = 0; i < 10; i++){
}

/* ---- Set A/D inputs----*/
adinsel = 0x30;    /* Analog input pin : AN0 to AN7 */
/* A/D sweep pin count : 8 pins */
/* A/D input group : Port Po */

```

```
/* ---- Set A/D control register-----*/
adcon1 = 0x30;
    /* Not select extended analog input pin */
    /* 10-bit mode */
    /* Enable A/D operation */
    /* A/D open-circuit detection
    /* assist function disabled */

/* When the ADSTBY bit is changed */
/* from 0 (A/D operation stops) to 1*/
/* (A/D operation enabled),*/
/* wait for 1 fAD cycle */
for (int i = 0; i < 10; i++){
}
}
```

[4.ad_read]

Let's create a program that reads the AD value without using interrupt.

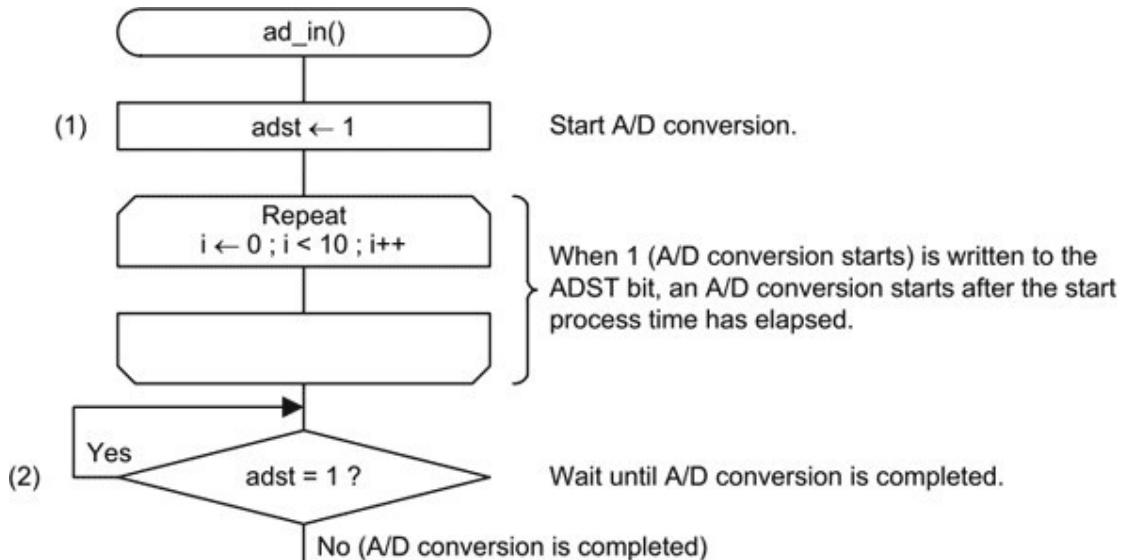
- (1) Create “Analog.cpp” file.

```
#include "LineTracer.h"  
unsigned short ad_data[4]; //A/D conversion result
```

```
void InitAdc()  
{  
    for ( int n = 0 ; n < 4 ; n++ ){  
        ad_data[n] = 0 ;  
    }  
}  
void ad_read()  
{  
}
```

Describe in “ad_read()” function.

(2) Describe in accordance with the flow chart of Renesas.



(3) A/D start

28.2.5 A/D Control Register 0 (ADCON0)

Address 00D6h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	ADST
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	ADST	A/D conversion start flag	0: Stop A/D conversion 1: Start A/D conversion

```

adst = 1;          /* Conversion start */
for (int i = 0; i < 10; i++);
    /* When 1 (A/D conversion starts) */
    /* is written to the ADST bit in the ADCON0 */
    /* register, an A/D conversion starts after */
    /* the start process time has elapsed */
  
```

```
/* Wait A/D conversion */
while(adst == 1); /* A/D conversion completed ? */
```

(3) Store the A/D value

(4) A/D read

28.2.2 A/D Register i (AD_i) (i = 0 to 7)

Address 00C1h to 00C0h (AD0), 00C3h to 00C2h (AD1), 00C5h to 00C4h (AD2),
 00C7h to 00C6h (AD3), 00C9h to 00C8h (AD4), 00CBh to 00CAh (AD5),
 00CDh to 00CCh (AD6), 00CFh to 00CEh (AD7)

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	—
After Reset	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8
Symbol	—	—	—	—	—	—	—	—
After Reset	0	0	0	0	0	0	X	X

```
ad_data[0] = ad7 & 0x3FF ; /*Save the A/D value*/
ad_data[1] = ad6 & 0x3FF ; /*Save the A/D value*/
ad_data[2] = ad5 & 0x3FF ; /*Save the A/D value*/
ad_data[3] = ad4 & 0x3FF ; /*Save the A/D value*/
```

From now on, you can use ad_data[] as sensor value in your application.

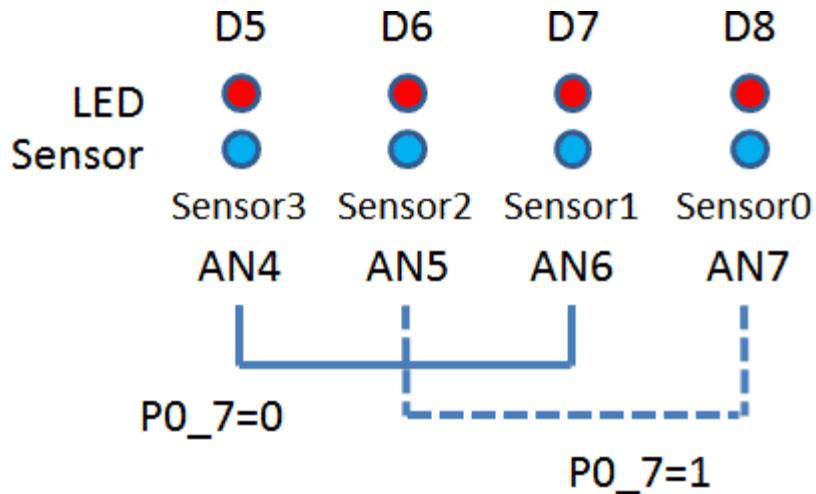
(5) In summary, the code will be as follows.

```
void ad_read()
{
    adst = 1;      /* Conversion start */
    for (int i = 0; i < 10; i++);
        /* When 1 (A/D conversion starts) */
        /* is written to the ADST bit in the ADCON0 */
        /* register, an A/D conversion starts after */
        /* the start process time has elapsed */

    /* Wait A/D conversion */
    while(adst == 1); //A/D conversion completed ?
    ad_data[0] = ad7 & 0x3FF ; //Save the A/D value
    ad_data[1] = ad6 & 0x3FF ;
    ad_data[2] = ad5 & 0x3FF ;
    ad_data[3] = ad4 & 0x3FF ;
}
```

[5.Check]

Let's try to turn on/off the LED by this analog value.



(1) Add code to “LineTracer.h” file.

```
extern unsigned short ad_data[4]; //A/D conversion result
```

(2) Add code to “Prototype.h” file.

```
void InitAdc(void);  
void ad_read(void); /* Set A/D fixed value */
```

(3) Add code to “main()” function as follows.

```
void main(void)  
{  
    Hwsetup();  
  
    InitAdc();  
    //Loop of main
```

```

while(1){      /* Main processing */
    ad_read();

    po_7 = ~po_7;

    if( po_7 == 0 ){
        //po_1 AN6 Sensor[1]
        po_5 = ( ad_data[1] > 0x01FF ) ? 1 : 0;
        //po_3 AN4 Sensor[3]
        po_6 = ( ad_data[3] > 0x01FF ) ? 1 : 0;
    }
    else {
        //po_0 AN7 Sensor[0]
        po_5 = ( ad_data[0] > 0x01FF ) ? 0 : 1;
        //po_2 AN5 SENSOR[2]
        po_6 = ( ad_data[2] > 0x01FF ) ? 0 : 1;
    }
}

```

When you have finished the above checks, in order to improve the prospects, the lighting of the LED is done in “Analog.cpp” file.

(4) Move code in “while loop” to “Analog.cpp” file as follows.

```
void Analog()
{
    ad_read();

    po_7 = ~po_7;

    if ( po_7 == 0 ){
        //po_1 AN6 Sensor[1]
        po_5 = ( ad_data[1] > 0x01FF ) ? 1 : 0;
        //po_3 AN4 Sensor[3]
        po_6 = ( ad_data[3] > 0x01FF ) ? 1 : 0;
    }
    else {
        //po_0 AN7 Sensor[0]
        po_5 = ( ad_data[0] > 0x01FF ) ? 0 : 1;
        //po_2 AN5 Sensor[2]
        po_6 = ( ad_data[2] > 0x01FF ) ? 0 : 1;
    }
}
```

Then “while loop” of “main” function will be,

```
//Loop of main
while(1){      /* Main processing */
    Analog();
}
```

Add code to “Prototype.h” file.

```
void Analog(void);
```

(5) Measured value

Since the sensor is 10 bits, the value is 0 to 1023 (0 ... ox3FF).
ox01FF is just the value of the middle.
VCC of MCU is 3.0V.

I shows a comparison between the measured value.

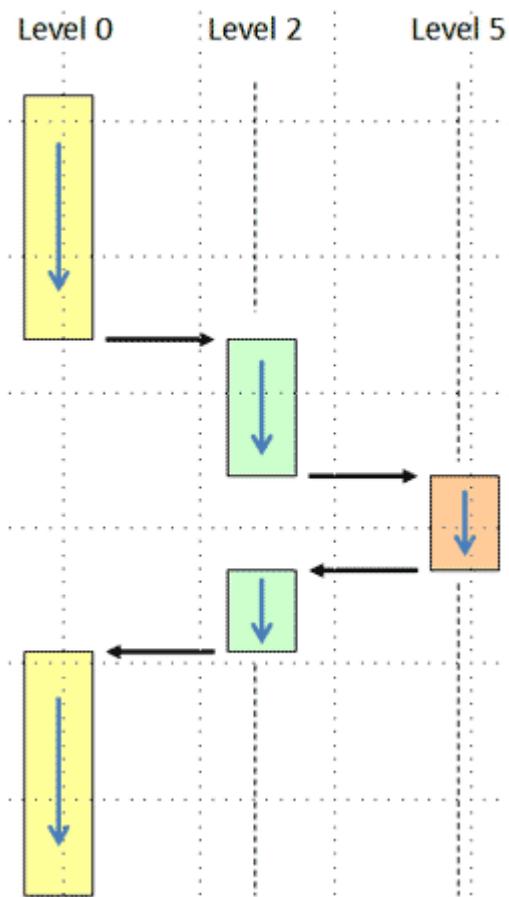
A/D	0	---	1023
Voltage	0v		3.0v
	Bright		Dark
measurement	0.4v		2.5v
A/D	136		853

[4] Interrupt

(1) The role of interrupt

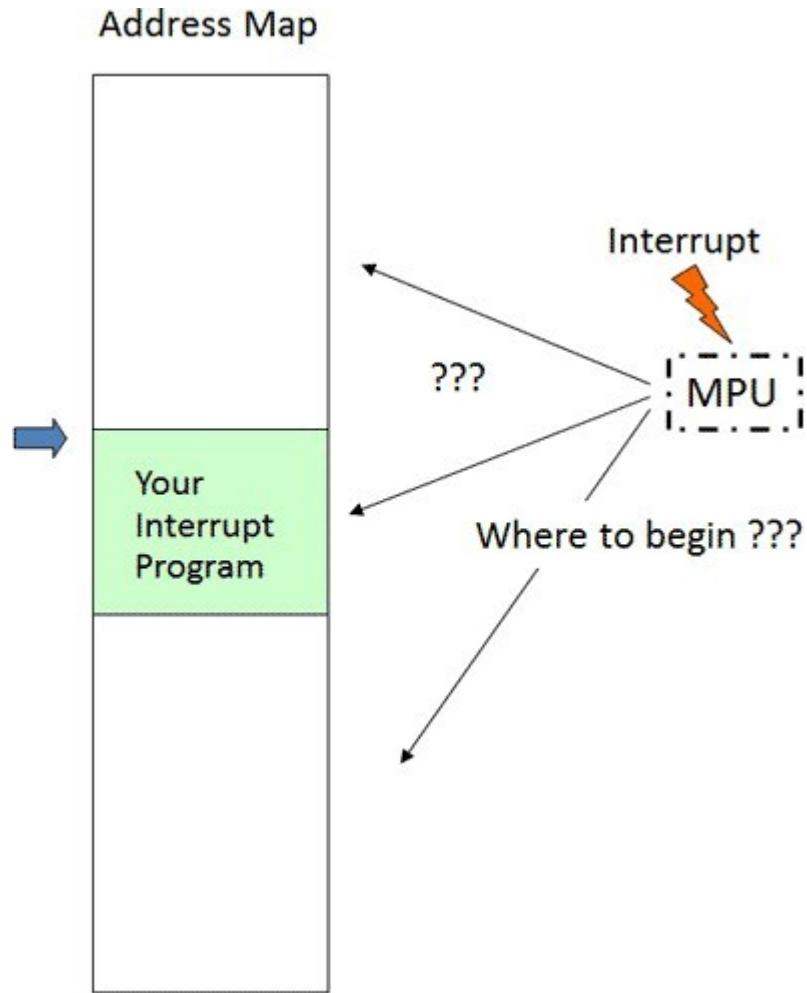
When level n process is running, if higher level process requests to execute, interrupt is used for that purpose.

- a) There is the level from 0 to 7
- b) Numerical big one has high priority



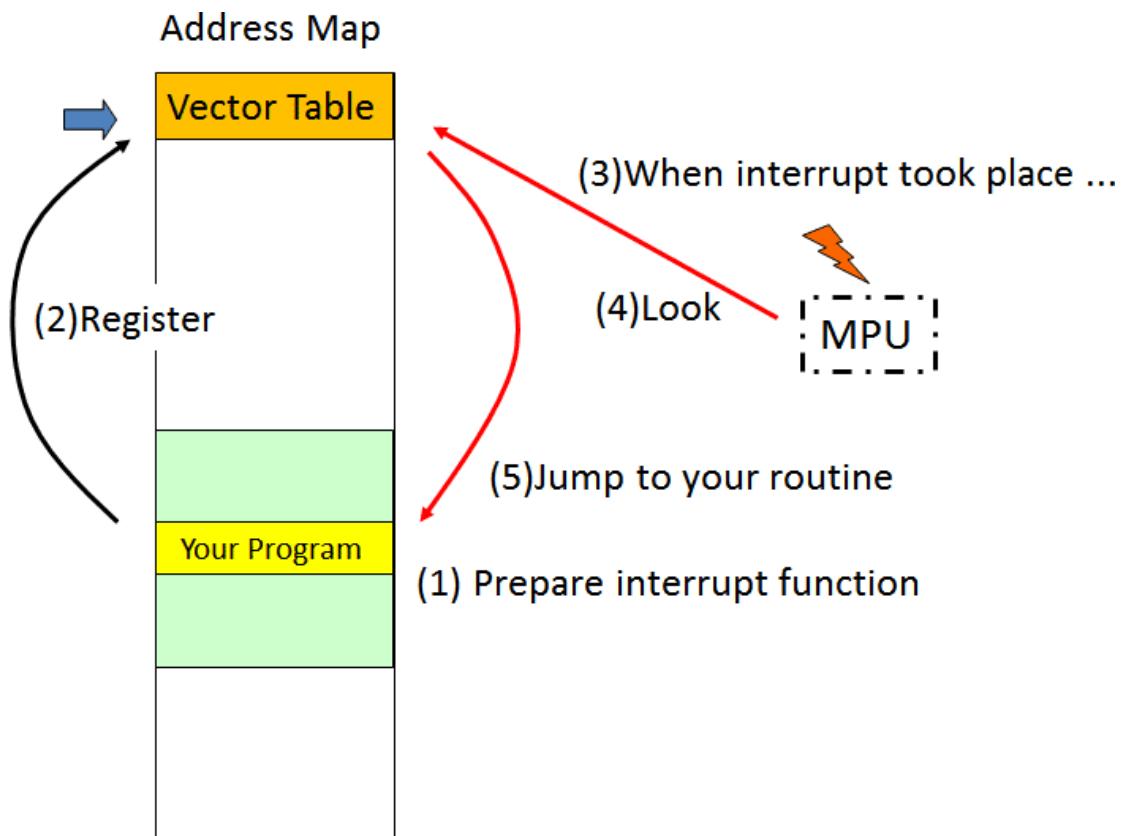
Give priority to the higher interrupt of the urgent degree including the data reception.

(2) When interrupt took place, how can MPU find the address of your interrupt program ?



It is necessary to prepare some kind of methods to inform MPU about start addresses.

(3) Vector table is the method of that purpose.



(4) How to write the interrupt processing function

The procedure is next two.

- Write the interrupt processing function.
- Registration to an interrupt vector table.

(5) When you write,

#pragma INTERRUPT (Interrupt function name)
then HEW compiler makes special code to be able to return to an original function,

- Save all registers before processing,
- Execute processing,
- Restore all registers after processing

(6) We take trigger of the AD conversion by a timer interrupt every 10ms.

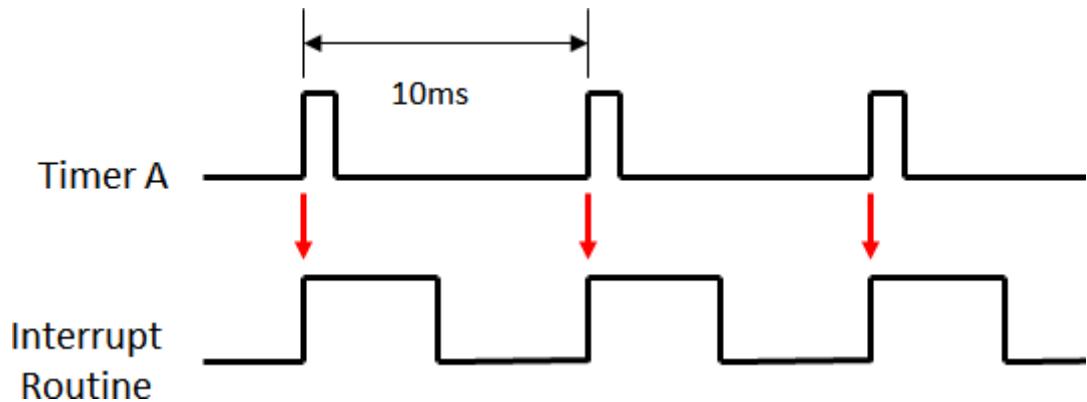
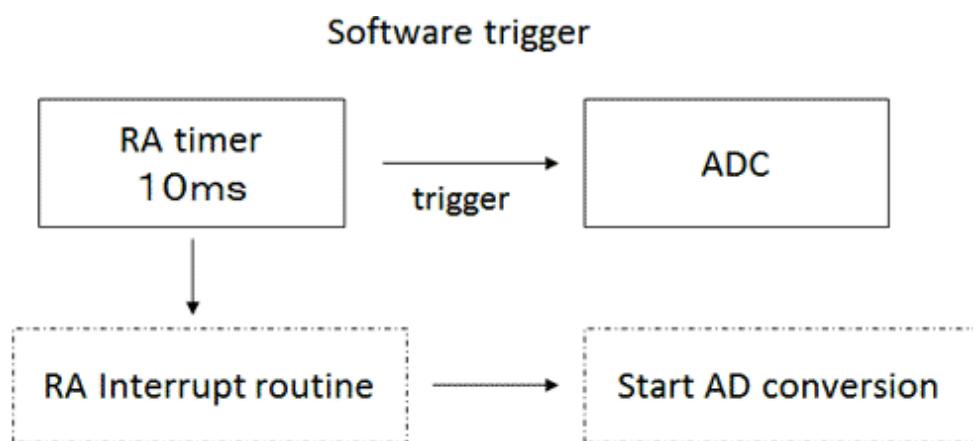


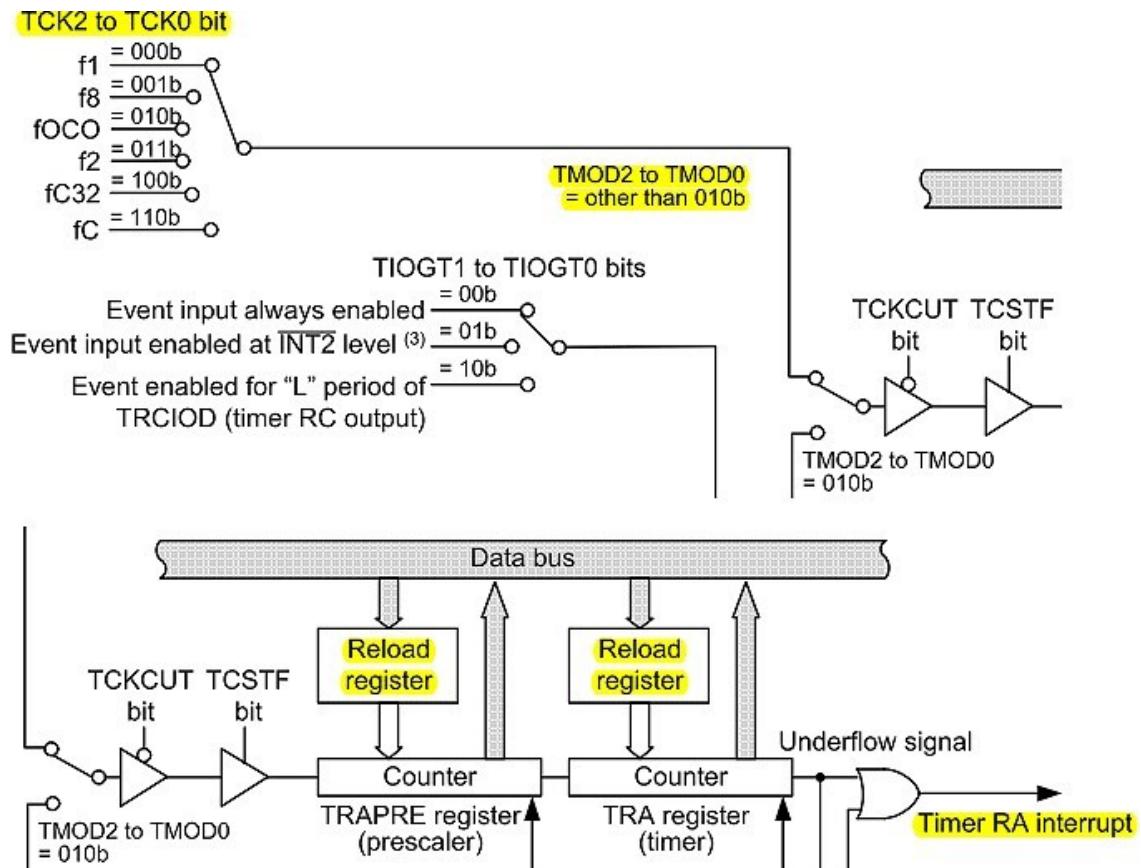
Table 16.1 Functional Comparison of Timers

Item	Timer RA	Timer RB	Timer RC
Configuration	8-bit timer with 8-bit prescaler (with reload register)	8-bit timer with 8-bit prescaler (with reload register)	16-bit timer (with input capture and output compare)
Count	Decrement	Decrement	Increment
Timer RD		Timer RE	
16-bit timer × 2 (with input capture and output compare)		4-bit counter 8-bit counter	
Increment/Decrement		Increment	

(7) We use the Timer RA as a trigger of the AD conversion.



(8) This is the Block diagram of the timer RA.



(9) Add a timer initialize function in “Hwsetup.cpp” file.

```

static void timer_ra_init(void);

void Hwsetup(void)
{
    asm("FCLR I");      /* Interrupt disabled */
    mcu_init();
    port_init();
    ad_init();
    timer_ra_init();
    asm("FSET I");      /* Interrupt enable */
}

```

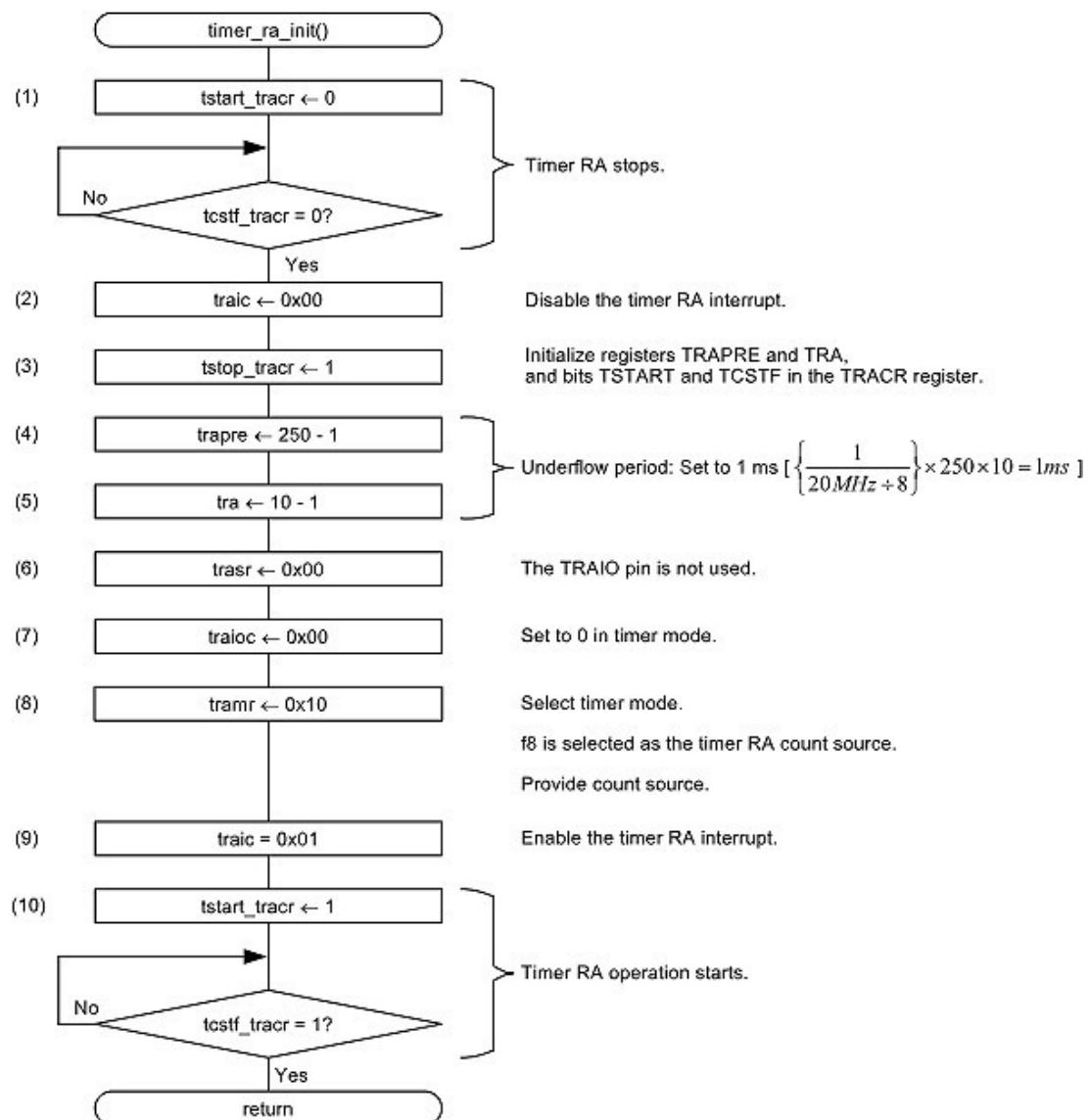
[1.timer_ra_init]

```
static void timer_ra_init ()  
{  
}
```

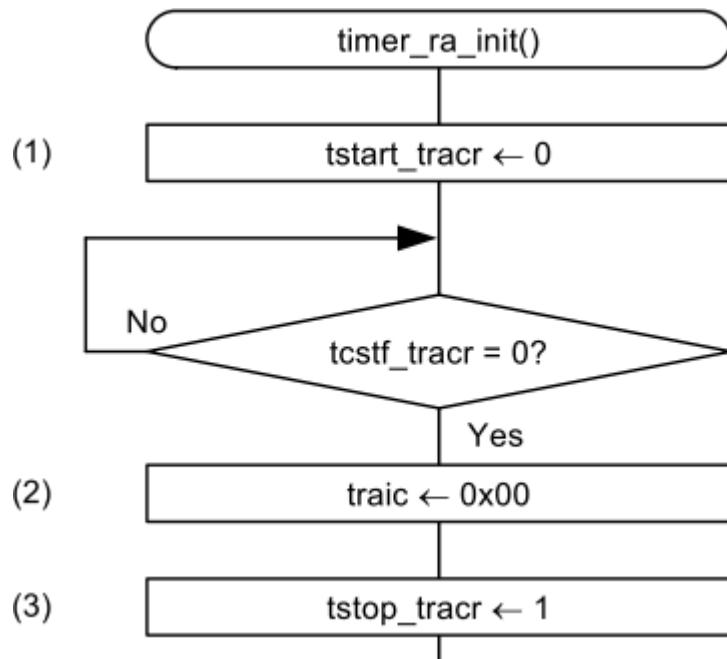
Write in “timer_ra_init()”.

Describe in accordance with a manual of “Renesas”.

- Flowchart



First, from the portion of the first half.



(1) Control

17.2.1 Timer RA Control Register (TRACR)

Address 0100h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	TUND _F	TEDGF	—	TSTOP	TCSTF	TSTART
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TSTART	Timer RA count start bit ⁽¹⁾	0: Count stops 1: Count starts
b1	TCSTF	Timer RA count status flag ⁽¹⁾	0: Count stops 1: During count

```

tstart_tracr = 0;      /* Stop Timer RA operation */
while(tcstf_tracr != 0);
    /* wait until tcstf_tracr becomes 0 */

```

(2) Interrupt

11.2.1 Interrupt Control Register

(TREIC, S2TIC, S2RIC, KUPIC, ADIC, SOTIC, SORIC, S1TIC, S1RIC, TRAIC, TRBIC, U2BCNIC, VCMP1IC, VCMP2IC)

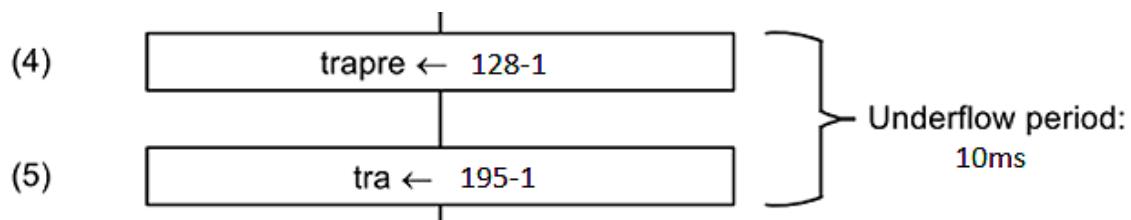
Address 004Ah (TREIC), 004Bh (S2TIC), 004Ch (S2RIC), 004Dh (KUPIC), 004Eh (ADIC),
0051h (SOTIC), 0052h (SORIC), 0053h (S1TIC), 0054h (S1RIC), 0056h (TRAIC),
0058h (TRBIC), 005Eh (U2BCNIC), 0072h (VCMP1IC), 0073h (VCMP2IC),

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	IR	ILVL2	ILVL1	ILVL0
After Reset	X	X	X	X	X	0	0	0

Bit	Symbol	Bit Name	Function
b0	ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0: Level 0 (interrupt disabled)
b1	ILVL1		0 0 1: Level 1
b2	ILVL2		0 1 0: Level 2
			0 1 1: Level 3
			1 0 0: Level 4
			1 0 1: Level 5
			1 1 0: Level 6
			1 1 1: Level 7

```
traic = 0x00;      /* Disable Timer RA Interrupt */
tstop_tracr = 1;
/* The TRAPRE and TRA registers are initialized */
```

(3) Prescaler



17.2.4 Timer RA Prescaler Register (TRAPRE)

Address 0103h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	—
After Reset	1	1	1	1	1	1	1	1

Bit	Mode	Function	Setting
b7 to b0	Timer mode	Counts an internal count source	00h to FFh

(4) Timer Register

17.2.5 Timer RA Register (TRA)

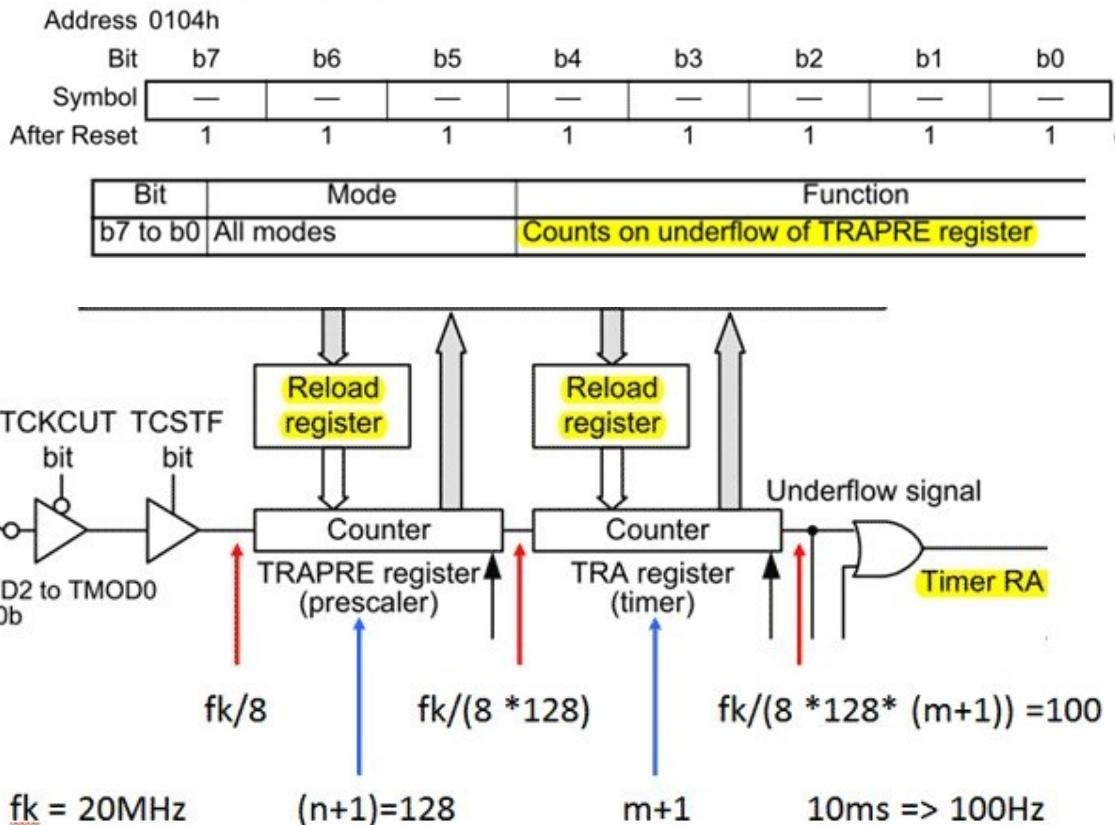


Table 17.2 Timer Mode Specifications

Item	Specification
Count sources	f1, f2, f8, fOCO, fC32, fC
Count operations	<ul style="list-style-type: none"> Decrement When the timer underflows, the contents of the reload register are reloaded and the count is continued.
Divide ratio	$1/(n+1)(m+1)$ n: Value set in TRAPRE register, m: Value set in TRA register

$$f_8 = \text{fk}/8$$

$$f_8 / ((N+1)(M+1)) = 100 \text{ Hz}$$

$$\text{if } (n+1) \quad \textcircled{O} \quad 128$$

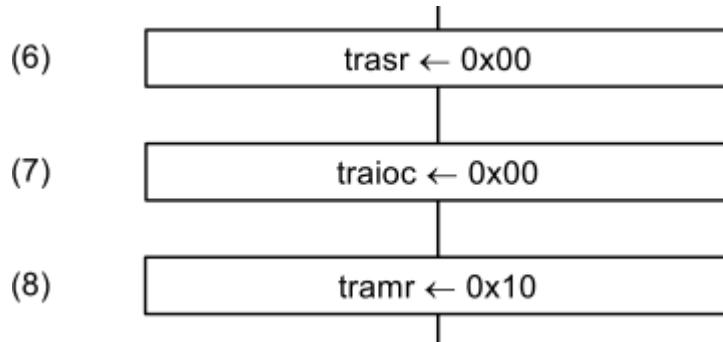
$$m+1 = f_8 / (128 * 100) = 195.3125$$

```

/* Period between underflows 10MS */
trapre = 128 - 1; //Set (128 - 1) in TRAPRE register
tra = 195 - 1; //Set (195 - 1) in TRA register

```

(5) TRAIO pin



17.2.6 Timer RA Pin Select Register (TRASR)

Address 0180h								
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	TRAIOSEL0	—	TRAIOSEL1	TRAIOSEL0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TRAIOSEL0	TRAIO pin select bit	b1 b0 0 0: TRAIO pin not used
b1	TRAIOSEL1		0 1: P1_7 assigned 1 0: P1_5 assigned 1 1: Do not set.

17.3.1 Timer RA I/O Control Register (TRAIOC) in Timer Mode

Address 0101h								
Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	TIOGT1	TIOGT0	TIPF1	TIPF0	—	TOENA	TOPCR	TEDGSEL
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TEDGSEL	TRAIO polarity switch bit	Set to 0 in timer mode.
b1	TOPCR	TRAIO output control bit	
b2	TOENA	TRAIO output enable bit	
b3	—	Reserved bit	Set to 0.
b4	TIPF0	TRAIO input filter select bit	Set to 0 in timer mode.
b5	TIPF1		
b6	TIOGT0	TRAIO event input control bit	
b7	TIOGT1		

```

trasr = 0x00; /* TRAIO pin not used */
traioc = 0x00; /* Set to "0" in timer mode */

```

(6) Timer Mode

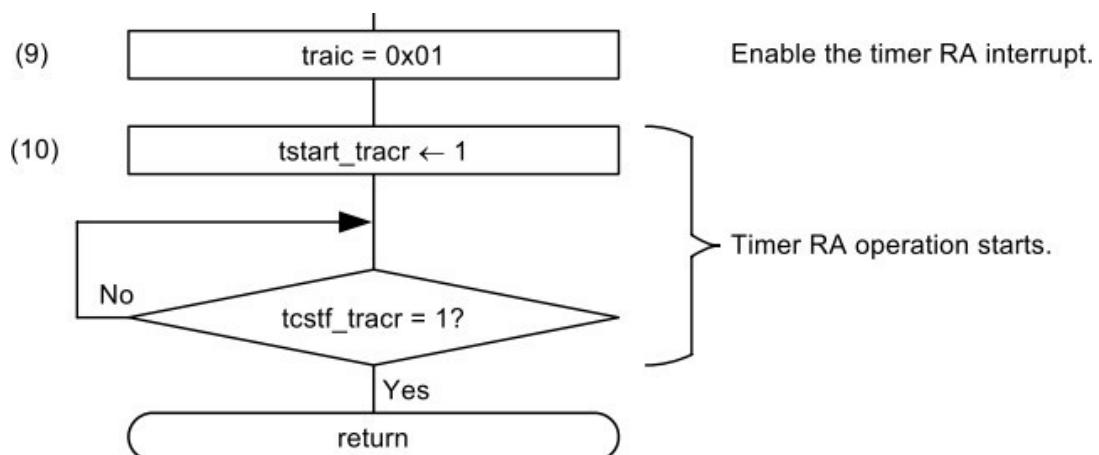
17.2.3 Timer RA Mode Register (TRAMR)

Address 0102h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	TCKCUT	TCK2	TCK1	TCK0	—	TMOD2	TMOD1	TMOD0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TMOD0	Timer RA operating mode select bit	b2 b1 b0 0 0 0: Timer mode
b1	TMOD1		0 0 1: Pulse output mode 0 1 0: Event counter mode 0 1 1: Pulse width measurement mode 1 0 0: Pulse period measurement mode 1 0 1: Do not set. 1 1 0: Do not set. 1 1 1: Do not set.
b2	TMOD2		
b3	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b4	TCK0	Timer RA count source select bit	b6 b5 b4 0 0 0: f1 0 0 1: f8
b5	TCK1		0 1 0: fOCO 0 1 1: f2 1 0 0: fC32 1 0 1: Do not set. 1 1 0: fC 1 1 1: Do not set.
b6	TCK2		
b7	TCKCUT	Timer RA count source cutoff bit	0: Provides count source 1: Cuts off count source

```
tramr = 0x10; /* Set to "ooo" in timer mode */
                /* Select "f8" in Count Source */
                /* Provides count source */
```



(7) Interrupt

11.2.1 Interrupt Control Register

(TREIC, S2TIC, S2RIC, KUPIC, ADIC, SOTIC, SORIC, S1TIC, S1RIC, TRAIC, TRBIC, U2BCNIC, VCMP1IC, VCMP2IC)

Address 004Ah (TREIC), 004Bh (S2TIC), 004Ch (S2RIC), 004Dh (KUPIC), 004Eh (ADIC),
0051h (SOTIC), 0052h (SORIC), 0053h (S1TIC), 0054h (S1RIC), 0056h (TRAIC),
0058h (TRBIC), 005Eh (U2BCNIC), 0072h (VCMP1IC), 0073h (VCMP2IC),

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	IR	ILVL2	ILVL1	ILVL0
After Reset	X	X	X	X	X	0	0	0

Bit	Symbol	Bit Name	Function
b0	ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0: Level 0 (interrupt disabled)
b1	ILVL1		0 0 1: Level 1
b2	ILVL2		0 1 0: Level 2
			0 1 1: Level 3
			1 0 0: Level 4
			1 0 1: Level 5
			1 1 0: Level 6
			1 1 1: Level 7

17.2.1 Timer RA Control Register (TRACR)

Address 0100h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	TUND _F	TEDGF	—	TSTOP	TCSTF	TSTART
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TSTART	Timer RA count start bit (1)	0: Count stops 1: Count starts
b1	TCSTF	Timer RA count status flag (1)	0: Count stops 1: During count

```
traic = 0x01; /* enable Timer RA Interrupt */
```

```
tstart_tracr = 1; /* Start Timer RA operation */
```

```
while(tcstf_tracr != 1);
```

```
/* wait until tcstf_tracr becomes 1 */
```

(8) In summary, the code will be as follows.

```
static void timer_ra_init()
{
    tstart_tracr = 0;      /* Stop Timer RA operation */
    while(tcstf_tracr != 0);
        /* wait until tcstf_tracr becomes 0 */

    traic = 0x00;          /* Disable Timer RA Interrupt */
    tstop_tracr = 1;
    /* The TRAPRE and TRA registers are initialized */
    /* Period between underflows 10MS */
    trapre = 128 - 1;
        /* Set (128 - 1) in TRAPRE register */
    tra = 195 - 1;          /* Set (195 - 1) in TRA register */
    trasr = 0x00;           /* TRAO pin not used */
    traioc = 0x00;          /* Set to "0" in timer mode */
    tramr = 0x10;           /* Set to "ooo" in timer mode */
        /* Select "f8" in Count Source */
        /* Provides count source */
    traic = 0x05;           //enable Timer RA Interrupt level 5
```

```
tstart_tracr = 1; /* Start Timer RA operation */
while(tcstf_tracr != 1);
    /* wait until tcstf_tracr becomes 1 */
}
```

(9) As we can use the interrupt, add the code to the "ad_init()".

```
void ad_init(void)
{
    :
    for (i = 0; i < 10; i++){
    }
    /* ---- Enable A/D conversion interrupt-----*/
    adic = 0x05;      /* level 5*/
}
```

(10) We add the part of the interrupt from now on.

You can see the following description in “Section definition” file which is generated by Renesas Project Generator.

```
; When you use "#pragma interrupt" with "vect=",
; you need not define interrupt vector.
;
; When you use "#pragma interrupt" without "vect=",
; you must define all interrupt vectors like the following example.
; You define dummy_int for interrupt vector not used.
;
;     .lword    dummy_int    ; vector  0
;     .lword    dummy_int    ; vector  1
;     .lword    dummy_int    ; vector  2
;     :
;     lword    dummy_int    ; vector 63
```

Table 11.2 Relocatable Vector Tables

Interrupt Source	Vector Addresses (1) Address (L) to Address (H)	Software Interrupt Number	Interrupt Control Register
A/D conversion	+56 to +59 (0038h to 003Bh)	14	ADIC
Synchronous serial communication unit / I ² C bus interface (2)	+60 to +63 (003Ch to 003Fh)	15	SSUIC/IICIC
(Reserved)		16	-
UART0 transmit	+68 to +71 (0044h to 0047h)	17	S0TIC
UART0 receive	+72 to +75 (0048h to 004Bh)	18	S0RIC
UART1 transmit	+76 to +79 (004Ch to 004Fh)	19	S1TIC
UART1 receive	+80 to +83 (0050h to 0053h)	20	S1RIC
INT2	+84 to +87 (0054h to 0057h)	21	INT2IC
Timer RA	+88 to +91 (0058h to 005Bh)	22	TRAIC

(11) Add code to “Analog.cpp” file as follows.

```
#pragma interruptintTRAIC(vect=22)
void intTRAIC(void)
{
    adst = 1;      //ADC Start
}

#pragma interrupt/B intADIC(vect=14)
void intADIC(void)
{
    /* Write A/D conversion result */
    ad_data[0] = ad7 & ox03FF ; /* AN7 */
    ad_data[1] = ad6 & ox03FF ; /* AN6 */
    ad_data[2] = ad5 & ox03FF ; /* AN5 */
    ad_data[3] = ad4 & ox03FF ; /* AN4 */
}
```

Comment out “ad_read ()” from “Analog()” function.

```
void Analog(void)
{
    //ad_read();
```

(12) Try to run.

If the LED are turned on/off by the value of sensor, then we can find the interrupt is working.

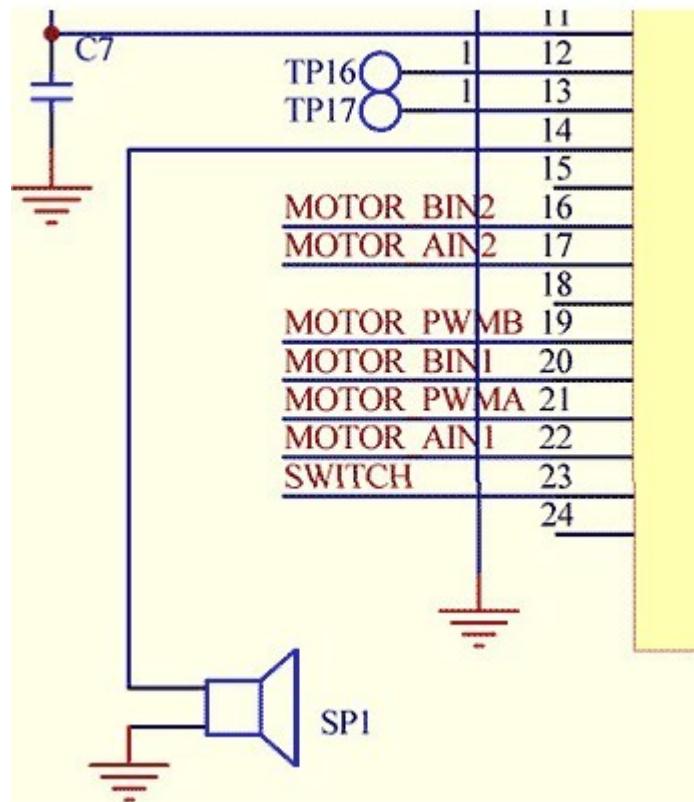
{Question 4}

Why do we carry out the A / D conversion by interrupt?

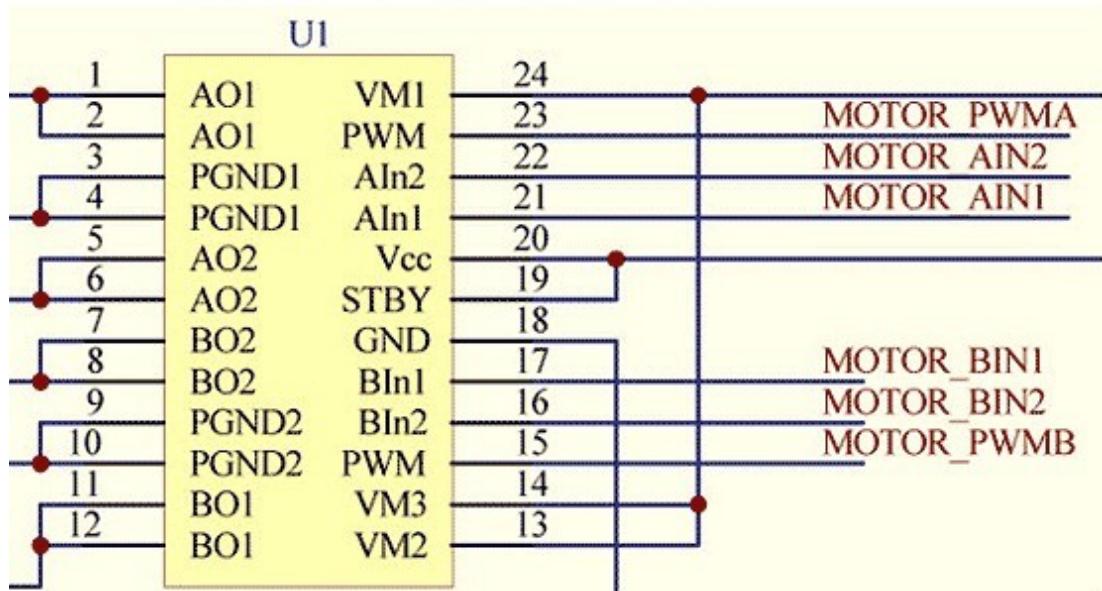
[5] Motor drive

Well, let's make an important BIOS part left.
It is about motor drive

MPU (R8C/34C)



Motor Drive IC (TB6612FNG)

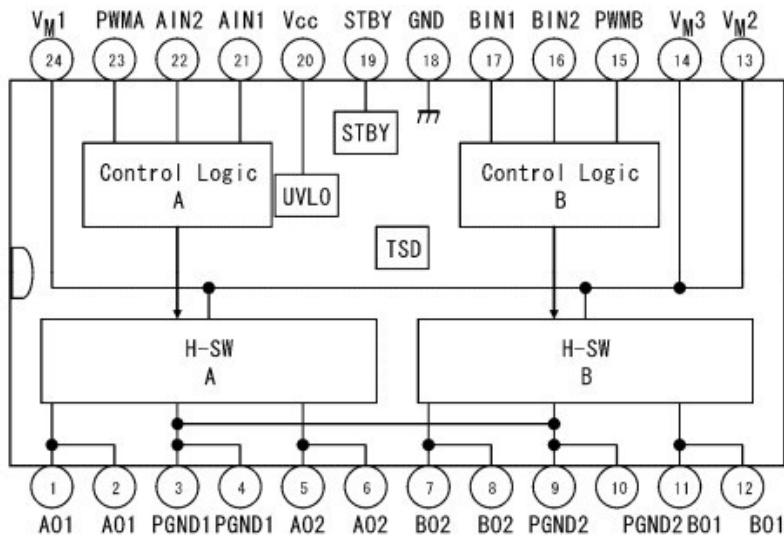


[1.Motor Drive IC]

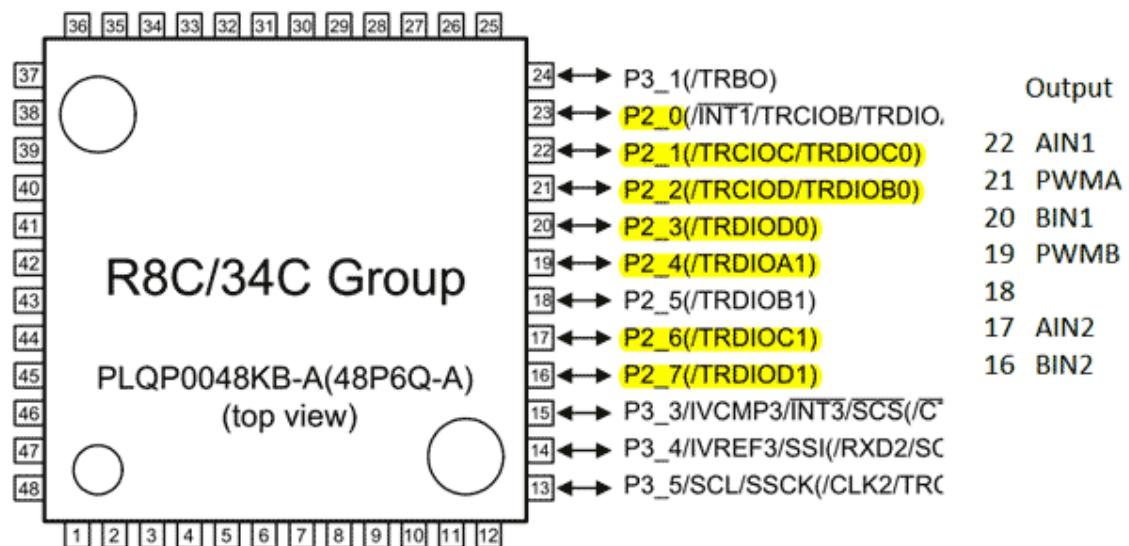
TOSHIBA

TB6612FNG

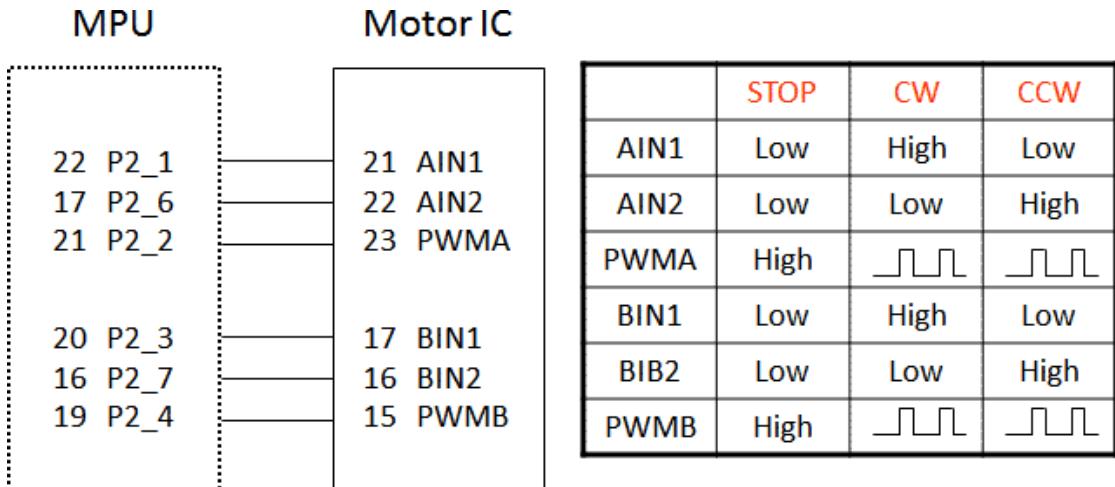
Block Diagram



Driver IC for Dual DC motor



Since the output signal to the motor IC is connected to the terminals of the timer D, we use the timer D.



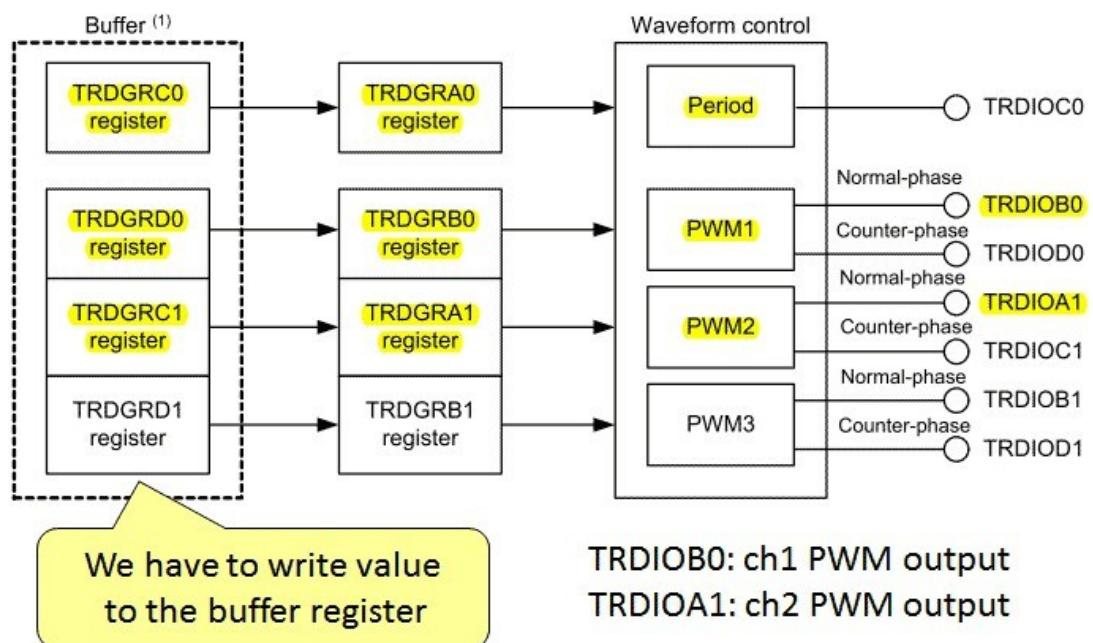
When you change the value to “IN1” and “IN2”, the state is switched to “STOP”, “CW”, “CCW”.

If you change the Duty ratio of the PWM pin, rotation speed will change.

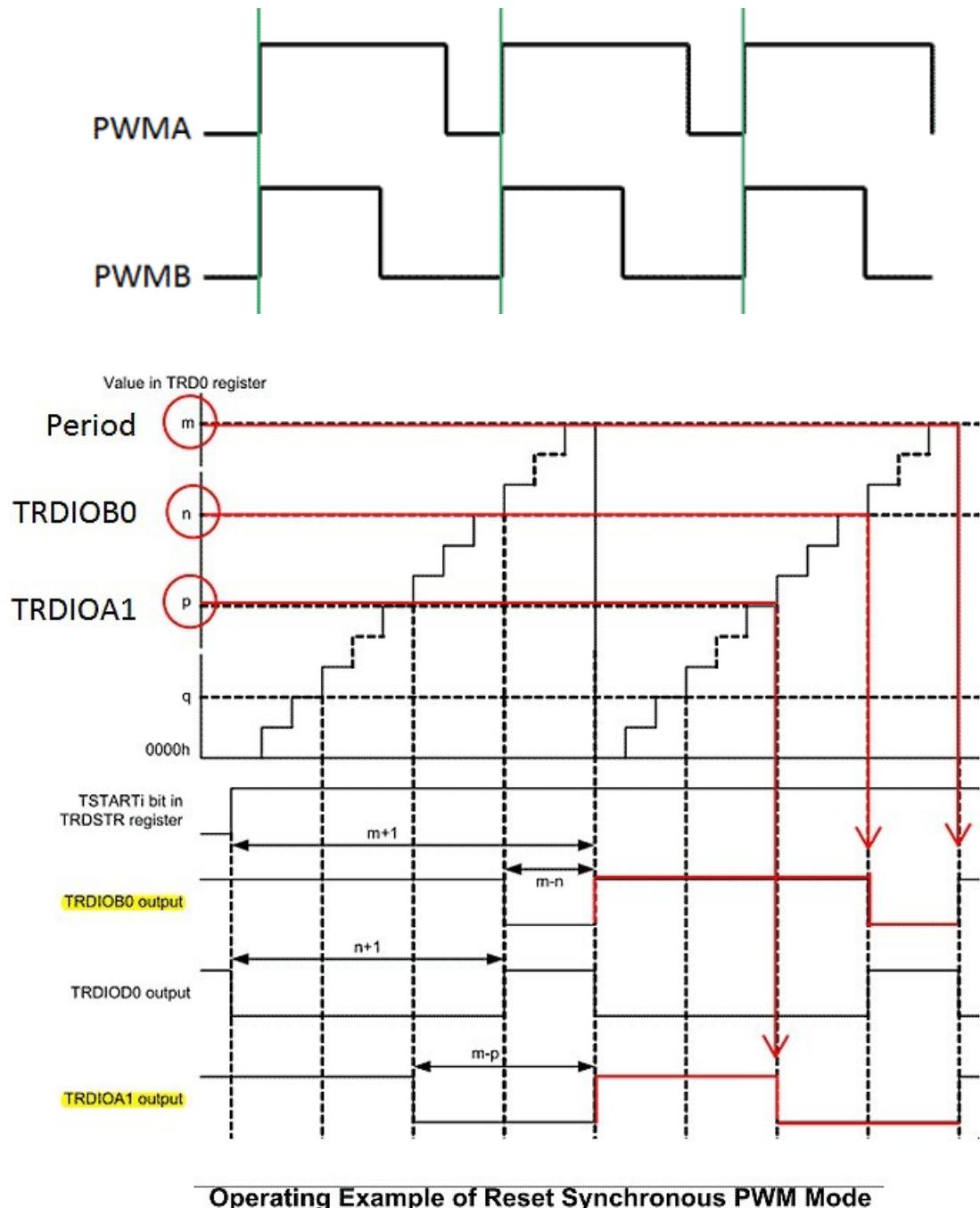
(1) We use Timer D for making PWM wave.

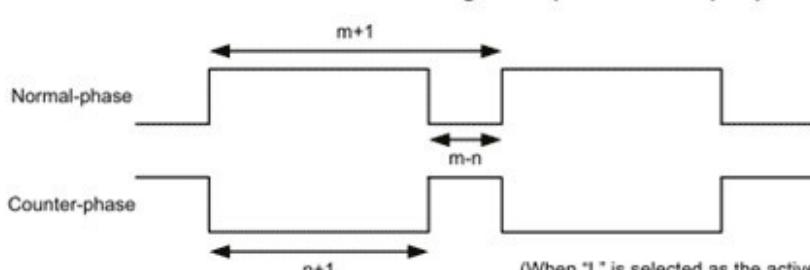
Timer D has two 16-bit timer (timer RDo and timer RD1).
We use timer RDo.

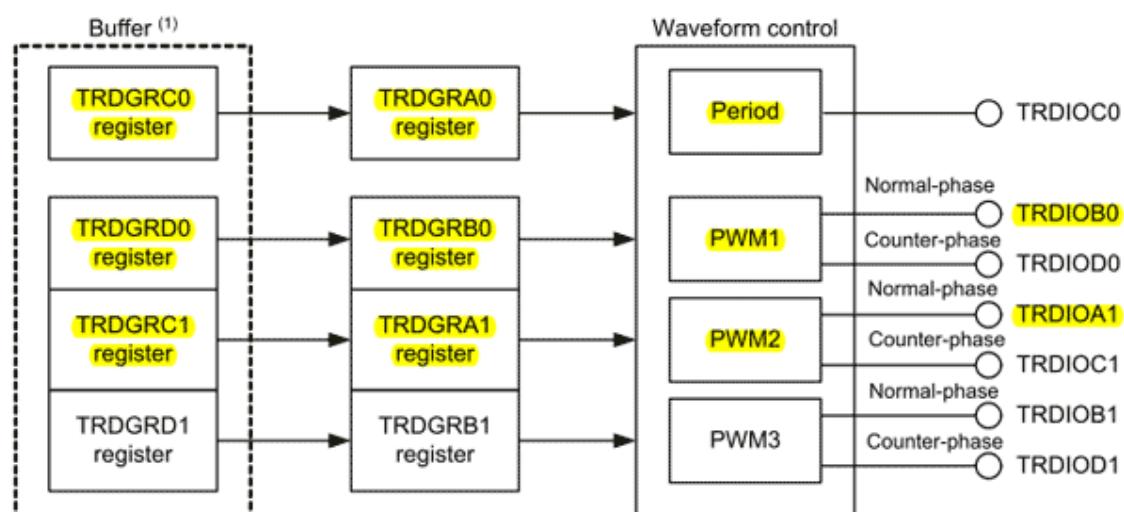
20.6 Reset Synchronous PWM Mode



(2) We use the two channels of PWM in “Reset Synchronous Mode”.



PWM waveform	PWM period $: 1/fk \times (m+1)$ Active level width of normal-phase : $1/fk \times (m-n)$ Active level width of counter-phase: $1/fk \times (n+1)$ fk: Frequency of count source m: Value set in the TRDGRA0 register n: Value set in the TRDGRB0 register (PWM1 output), Value set in the TRDGRA1 register (PWM2 output), Value set in the TRDGRB1 register (PWM3 output)
	 (When "L" is selected as the active level)



(3) We have to write value to the buffer register.

Then the value is transferred at next correct timing.
 TRDGRC0, TRDGRD0, TRDGRC1 are the buffer register.

PWM Period $p = 1/fk * (m+1)$
 if $p = 16\text{ms}$, $fk = 20\text{MHz} / 8$
 $m = p * fk - 1$
 $m = 16/1000 * (20000000 / 8) - 1 = 40000 - 1$
 This value is within “unsigned int” range.
 So we can set TRDGRCo = m

(4) Add code in “Hwsetup.cpp” file.

```

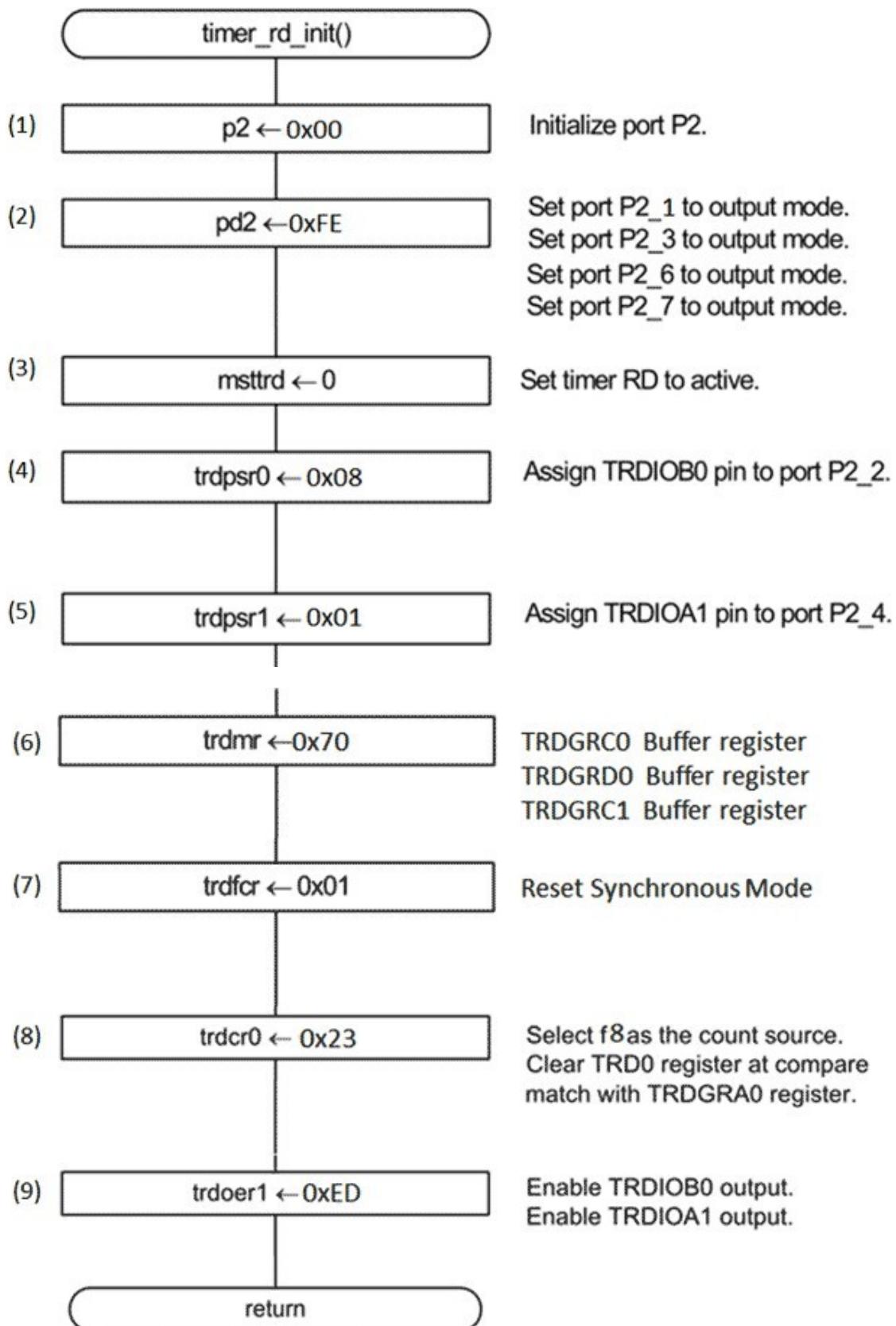
static void timer_rd_init(void);

void Hwsetup(void)
{
    asm("FCLR I");      /* Interrupt disabled */
    mcu_init();
    port_init();
    ad_init();
    timer_ra_init();
    timer_rd_init();
    asm("FSET I");      /* Interrupt enable */
}

static void timer_rd_init ()
{
}
  
```

Describe code in “timer_rd_init()”.

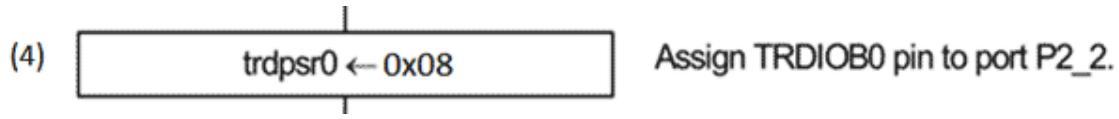
[2.TIMER_rd_init]



Item “(1) to (3)” are already written in the following function.

```
static void mcu_init()
{
    :
    mstcr = 0x28; /* SSU,I2C and Timer RC*/
                    /*to Standby mode*/
}
static void port_init()
{
    :
    // SWITCH Button & Motor Output
    P2 = 0 ;      //P2_0=Switch Input, P2_1...P2_7=Motor Output
    pu04 = 1 ;    //Pullup for Switch Input
    PD2 = 0xFE ; //P2_0 Input, P2_1...P2_7 Output
}
```

(4) Assign TRDIOBo pin to port P2_2



20.6.14 Timer RD Pin Select Register 0 (TRDPSR0)

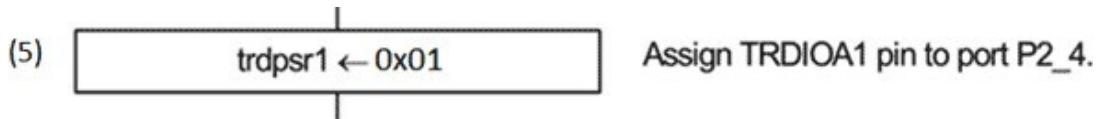
Address 0184h

Bit	b7	b6	b5	b4	b3	b2
Symbol	—	TRDIOD0SEL0	TRDIOC0SEL1	TRDIOC0SEL0	TRDIOB0SEL1	TRDIOB0SEL0
After Reset	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b2	TRDIOB0SEL0	TRDIOB0 pin select bit	b3 b2 0 0: TRDIOB0 pin not used 0 1: Do not set. 1 0: P2_2 assigned 1 1: Do not set.
b3	TRDIOB0SEL1		
b4	TRDIOC0SEL0	TRDIOC0 pin select bit	b5 b4 0 0: TRDIOC0 pin not used 0 1: Do not set. 1 0: P2_1 assigned 1 1: Do not set.
b5	TRDIOC0SEL1		
b6	TRDIOD0SEL0	TRDIOD0 pin select bit	0: TRDIOD0 pin not used 1: P2_3 assigned

trdpsr0 = 0x08; // TRDIOBo = P2_2

(5) Assign TRDIOA1 pin to port P2_4



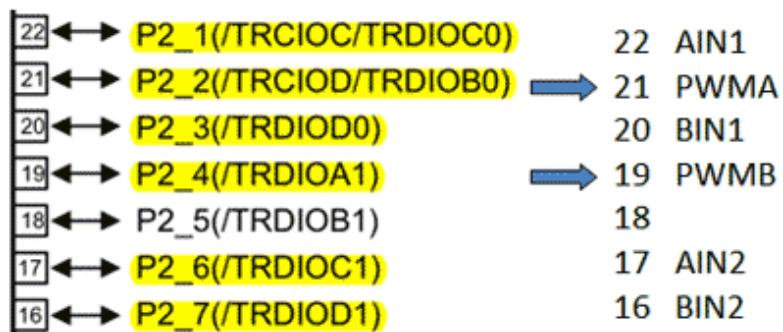
20.6.15 Timer RD Pin Select Register 1 (TRDPSR1)

Address 0185h

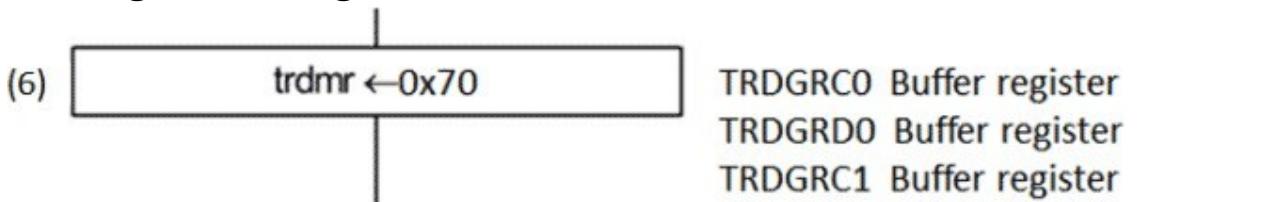
Bit	b7	b6	b5	b4	b3	b2
Symbol	—	TRDIOD1SEL0	—	TRDIOC1SEL0	—	TRDIOB1SEL0
After Reset	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TRDIOA1SEL0	TRDIOA1 pin select bit	0: TRDIOA1 pin not used 1: P2_4 assigned
b1	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b2	TRDIOB1SEL0	TRDIOB1 pin select bit	0: TRDIOB1 pin not used 1: P2_5 assigned
b3	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b4	TRDIOC1SEL0	TRDIOC1 pin select bit	0: TRDIOC1 pin not used 1: P2_6 assigned
b5	—	Reserved bit	Set to 0.
b6	TRDIOD1SEL0	TRDIOD1 pin select bit	0: TRDIOD1 pin not used 1: P2_7 assigned

trdpsr1 = 0X01; // TRDIOA1 = P2_4



(6) Assign Buffer register.



20.6.5 Timer RD Mode Register (TRDMR) in Reset Synchronous PWM

Address 0138h

Symbol	b7	b6	b5	b4	b3	b2	b1	b0
BFD1	0	0	0	0	1	1	1	0
After Reset								

Bit	Symbol	Bit Name	Function
b4	BFC0	TRDGRC0 register function select bit	0: General register 1: Buffer register of TRDGRA0 register
b5	BFD0	TRDGRD0 register function select bit	0: General register 1: Buffer register of TRDGRB0 register
b6	BFC1	TRDGRC1 register function select bit	0: General register 1: Buffer register of TRDGRA1 register

trdmr = 0x70; // Buffer register

(7) Set to “Reset Synchronous Mode”.



20.6.6 Timer RD Function Control Register (TRDFCR) in Reset Sync PWM Mode

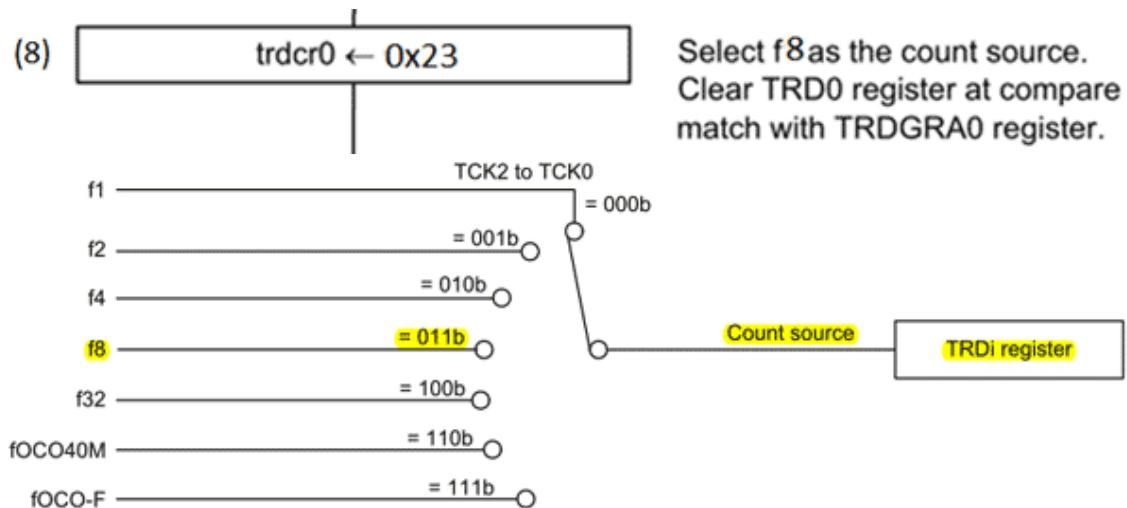
Address 013Ah

Symbol	b7	b6	b5	b4	b3	b2	b1	b0
PWM3	1	0	0	0	0	0	0	0
After Reset								

Bit	Symbol	Bit Name	Function
b0	CMD0	Combination mode select bit (1)	Set to 01b (reset synchronous PWM mode) in reset synchronous PWM mode.
b1	CMD1		
b2	OLS0	Normal-phase output level select bit (in reset synchronous PWM mode or complementary PWM mode)	0: Initial output "H", Active level "L" 1: Initial output "L", Active level "H"
b3	OLS1	Counter-phase output level select bit (in reset synchronous PWM mode or complementary PWM mode)	

```
trdfcr = 0x01; //Reset Synchronous Mode
```

- (8) Select count source to f8. (f8= fk/8 divide an original clock)
TRDo cleared at compare match



20.6.9 Timer RD Control Register 0 (TRDCR0) in Reset Synchronous PWM

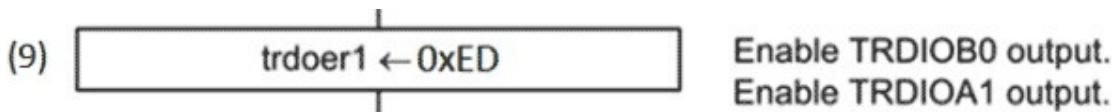
Address 0140h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	CCLR2	CCLR1	CCLR0	CKEG1	CKEG0	TCK2	TCK1	TCK0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TCK0	Count source select bit	b2 b1 b0 0 0 0: f1 0 0 1: f2 0 1 0: f4 0 1 1: f8 1 0 0: f32 1 0 1: TRDCLK input (1) or fC2 (2) 1 1 0: fOCO40M 1 1 1: fOCO-F (4)
b2	TCK2		
b3	CKEG0	External clock edge select bit (3)	b4 b3 0 0: Count at the rising edge 0 1: Count at the falling edge 1 0: Count at both edges 1 1: Do not set.
b4	CKEG1		
b5	CCLR0	TRD0 counter clear select bit	Set to 001b (TRD0 register cleared at compare match with TRDGRA0 register) in reset synchronous PWM mode.
b6	CCLR1		
b7	CCLR2		

```
trdcro = 0x23; //TRDo cleared at compare match
```

(9) Enable TRDIOBo,TRDIOA1



20.6.7 Timer RD Output Master Enable Register 1 (TRDOER1) in Synchronous PWM Mode

Address 013Bh

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	ED1	EC1	EB1	EA1	ED0	EC0	EB0	EA0
After Reset	1	1	1	1	1	1	1	1

Bit	Symbol	Bit Name	Function
b0	EA0	TRDIOA0 output disable bit	Set this bit to 1 (the TRDIOA0 pin is used as a programmable I/O port) in reset synchronous PWM mode.
b1	EB0	TRDIOB0 output disable bit	0: Enable output 1: Disable output (The TRDIOB0 pin is used as a programmable I/O port.)
b4	EA1	TRDIOA1 output disable bit	0: Enable output 1: Disable output (The TRDIOA1 pin is used as a programmable I/O port.)

`trdoer1 = 0xED;` //Enable TRDIOBo,TRDIOA1

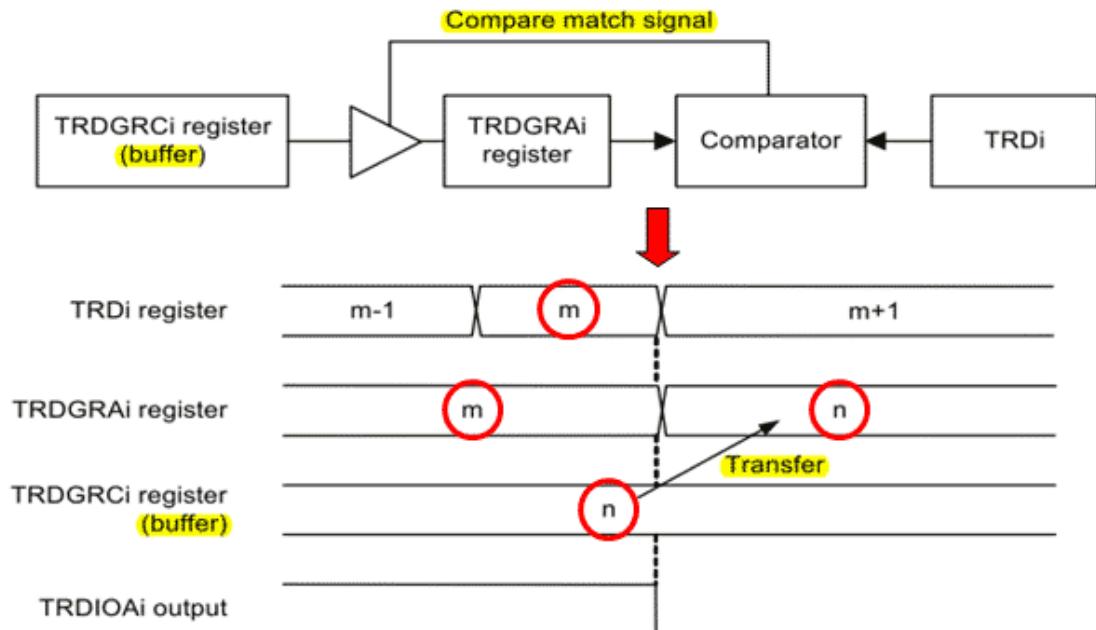


Table 20.12 TRDGRji Register Functions in Reset Synchronous PWM Mode

Register	Setting	Register Function	PWM Output Pin
TRDGRA0	–	General register. Set the PWM period.	(Output inverted every PWM period and TRDI0C0 pin)
TRDGRB0	–	General register. Set the changing point of PWM1 output.	TRDIOB0 TRDIOD0
TRDGRA1	–	General register. Set the changing point of PWM2 output.	TRDIOA1 TRDI0C1
TRDGRB1	–	General register. Set the changing point of PWM3 output.	TRDIOB1 TRDIOD1
TRDGRC0	BFC0 = 1	Buffer register. Set the next PWM period. (Refer to 20.2.2 Buffer Operation.)	(Output inverted every PWM period and TRDI0C0 pin)
TRDGRD0	BFD0 = 1	Buffer register. Set the changing point of the next PWM1 output. (Refer to 20.2.2 Buffer Operation.)	TRDIOB0 TRDIOD0
TRDGRC1	BFC1 = 1	Buffer register. Set the changing point of the next PWM2 output. (Refer to 20.2.2 Buffer Operation.)	TRDIOA1 TRDI0C1

(10) In summary, the code will be as follows.

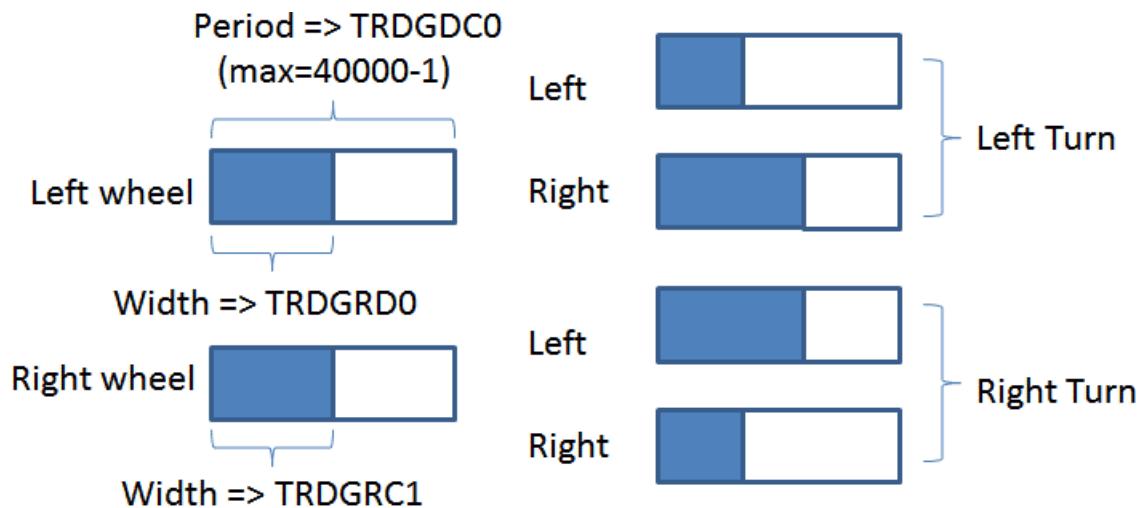
```
static void timer_rd_init ()
{
    trdpsr0 = 0x08; // TRDIOB0 = P2_2
    trdpsr1 = 0x01; // TRDIOA1 = P2_4
    trdmr = 0x70; // Buffer register
    trdfcr = 0x01; //Reset Synchronous Mode
    trdcro = 0x23; //TRD0 cleared at compare match
    trdoer1 = 0xED; //Enable TRDIOB0,TRDIOA1
}
```

With this, BIOS part is almost finished.

[3.motor control]

Let's make the motor control part.

Now that we have set the “PWM Period = 16ms”, as calculated before, “m + 1” is 40000.



As the user interface, it is not preferable to use the counter value itself.

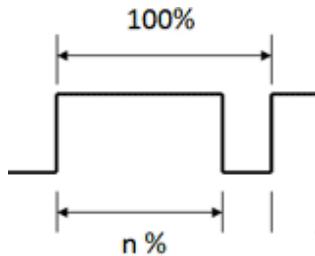
It is much easier way to use a ratio corresponding to maximum velocity.

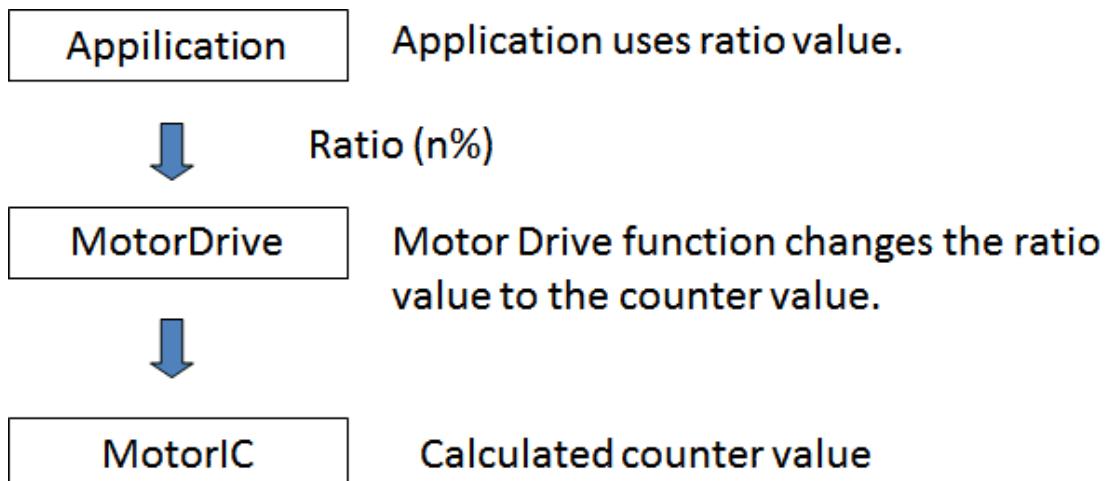
Function MotorDrive(int left, intright)

left, right is % for maxVal

TRDGRD0 = ((long) maxVal * left) / 100;

TRDGRC1= ((long) maxVal * right) / 100;





[4.Motor.cpp]

Create a file named "Motor.cpp".

(1) Add code in "LineTracer.h" file.

```
enum { STOP, FORWARD, BACKWARD, TOWARD, TOLEFT };
```

(2) Add code in "Prototype.h" file.

```
void InitMotor( int rate );
void MotorDrive( int dir, int left, int right);
```

(3) Prepare functions in "Motor.cpp" file as follows.

```
#include "LineTracer.h"
#define PERIOD      40000
static long maxValue ;
```

```

void InitMotor( int rate)
{
    maxValue = ((long)PERIOD * rate ) /100 ;
    trdgrao = trdgrco = PERIOD - 1;
    trdgrbo = trdgrdo = maxValue / 2; // PWM 1
    trdgra1 = trdgRC1 = maxValue / 2; // PWM 2
    trdstr = ox05; // start TRDO count
}

void MotorDrive( int dir, int left, int right)
{
    if ( dir == STOP )      //Stop
    {
        P2_1 = 0 ;
        P2_6 = 0 ;
        P2_3 = 0 ;
        P2_7 = 0 ;
    }
    else
    {
        trdgrdo = (maxValue * left) / 100 ; // PWM1
        trDGRC1 = (maxValue * right) / 100; // PWM2
        switch ( dir ){

```

```
case FORWARD :  
    P2_1 = 0 ;      // Left wheel  
    P2_6 = 1 ;  
    P2_3 = 0 ;      // Right wheel  
    P2_7 = 1 ;  
    break ;  
case BACKWARD :  
    P2_1 = 1 ;      // Left wheel  
    P2_6 = 0 ;  
    P2_3 = 1 ;      // Right wheel  
    P2_7 = 0 ;  
    break ;  
case TORIGHT :  
    P2_1 = 0 ;      // Left Forward  
    P2_6 = 1 ;  
    P2_3 = 1 ;      // Right Backward  
    P2_7 = 0 ;  
    break ;  
case TOLEFT :  
    P2_1 = 1 ;      // Left Backward  
    P2_6 = 0 ;  
    P2_3 = 0 ;      // Right Forward  
    P2_7 = 1 ;  
    break ;  
}  
}  
}
```

(4) Let try whether motor works.

Modify “main()” function as follows.

```
void main(void)
{
    Hwsetup();

    InitAdc();
    InitMotor(50);      //50%

    while(1)          /* Main processing */
    {
        //if SWITCH == ON
        if( P2_O == 0 )
        {
            MotorDrive(FORWARD, 60, 60);      //%
        }
        else
        {
            MotorDrive(STOP, 0, 0);
        }
    }
}
```

Make sure the motor is rotated by pressing the switch button.

{Question 5}



As we give pulsed voltage waveform, but...

Why motor rotate so smoothly?

There is a great physical phenomena, but do you know what it is?

[6] Digital Input

Consider the actual operation. Because there is only one button,

a) When you press the start button, line tracer starts moving.

b) While in motion and when you press the start button again, it stops action. Let this way.

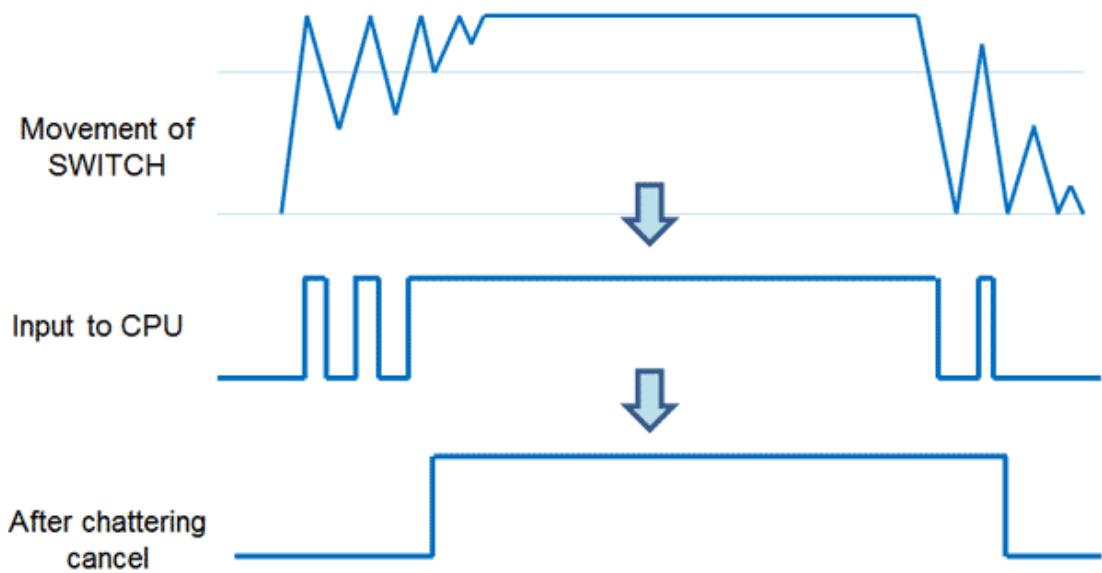
```
bool runFlag = false ;  
if ( runFlag == false ){  
    if ( Start_PB == ON ){  
        Motor Start ;  
        runFlag = true ;  
    }  
}  
else {  
    if ( Start_PB == ON ){  
        Motor Stop ;  
        runFlag = false ;  
    }  
}
```

Execute in this way several times.

You'll notice that sometimes it works well, but other it does not.

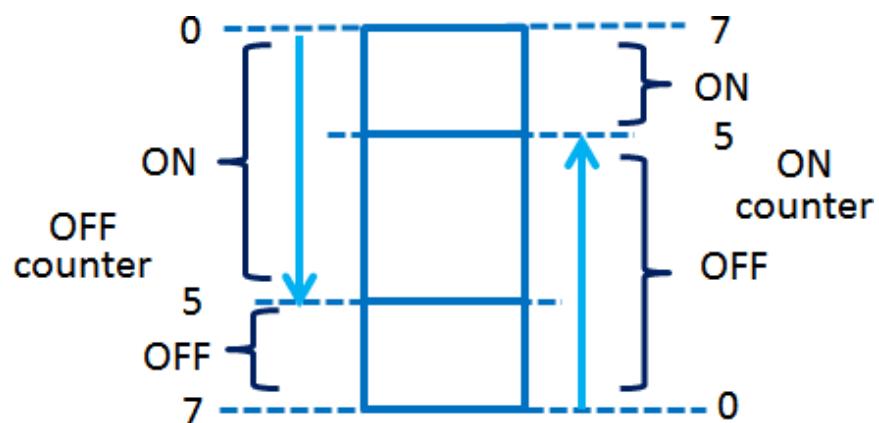
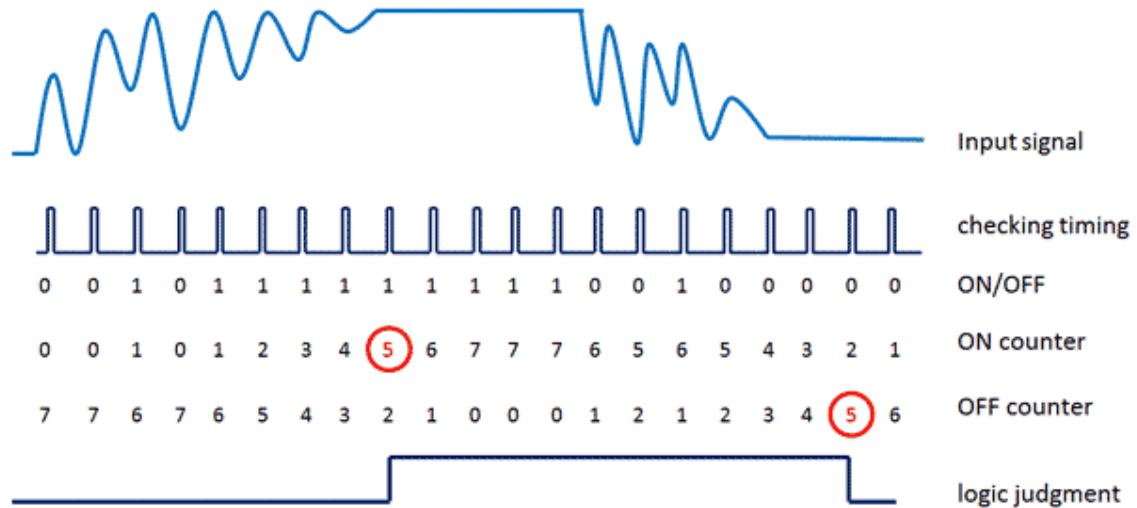
Then you will know that you must change a way of capturing the start button.

First, we have to remove the chattering from an apparatus which has the mechanical contact.



[1.Chattering canceller]

You can consider many ways, but we use the following method.



[2.INPUT.cpp]

(1) Create a file named “Input.cpp”.
Describe in this file.

```
#include "LineTracer.h"
struct inpCtr {
    unsigned short on ;
    unsigned short off ;
};
static struct inpCtr inpTbl[INNUM] ;
const bool logic[INNUM] = {
    NEG,      // o START_PB
};
static const unsigned short LIMIT = 50 ;
static const unsigned short MANY = 45 ;
static int step ;

void InitInput()
{
    struct inpCtr *sp = inpTbl ;
    for ( int n = INNUM ; n-- > 0 ; ) {
        sp->on = 0 ;
        sp->off = 0 ;
        sp++ ;
    }
    step = 0 ;
}
```

```
void Input()
{
    int n ;
    struct inpCtr *sp = inpTbl ;
    for ( int m = 0 ; m < INNUM ; m++ ){
        switch ( m ){
            case START_PB :
                n = P2_O ;
                break ;
            default :
                n = 0 ;
        }
        if( n ){
            if( sp->on < LIMIT )
                sp->on++ ;
            if( sp->off > 0 )
                sp->off-- ;
        }
        else {
            if( sp->off < LIMIT )
                sp->off++ ;
            if( sp->on > 0 )
                sp->on-- ;
        }
        sp++ ;
    }
}
```

[3.Method]

```
int ChkIn( int n )
{
    struct inpCtr *sp ;
    const bool *sq ;
    sp = &inpTbl[n] ;
    sq = &logic[n] ;
    if( sp->on > MANY )
        return ( *sq == POS ) ? ON : OFF ;
    else if( sp->off > MANY )
        return ( *sq == POS ) ? OFF : ON ;
    else
        return -1 ;
}
```

(1) If you use the same switch for ON and OFF, the following method will be helpful.

Function IsStartPB

prepare step as a global data;

```
Set flag false;
switch( step ){
    case waiting ON step :
        if ( Start button pushed ? ){
            Set flag true;
            go to Start button OFF wait step;
        }
        break;
    case waiting OFF step :
        if ( Start button released ? ){
            go to Start button ON wait step;
        }
        break;
}
return flag;
```

```
bool IsStartPB()
{
    bool flag = false ;
    switch ( step )
    {
        case 0 :
            if ( ChkIn(START_PB) == ON )
            {
                flag = true ;
                step++ ;
            }
            break;
        case 1 :
            if ( ChkIn(START_PB) == OFF )
                step = 0 ;
            break;
        default :
            step = 0;
    }
    return flag ;
}
```

(2) Add code in “LineTracer.h” file.

```
enum {
    START_PB,
    INNUM
};
enum { OFF,ON,UP };
enum { NEG, POS };
```

(3) Add code in “Prototype.h” file.

```
void InitInput(void);
void Input(void);
bool IsStartPB(void);
```

(4) Modify “main” file as follows.

Instead of “(P2_O == 0)”, you can use “(IsStartPB() == true)”.

```
void main(void)
{
    Hwsetup();
    InitAdc();
    InitMotor(50);      //50%
    InitInput();
    while(1)           /* Main processing */
    {
        Analog();
        Input();
        //if SWITCH == ON
        if (IsStartPB() == true)
        {
            MotorDrive (FORWARD, 60, 60);      //%
        }
        else
        {
            MotorDrive (STOP, 0, 0);
        }
    }
}
```

(5) Surely, you can start and stop it by the start button in this way.

You can extend this ways.

For example,

```
//Digital Input Port
```

```
enum {  
    Start_PB,  
    Stop_PB,  
    Reset_PB,  
    UrgentStop_PB,  
    SW3,  
    SW2-4,  
    INNUM  
};
```

{Exercise 1}

Consider the other way to remove the chattering of equipment with the point of contact.

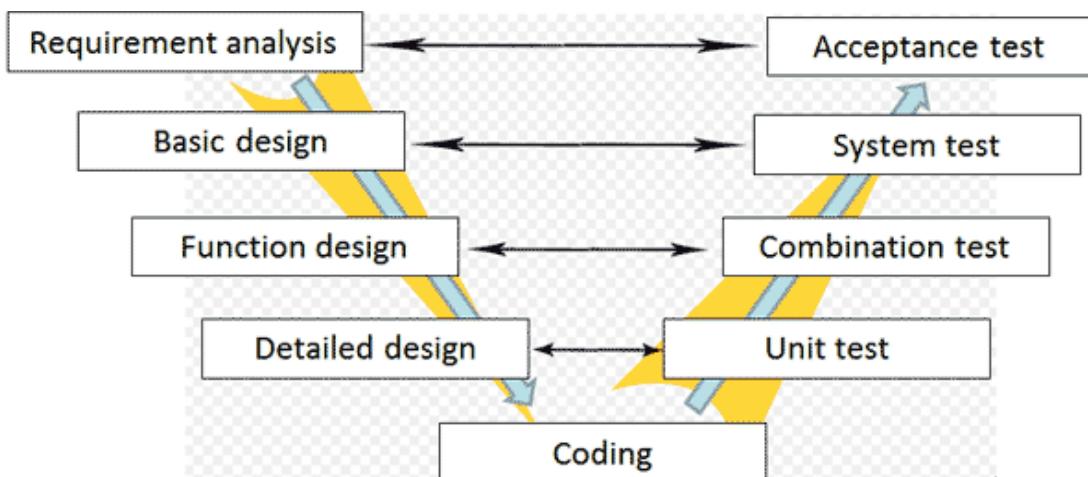
[7] Application

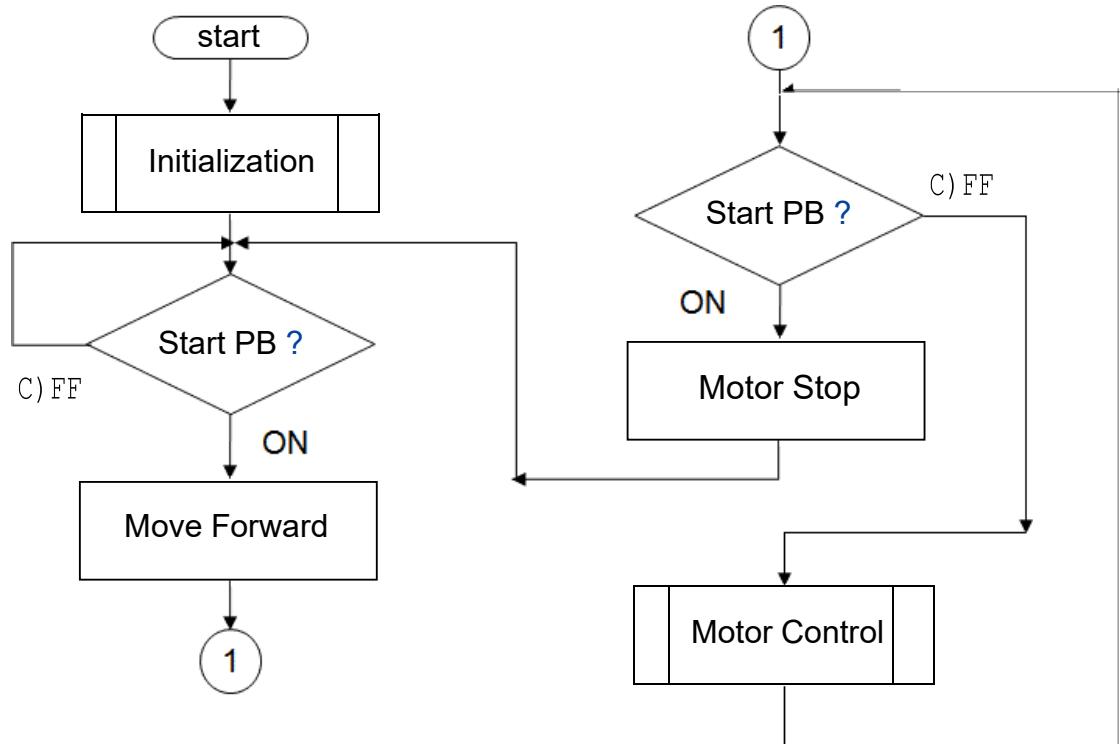
Now that we have made up BIOS section, from here we can advance to the application.

It is the place that your creativity can be exhibited.

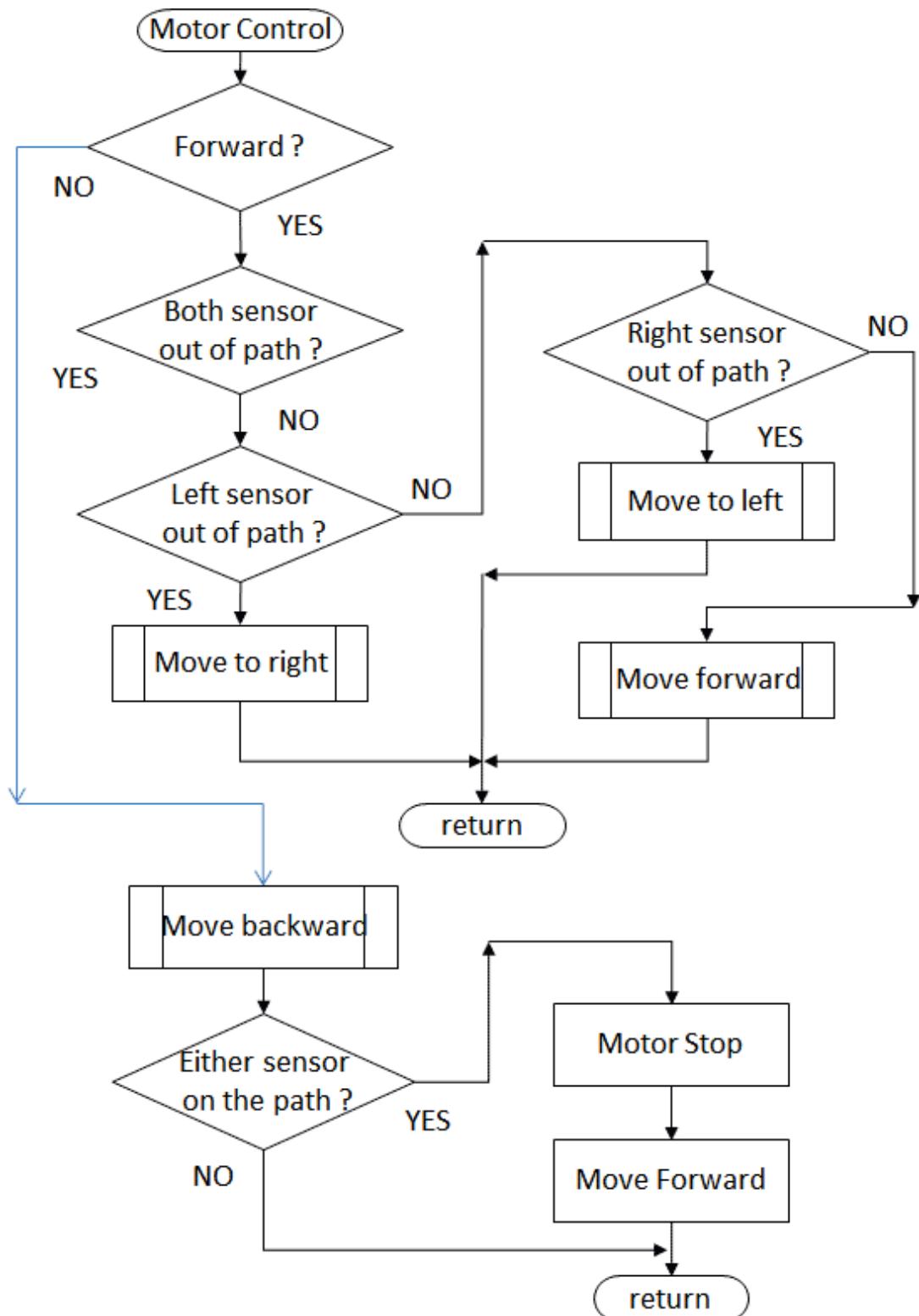
Before starting to write code, we'd better prepare a flow chart or some kind of document. After examining it many times, we can write it down to source code.

The following is typical software development process.
Before coding, there is a long design process.





[1. General Flow chart]



(1) At first, you'd better to describe it with your mother language.
For example...

```
Set run Flag false;
while ( 1 ){ //main loop
    if ( run Flag is false ){
        if( Start button pushed ){
            Motor start;
            Set run Flag true;
        }
    }
    else {
        if ( Start button pushed ){
            Motor stop;
            Set run Flag false;
        }
        else {
            Call Motor Control function;
        }
    }
}
```

```
Motor Control function()
{
    if ( Both sensor out of path ){
        Motor stop;
        Move back;
        if ( Either sensor on the path ){
            Motor stop;
            Move forward;
        }
    }
    else {
        Obtain the difference of the sensor of right and left;
        if ( the difference is small ){
            Normal speed;
            Move forward straight;
        }
        else {
            Slow down;
            Change direction according to
            the difference;
        }
    }
}
```

(2) To work correctly, some programming technique is necessary.
Let's divide the state of the control within the motor control function.

Motor Control()

```
{  
    switch ( state ){  
        case FORWARD :  
            forward process ;  
            break;  
        case BACKWARD :  
            backwardprocess;  
            break;  
    }  
}
```

(3) On FORWARD

```
if ( both sensor out of path ?){  
    confirm several times;  
    if ( so ){  
        Motor stop;  
        change state to BACKWARD;  
    }  
}
```

```
else {  
    obtain the difference of the sensor of right and left;  
    calculate output rate;  
    if ( the difference is small ){  
        normal speed;  
        Move forward straight;  
    }  
    else {  
        slow down;  
        change direction according to the difference;  
    }  
}  
}
```

(4) On BACKWARD

```
obtain the difference of the sensor of right and left;  
calculate output rate;  
slow speed;  
Move Backward with changing direction;  
if ( either sensor on the path ){  
    confirm several times;  
    if ( so ){  
        Motor stop;  
        change state to FORWARD;  
    }  
}
```

When you want to stop the motor, just you can call the Motor function with “STOP” argument.

```
MotorDrive( STOP, 0, 0 );
```

When you want to start the motor, you can use the Motor function with “FORWARD” argument and speed rate.

```
MotorDrive( FORWARD, 50, 50 ); //Left 50%, Right 50%
```

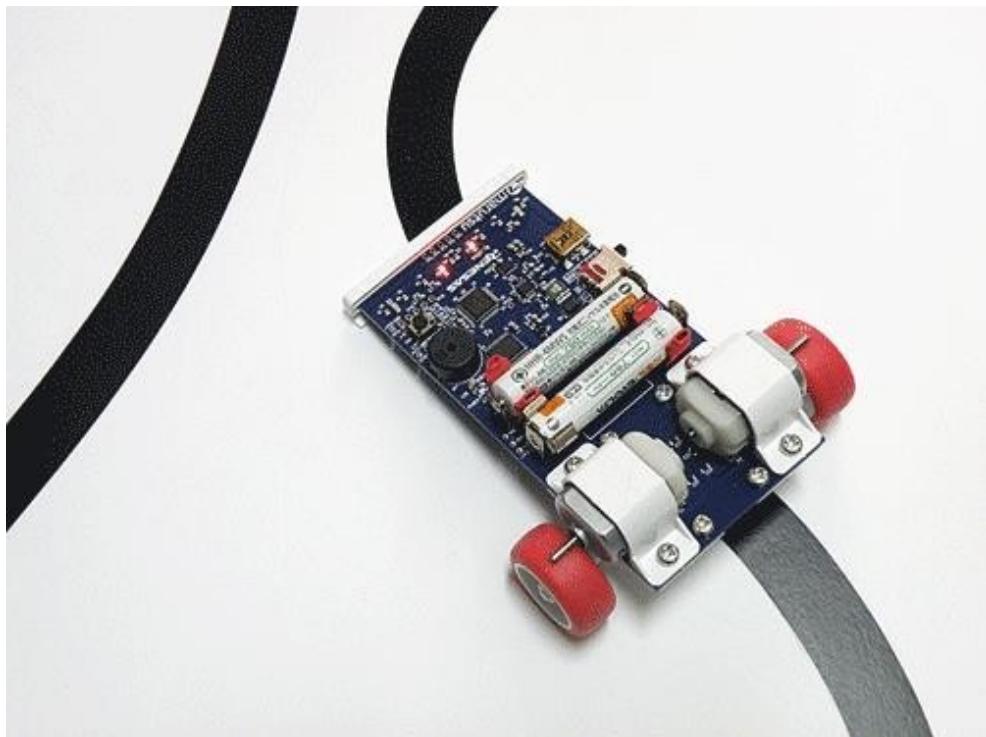
When you want to change direction, you can use the Motor function with “FORWARD” argument and different speed rate.

```
MotorDrive( FORWARD, 40, 50 ); //Left 40%, Right 50%
```

When you want to move back, you can use the Motor function with “BACKWARD” argument and any speed rate.

```
MotorDrive( BACKWARD, 40, 30 ); //Left 40%, Right 30%
```

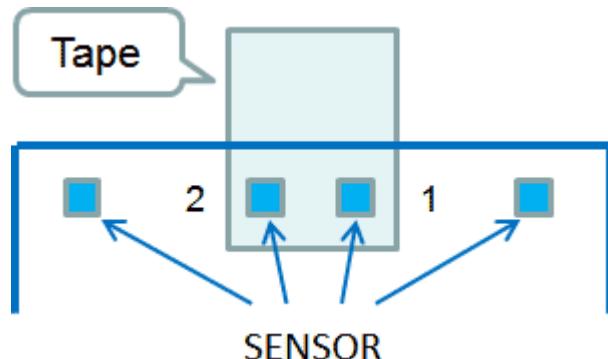
(5) At first, please call the “InitMotor(int rate)” function.
Argument “rate” is a ratio for the maximum speed.(0..100%)
If your car runs too rapid, you can lower this rate for overall speed.
You don’t need to adjust the rate of MotorDrive(FORWARD, 40,
50) for instance.



[2.Positional relationship]

First,

It is a positional relationship between the sensor position and black tape.

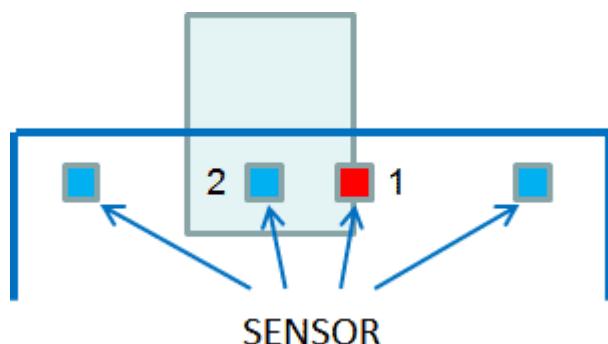


Because both sensors are on the tape, the value of the sensor 1 and 2 is about the same.

The state in which it runs along the tape, for example,

Sensor2	Sensor1
824	818
820	816
828	820
:	:
831	821

(1) When tracer is out of the tape position



Here, the value of sensor 1 and 2 will be different.

Sensor2	Sensor1
824	520
820	498
828	480
:	:
831	503

(2) Always we have to control as the difference between the value of both sensors is reduced.

For example, as the following way.

alpha is dead zone width.

```

if ( Left sensor < Right sensor - alpha ){
    Speed down ;
    Move to Left ;
}
else if ( Left sensor > Right sensor + alpha ){
    Speed down ;
    Move to Right ;
}

```

(3) However, how much, or would be good if we move it to the left and right?

First, let's test it by fixing the control amount.

Add code in front of “main()” function of
“LineTracer.cpp” file.

```
#define SENSOR_1      ad_data[1]
#define SENSOR_2      ad_data[2]
#define ALPHA        100    //tentative
```

(4) Modify in the “while loop” in “main()” function as follows.

```
bool runFlag = false;

while(1)          /* Main processing */
{
    Analog();
    Input();
    if ( runFlag == false ){
        if (IsStartPB() == true )
        {
            MotorDrive(FORWARD, 60, 60);      //%
            runFlag = true;
        }
    }
}
```

```
else
{
    if (IsStartPB() == true )
    {
        MotorDrive(STOP, 0, 0);
        runFlag = false;
    }
    else
    {
        if (SENSOR_1 < SENSOR_2 - ALPHA )
        {
            //% to Left
            MotorDrive(FORWARD, 50, 60 );
        }
        else if (SENSOR_1 > SENSOR_2 + ALPHA )
        {
            //% to Right
            MotorDrive(FORWARD, 60, 50 );
        }
        else
        {
            //% Straight
            MotorDrive(FORWARD, 60, 60 );
        }
    }
}
```

(5) When run, it runs along the line to some extent, but it occurs to protrude by the degree of the curve and speed.

Control amount must be varied depending on the magnitude of the difference.

To obtain the control amount, PID and Fuzzy is used, but here we use Fuzzy.

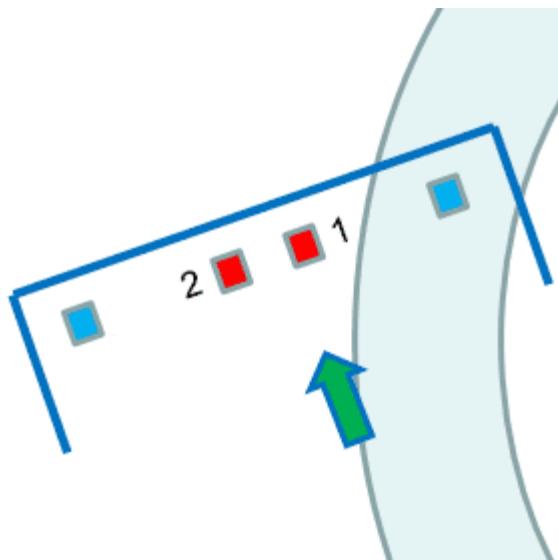
Fuzzy is described in the following chapters.

For example, do the following.

ZONE is dead zone width.

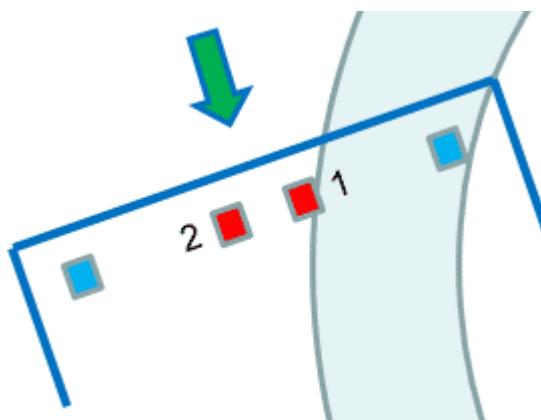
```
int diff = SENSOR_1 - SENSOR_2 ;
int result = Fuzzy( diff );
    //if small difference
if ( abs(result) < ZONE )
{
    int base = normalSpeed ;
    MotorDrive ( FORWARD, base, base );
}
    // if big difference
else
{
    int base = slowSpeed;
    MotorDrive (FORWARD, base+result, base-result );
}
```

(6) Also, if it has gone off the course as in the figure below,



It must go back on the track.

Even when go back, it would be good to control it as much as possible close to the course.



Sensor2	Sensor1
520	510
498	565
480	674
:	:
503	711

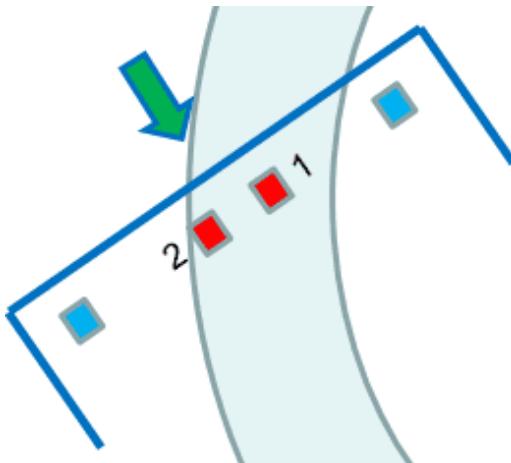
(7) Since the positional relationship is reversed this time,

```
if ( Left sensor < Right sensor - alpha ){
    Speed down ;
    Move to Right ;
}
else if ( Left sensor > Right sensor + alpha ){
    Speed down ;
    Move to Left ;
}
```

In the method of using the Fuzzy,

```
int base = slowSpeed;
MotorDrive (FORWARD, base-result, base+result );
```

..change the symbol to reverse.



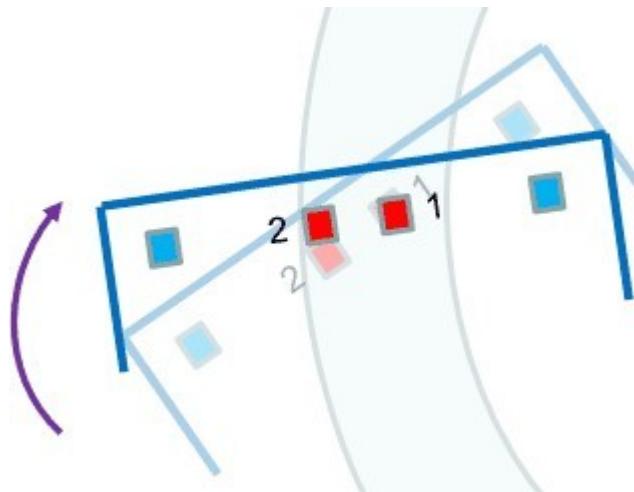
(8) When both sensors ride on the course, stop it.
As this position is tilted than when it was sticking out,
we have to modify the posture.

(9) When go back, we must remember which sensor got on tape first.

In the figure above, as the right sensor reached first, stop the right wheel and let a little forward only the left wheel.

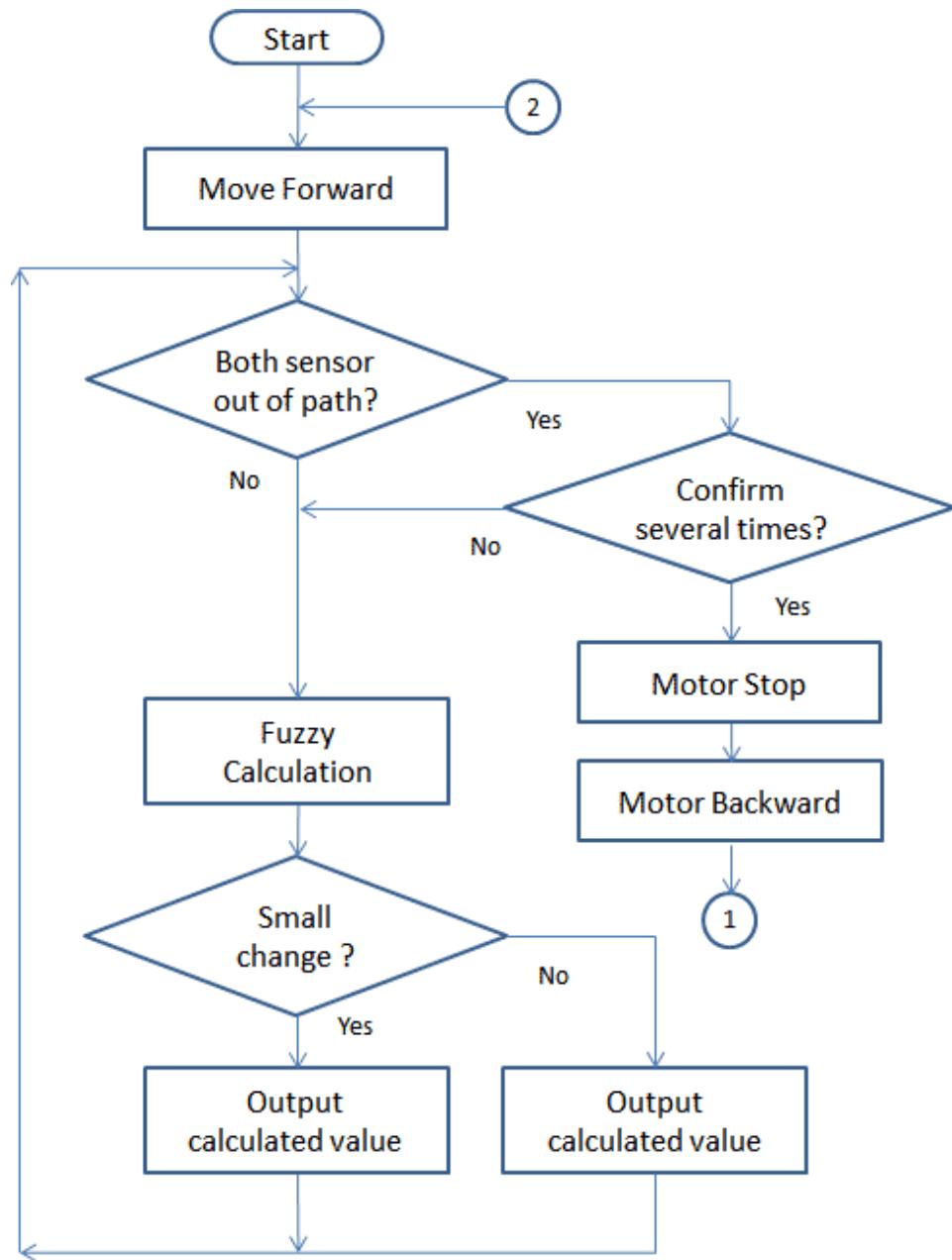
Since the encoder is not attached, advance amount will be the actual alignment in the balance with the speed.

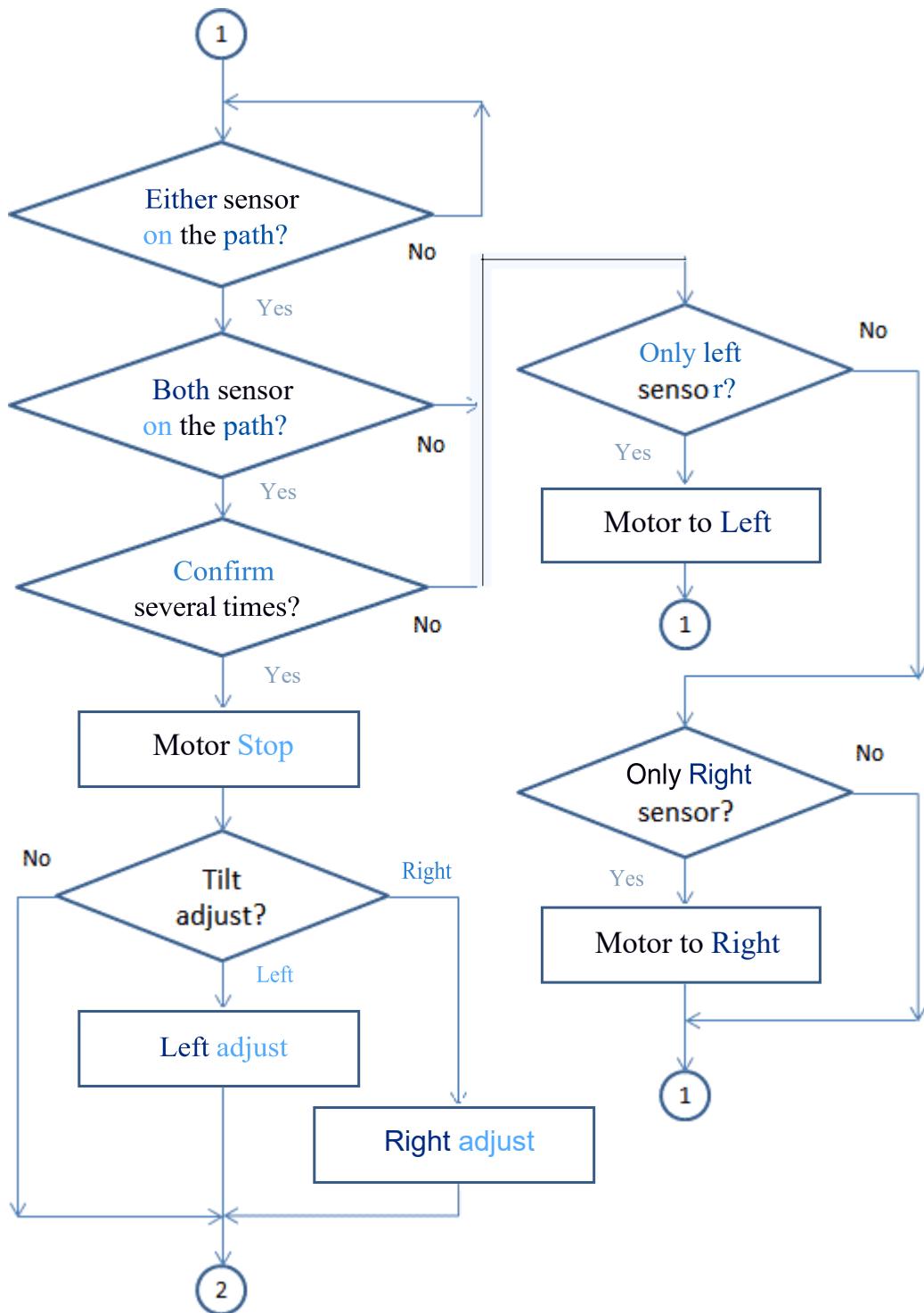
Time difference two sensors ride on the tape will be the guide



And return to normal forward.

[3.Flow chart of Motor Control]





Here in advance, let's prepare some of the function.

(1) Add the function in “Analog.cpp” file as follows.

```
//  
void GetValues(int data[])  
{  
    for ( int n = 0 ; n < 4 ; n++ )  
    {  
        data[n] = ad_data[n] ;  
    }  
}
```

(2) Create a new file named “Fuzzy.cpp”, and describe as follows.

```
#include “LineTracer.h”
```

```
int Fuzzy(int newVal)  
{  
    return 0;  
}
```

(3) Add to “Prototype.h” file.

```
void MotorStart(void);  
void MotorStop(void);  
void MotorControl(void);  
void GetValues(int data[]);
```

```
void InitFuzzy(void);
int Fuzzy(int newVal);
```

(4) Add some constant in “LineTracer.h” file.

```
#define DARK      570    //03C4=>964 700->570
#define BRIGHT    450    //0038 =>56  300->450
```

[4.Motor.cpp]

Modify “Motor.cpp” file as follows.

(1) First, add definitions and variables.

```
#define slowSpeed      50    // % tentative
#define normalSpeed     70    // % tentative
#define CONFIRM2        5     //tentative
#define CONFIRM3        30    //tentative
#define ZONE            5
#define BETA             50    //10->0
#define abs(n) ((n >= 0) ? n : -n )
static short counter ;
static short stage ;      //Control Stage
enum { S_FORWARD, S_BACKWARD, S_BACKWARD2,
       S_RETURN, S_RETURN2 };
static int oldResult ;    //Previous result
static int which ;
```

(2) Add three functions.

```
//  
void MotorStart()  
{  
    stage = S_FORWARD ;  
    counter = 0 ;  
    oldResult = 0 ;  
    MotorDrive( FORWARD, slowSpeed, slowSpeed );  
}  
//  
void MotorStop()  
{  
    MotorDrive( STOP, 0, 0 );  
}
```

```
//  
void MotorControl()  
{  
    int diff = 0 ;  
    int result = 0;  
    int sensor[4];  
  
    GetValues(sensor);      // Get sensor values  
  
    switch ( stage ){  
        case S_FORWARD :      //FORWARD  
            //Both sensor is out of the path  
            if( sensor[1] < BRIGHT && sensor[2] < BRIGHT )  
            {  
                //For confirmation  
                if(( counter += 1 ) >= CONFIRM2 )  
                {  
                    MotorDrive( STOP, 0, 0 );      //Pause  
                    stage = S_BACKWARD ;        //To backward  
                }  
            }  
    }  
}
```

```

else      //On the path
{
    counter = 0;
    diff = sensor[2] - sensor[1];
    result = Fuzzy( diff );
    if ( oldResult != result )
    {
        oldResult = result ;      //Previous result
        //When the operaton instruction is small
        if ( abs(result) <= ZONE )
        {
            int base = normalSpeed ;
            MotorDrive( FORWARD, base+result,
                        base-result );      //Normal speed
        }
        //When the operaton instruction is big
        else
        {
            int base = slowSpeed;
            MotorDrive( FORWARD, base+result,
                        base-result );      //Slow down
        }
    }
    break ;
}

```

```

case S_BACKWARD :      //BACKWARD
    MotorDrive( BACKWARD, slowSpeed,
                slowSpeed );
    counter = 0 ;
    which = 0;
    stage++;
    break;
case S_BACKWARD2 :
    //Either of the sensor is on the path
    if ( sensor[1] >= DARK || sensor[2] >= DARK )
    {
        //For confirmation,
        //until both sensor is on the path
        if( sensor[1] >= DARK && sensor[2] >= DARK )
        {
            if (( counter += 1 ) > CONFIRM3 )
            {
                MotorDrive( STOP, 0,0 );      //Pause
                stage = S_RETURN ;          //Next stage
                break ;
            }
        }
        else
        {
            counter = 0;
        }
    }

```

```

        //Left sensor is on the path
if( sensor[2] > sensor[1] + BETA )
{
    int base = slowSpeed;      //Slow down
    //Modify the direction
    MotorDrive( TOLEFT, base-20, base+20 );
    if( which == o )
        which = TOLEFT;
}
//Right sensor is on the path
else if( sensor[2] < sensor[1] - BETA )
{
    int base = slowSpeed;      //Slow down
    //Modify the direction
    MotorDrive( TORIGHT, base+20, base-20 );
    if( which == o )
        which = TORIGHT;
}
else
{
    counter = o;
}
break ;

```

```
case S_RETURN :      //Modify the direction
    //Modify the direction using the value of counter
    if( which == TOLEFT )
    {
        int base = slowSpeed;      //Slow down
        //Left fix, move only Right wheel
        MotorDrive( FORWARD, 0, base+20 );
    }
    else if( which == TORIGHT)
    {
        int base = slowSpeed;      //Slow down
        //Right fix, move only Left wheel
        MotorDrive( FORWARD, base+20,0 );
    }
    if(( counter -= 1) <= 0 )
        stage = S_RETURN2 ;
    break ;
case S_RETURN2 :      //Return to Forward
    counter = 0 ;
    oldResult = 0 ;
    stage = S_FORWARD ;
    break ;
}
```

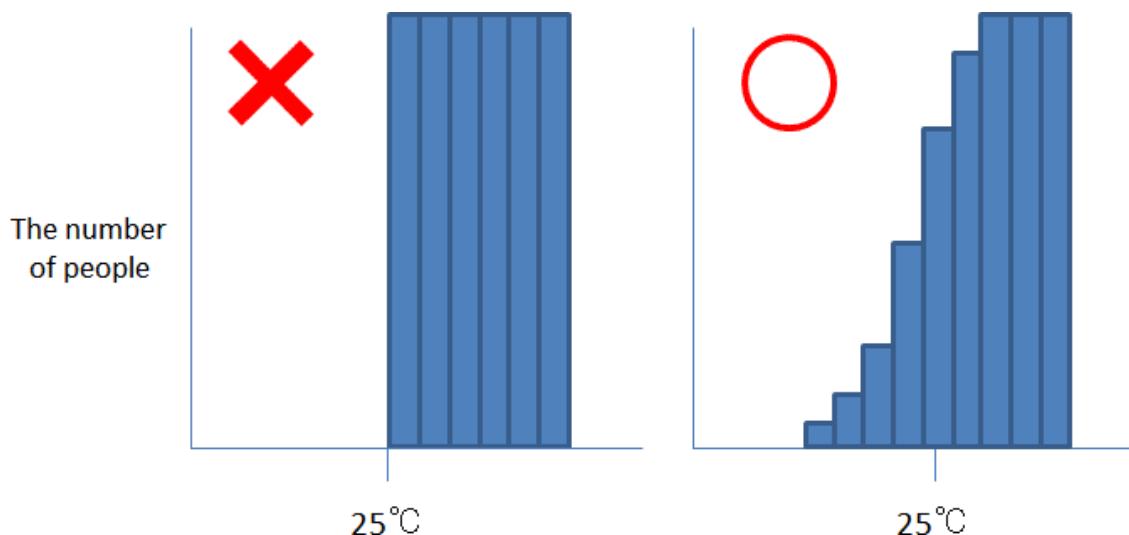
[8] Fuzzy

Fuzzy Reasoning has proven to be a very successful mechanism, especially in the field of fuzzy control.

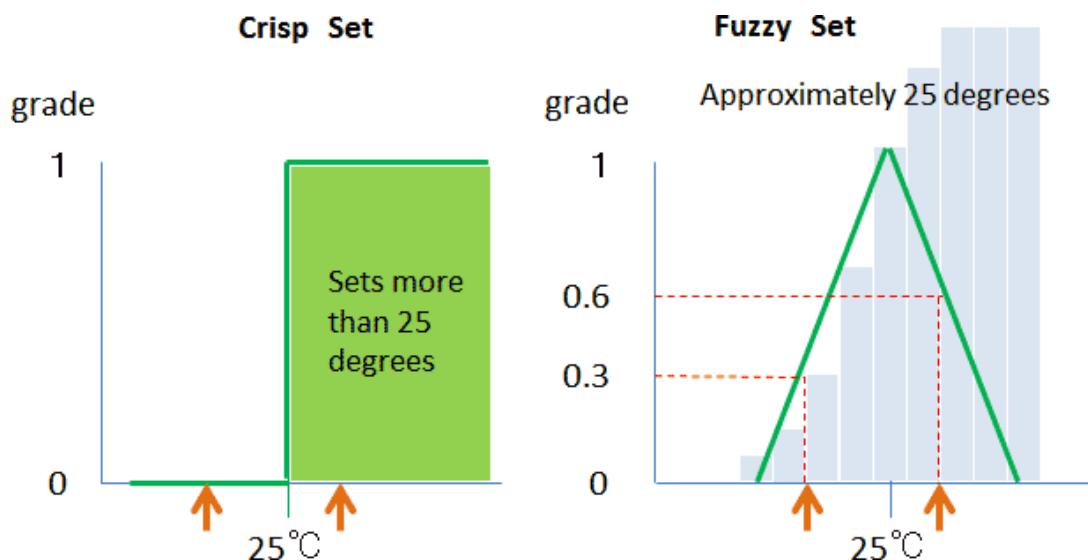
(1) For instance,

“The temperature that the persons want to cool”

How do you express this state?



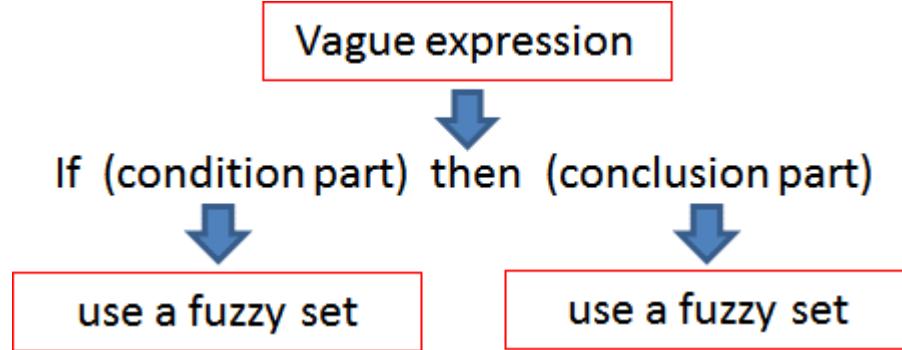
(2) Numerical expression as "20 degrees Celsius" or "50km/h" shows whether it belongs to a constant value, or not.
Crisp set means such an expression method.



(3) In contrast, fuzzy expression indicates how much it belongs to a certain range.

If I feel room temperature to be slightly hot, I strengthen some air conditioners.

If I feel room temperature to be slightly cold, I weaken some air conditioners.

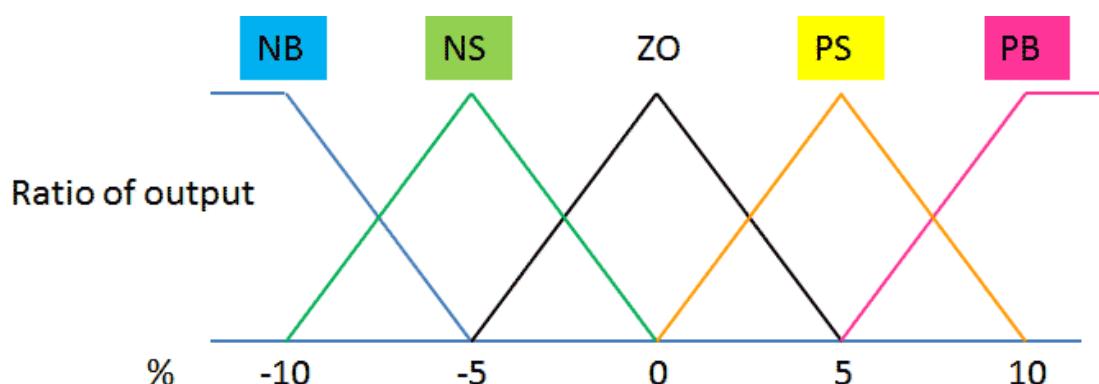
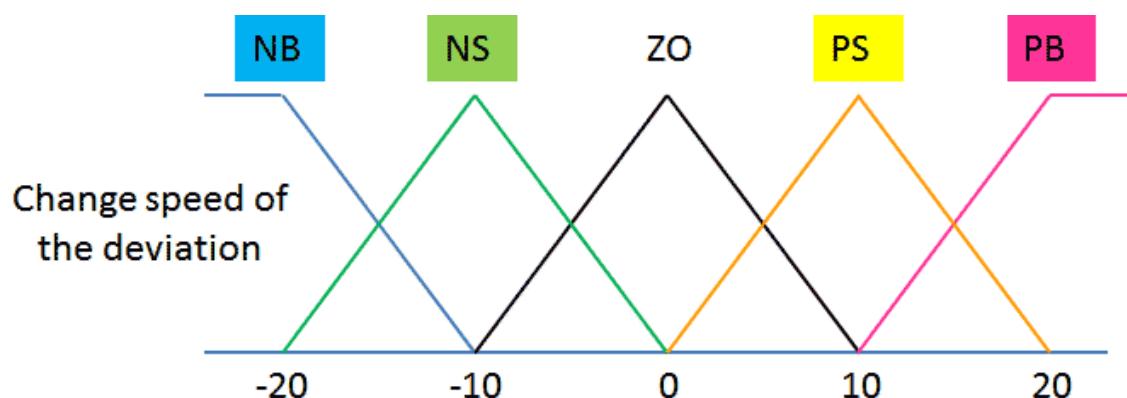
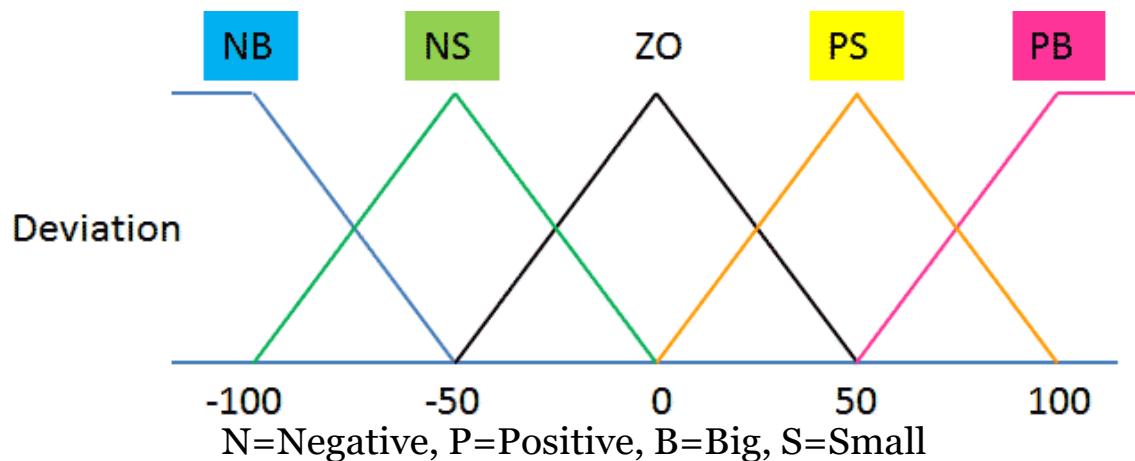


- a) Human know-how about the operation is available as the reasoning .
- b) Because we can make a general decision, the careful reasoning is possible.
- c) Precise mathematics model is unnecessary.
- d) The result that is almost a human sense is provided.

(4) The design procedure of the fuzzy control,

- a) Making of the fuzzy set.
- b) Making of the fuzzy rule.

[1.Fuzzy set]



[2.Fuzzy rule]

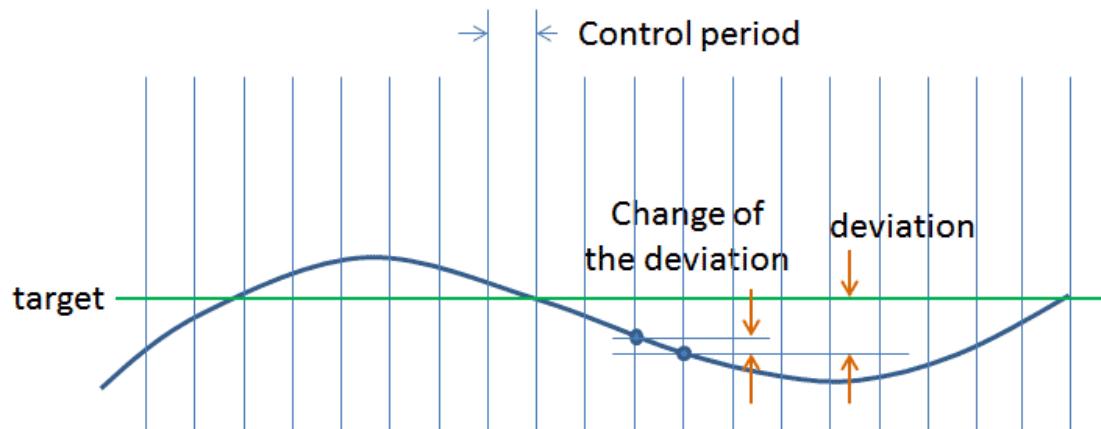
If deviation==ZO & Speed==ZO then Ratio =ZO
If deviation==ZO & Speed==PS then Ratio = NS
If deviation==ZO & Speed==PB then Ratio = NB
If deviation==ZO & Speed==NS then Ratio = PS
If deviation==ZO & Speed==NB then Ratio = PBI
If deviation==PS & Speed==ZO then Ratio =NS

:
If deviation==PB & Speed==ZO then Ratio = NB

:
If deviation==NS & Speed==ZO then Ratio = PS

:
If deviation==NB & Speed==ZO then Ratio = PB

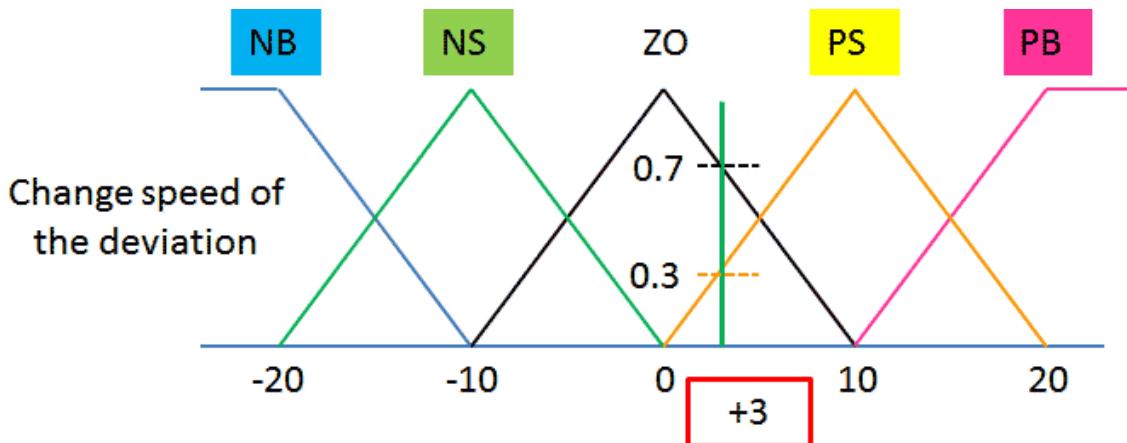
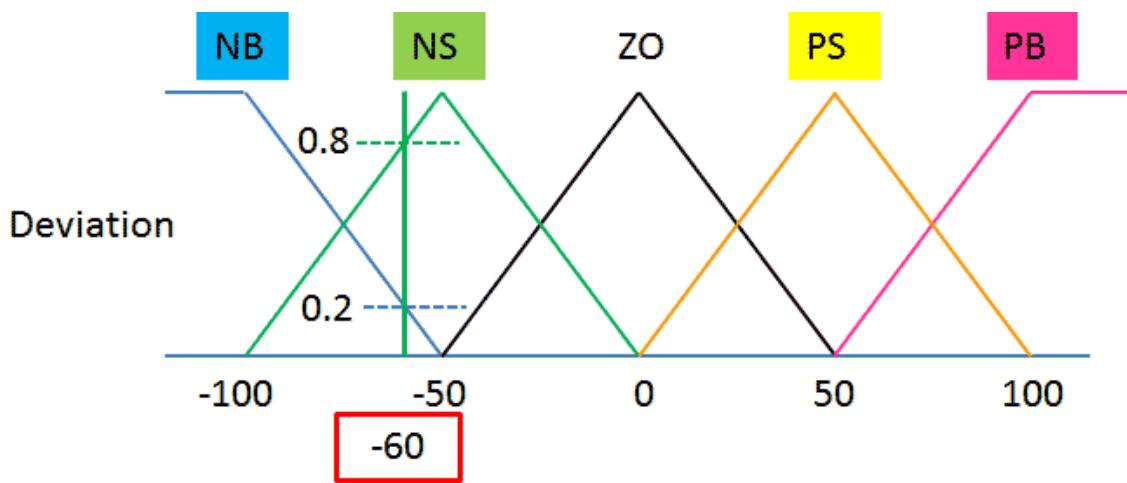
:



[3.Rule table]

		Change of the deviation				
		-20	-10	0	10	20
Number of rotation	NB	NS	ZO	PS	PB	
	NB	PB	PB	PB	PS	ZO
	NS	PB	PB	PS	ZO	NS
	ZO	PB	PS	ZO	NS	NB
	PS	PS	ZO	NS	NB	NB
	PB	ZO	NS	NB	NB	NB

N=Negative, P=Positive, B=Big, S=Small



In case of deviation = -60, the change of deviation = +3

		-20	-10	0	10	20	
		NB	NS	ZO	PS	PB	
0.2	-100	NB	PB	PB	PB	PS	ZO
0.8	-50	NS	PB	PB	PS	ZO	NS
	0	ZO	PB	PS	ZO	NS	NB
	50	PS	PS	ZO	NS	NB	NB
	100	PB	ZO	NS	NB	NB	NB

[4.Min-Max method]

Min-Max method

The former part is an AND operation=Use the smaller one

PB	0.2	PS	0.2
PS	0.7	ZO	0.3

The overlap part is an OR operation=Use the larger one

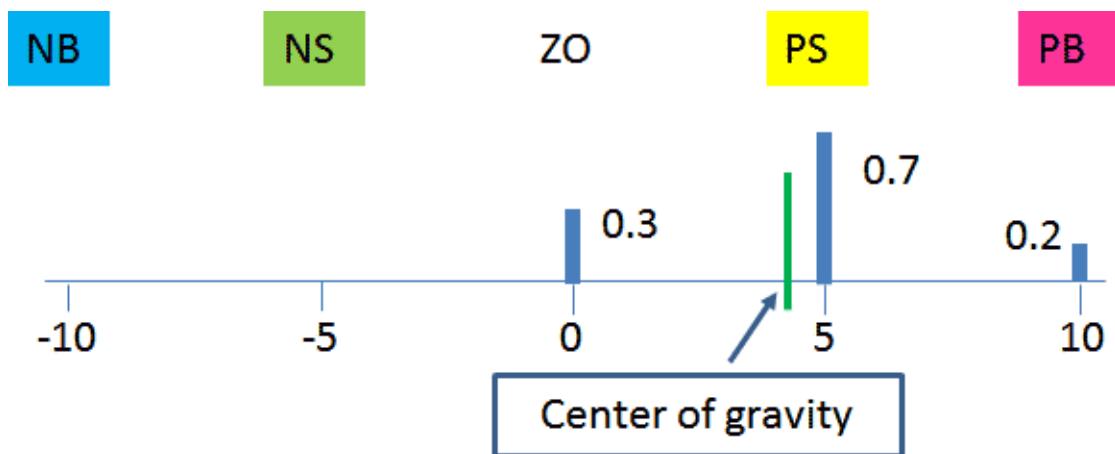
PB	0.2		
PS	0.7	ZO	0.3

The result so far =

ZO	0.3	PS	0.7	PB	0.2
----	-----	----	-----	----	-----

[5.Calculate Output]

The latter part (Output increase and decrease value)



(6) Expression for the center of gravity

$$Z_0 = \frac{NB * h_1 + NS * h_2 + ZO * h_3 + PS * h_4 + PB * h_5}{h_1 + h_2 + h_3 + h_4 + h_5}$$

From h_1 to h_5 is each grade level

Calculate the center of gravity

$$a = 0 * 0.3 + 5 * 0.7 + 10 * 0.2 = 5.5$$

$$b = 0.3 + 0.7 + 0.2 = 1.2$$

$$\begin{aligned} \text{the center of gravity} &= a / b \\ &= 4.58 \end{aligned}$$

This tiny MPU doesn't have the floating point circuit.
So we have to use the integer calculation.
This time we need fast speed and calculation, but we don't need accuracy.

For example, if you use 8 instead of 0.8, you can get good result.

[6.Fuzzy.cpp]

Add in “Fuzzy.cpp” file.

(1) Let's prepare constant tables in the file.

This MCU can not use the real, we have to 10 times the value.

```
//These values are tentative.
```

```
//Please adjust depending on the situation.
```

```
// The difference between the left and right of the sensor value
```

```
static const long deviation[] = { -1600, -800, 0, 800, 1600};
```

```
// Variation of the difference between the left and right of the  
sensor value
```

```
static const long change[] = { -300, -150, 0, 150, 300};
```

```
// Output value
```

```
static const long output[] = { -200, -100, 0, 100, 200};
```

(2) Provide a value that indicates each zone of the table

		Change of the deviation				
		-20	-10	0	10	20
		NB	NS	ZO	PS	PB
-100	NB	PB	PB	PB	PS	ZO
-50	NS	PB	PB	PS	ZO	NS
0	ZO	PB	PS	ZO	NS	NB
50	PS	PS	ZO	NS	NB	NB
100	PB	ZO	NS	NB	NB	NB

```

static const short areaPB[6][2] = {{0,0},{0,1},{0,2},{1,0},{1,1},{2,0}};
static const short areaPS[4][2] = {{0,3},{1,2},{2,1},{3,0}};
static const short areaZO[5][2] = {{0,4},{1,3},{2,2},{3,1},{4,0}};
static const short areaNS[4][2] = {{1,4},{2,3},{3,2},{4,1}};
static const short areaNB[6][2] = {{2,4},{3,3},{3,4},{4,2},{4,3},{4,4}};

```

(3) Prepare variables.

Because we don't specify a lot of stack space, we have brought variables to the global area.

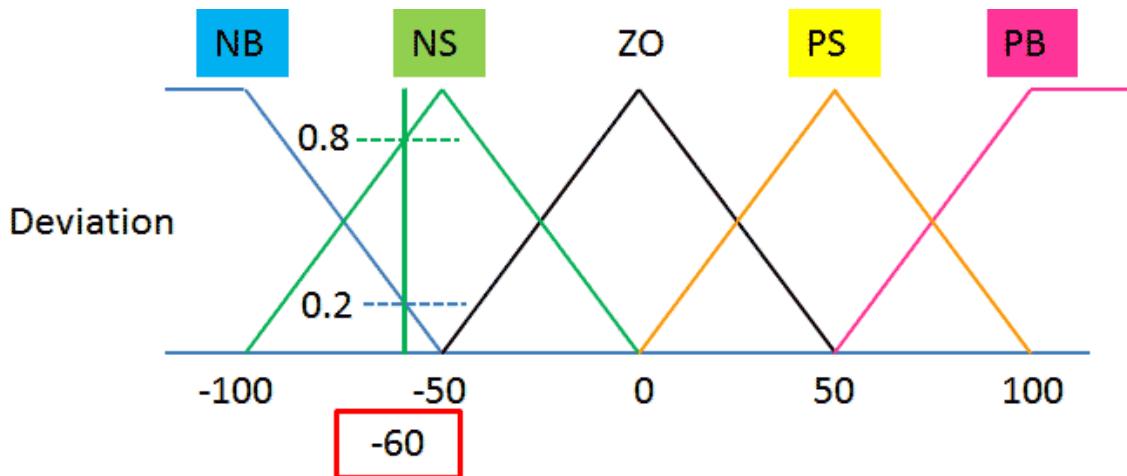
```
static long oldVal ;           // store the previous value
static long DATA1[5];          // the deviation
static long DATA2[5];          // the change of the deviation
static long result[5][5];       //result
static long dA ;               //AND operation
static long dB ;               //AND operation
static long oNB ;              //OR operation
static long oNS ;              //OR operation
static long oZO ;              //OR operation
static long oPS ;              //OR operation
static long oPB ;              //OR operation
```

(4) In addition, prepare the two methods.

```
void InitFuzzy()
{
    oldVal = 0 ;           // store the previous value
}
int Fuzzy(int newVal)
{
}
```

(5) Describe in this Fuzzy method.

```
int Fuzzy(int newVal)
{
    long now = newVal * 10; // Difference
    long diff = now - oldVal; // Variation of the difference
    oldVal = now;
    long nume = 0; // numerator
    long deno = 0; // denominator
```



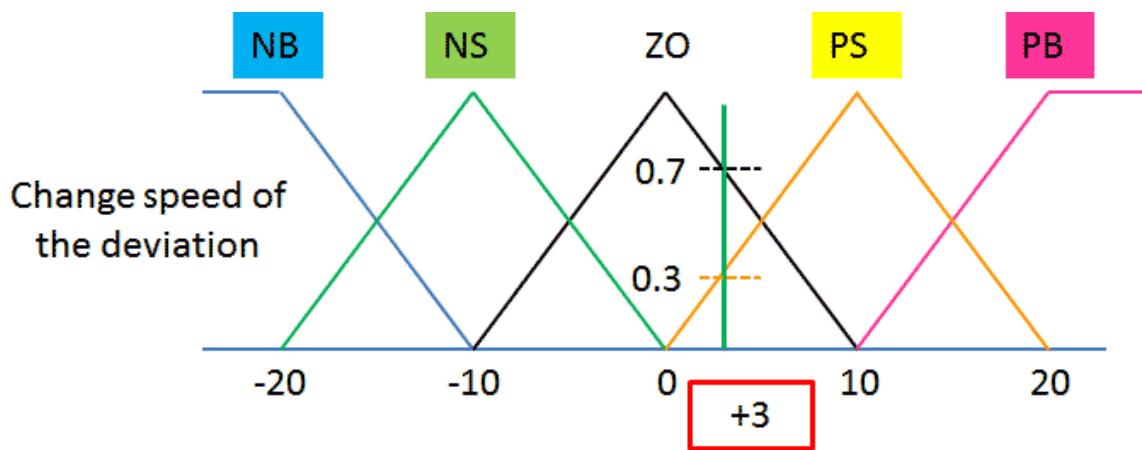
```
// calculate the deviation
int n ;
for ( n = 0; n < 5 ; n++)
    data1[N] = 0;
if ( now < deviation[0])
    data1[0] = 10;
else if ( now >= deviation[4])
    data1[4] = 10;
else
```

```

{
for ( n = 0; n < 4 ; n++)
{
    if( now >= deviation[n]
        && now < deviation[n+1])
    {
        nume = ( now - deviation[n] ) * 10;
        deno = deviation[n+1] - deviation[n];
        if( deno > 0)
        {
            data1[N+1] = nume / deno;
            data1[N] = 10 - data1[N+1];
        }
        break;
    }
}

```

(6) Then data1 table will be { 0.2, 0.8, 0, 0, 0 }



```

//calculate the change of the deviation
for ( n = 0; n < 5 ; n++)
    data2[n] = 0;
if ( diff < change[0])
    data2[0] = 10;
else if ( diff >= change[4])
    data2[4] = 10;
else
{
    for ( n = 0; n < 4 ; n++)
    {
        if ( diff >= change[n] && diff < change[N+1])
        {
            nume = ( diff - change[n] ) * 10;
            deno = change[N+1] - change[n];

```

```

        if ( deno > 0)
        {
            data2[N+1] = nume / deno;
            data2[n] = 10 - data2[N+1];
        }
        break;
    }
}

```

(7) Then data2 table will be { 0, 0, 0.7, 0.3, 0 }

		-20	-10	0	10	20
		NB	NS	ZO	PS	PB
0.2	-100	NB	PB	PB	PB	PS
0.8	-50	NS	PB	PB	PS	ZO
	0	ZO	PB	PS	ZO	NS
	50	PS	PS	ZO	NS	NB
	100	PB	ZO	NS	NB	NB

```
//AND operation
dA = 0;
dB = 0;
int m ;
for ( m = 0 ; m < 5 ; m++)
{
    for ( n = 0 ; n < 5 ; n++)
    {
        result[m][n] = 0;
    }
}
for ( m = 0 ; m < 5 ; m++)
{
    dA = DATA1[m];
    for ( n = 0; n < 5 ; n++)
    {
        dB = DATA2[n];
        if ( dA <= dB)
            result[m][n] = dA;
        else
            result[m][n] = dB;
    }
}
```

(8) The result table will be

{ 0, 0, 0.2, 0.2, 0}
{ 0, 0, 0.7, 0.3, 0}
{ 0, 0, 0, 0, 0}
{ 0, 0, 0, 0, 0}
{ 0, 0, 0, 0, 0}

Min-Max method

The former part is an AND
operation=Use the smaller one

PB	0.2	PS	0.2
PS	0.7	ZO	0.3

The overlap part is an OR
operation=Use the larger one

PB	0.2		
PS	0.7	ZO	0.3

```
//OR operation
oNB = o;
oNS = o;
oZO = o;
oPS = o;
oPB = o;
int j, k;
//Max of oPB
for ( n = o; n < 6 ; n++)
{
    j = areaPB[n][0];
    k = areaPB[n][1];
    if ( n == o)
        oPB = result[j][k];
    else
    {
        if ( oPB < result[j][k])
            oPB = result[j][k];
    }
}
```

```

//Max of oPS
for ( n = 0; n < 4 ; n++)
{
    j = areaPS[n][0];
    k = areaPS[n][1];
    if ( n == 0)
        oPS = result[j][k];
    else
    {
        if ( oPS < result[j][k])
            oPS = result[j][k];
    }
}

//Max of oZO
for ( n = 0; n < 5 ; n++)
{
    j = areaZO[n][0];
    k = areaZO[N][1];
    if ( n == 0)
        oZO = result[j][k];
    else
    {
        if ( oZO < result[j][k])
            oZO = result[j][k];
    }
}

```

```

//Max of oNS
for ( n = 0; n < 4 ; n++)
{
    j = areaNS[n][0];
    k = areaNS[N][1];
    if ( n == 0)
        oNS = result[j][k];
    else
    {
        if ( oNS < result[j][k])
            oNS = result[j][k];
    }
}

//Max of oNB
for ( n = 0; n < 6 ; n++)
{
    j = areaNB[n][0];
    k = areaNB[N][1];
    if ( n == 0)
        oNB = result[j][k];
    else
    {
        if ( oNB < result[j][k])
            oNB = result[j][k];
    }
}

```

(9) Then the result table will be

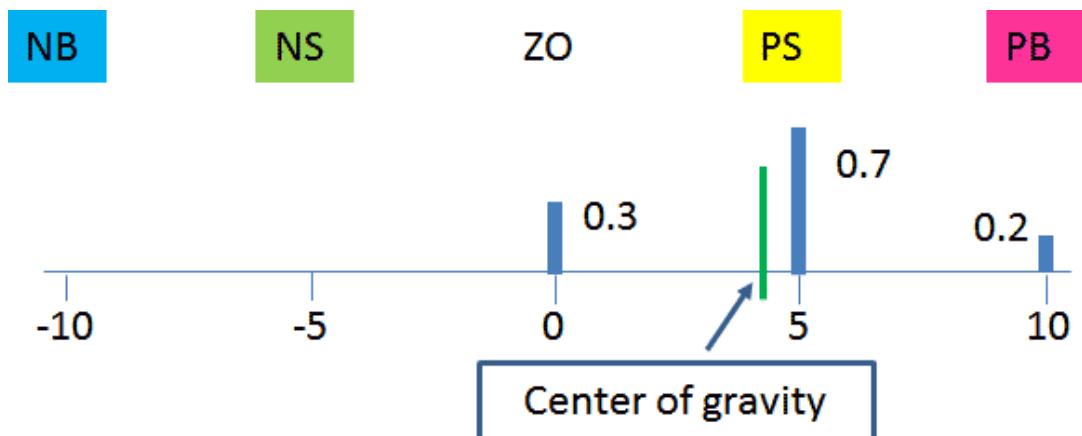
- { 0, 0, 0.2, 0, 0}
- { 0, 0, 0.7, 0.3, 0}
- { 0, 0, 0, 0, 0}
- { 0, 0, 0, 0, 0}
- { 0, 0, 0, 0, 0}

(10) The result table means,

- { 0, 0, PB=0.2, 0, 0}
- { 0, 0, PS=0.7, ZO=0.3, 0}
- { 0, 0, 0, 0, 0}
- { 0, 0, 0, 0, 0}
- { 0, 0, 0, 0, 0}

The result so far =

ZO	0.3	PS	0.7	PB	0.2
----	-----	----	-----	----	-----



```

// calculate gravity
long gravity = 0;
deno = oNB + oNS + oZO + oPS + oPB ;
if ( deno > 0)
{
    gravity = oNB * output[0] + oNS * output[1]
    + oZO * output[2] + oPS * output[3]
    + oPB * output[4];
    gravity /= deno;
}
return (int)( gravity / 10 );
}

```

- (11) Then you can call Fuzzy() method from MotorControl() function.

```

void MotorControl()
{
    :
    diff = sensor[2] - sensor[1];
    result = Fuzzy( diff );
    if ( oldResult != result )
    {
        :
    }
}

```

(12) Then we can call “MotorControl()” function as follows.
In “Main” function,

```
while(1)      /* Main processing */  
{  
    Analog();  
    Input();  
    if ( runFlag == false )  
    {  
        if ( IsStartPB() == true )  
        {  
            InitFuzzy();  
            MotorStart();  
            runFlag = true ;  
        }  
    }  
}
```

```
else
{
    if ( IsStartPB() == true )
    {
        MotorStop();
        runFlag = false ;
    }
    else
    {
        MotorControl();
    }
}
```

{Exercise 2}

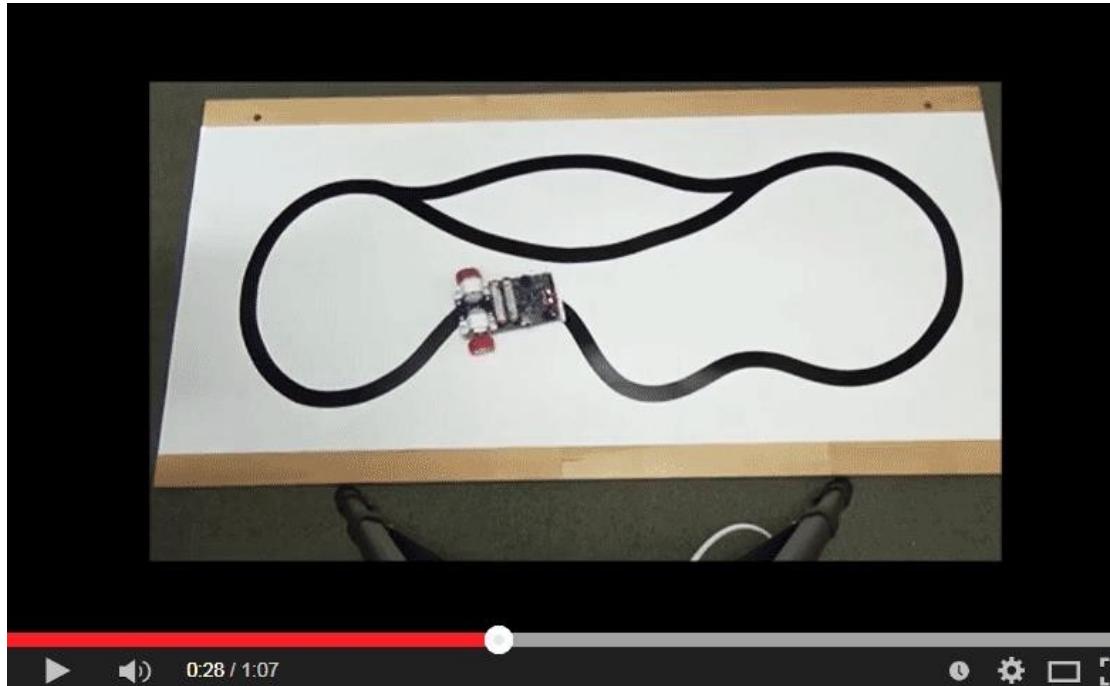
By changing the parameters of the Fuzzy, please examine the behavior.

{Exercise 3}

Also, please consider making a Fuzzy in a different way.

Please refer to the demonstration of the line tracer.

https://www.youtube.com/watch?v=387E_ZM8eWk

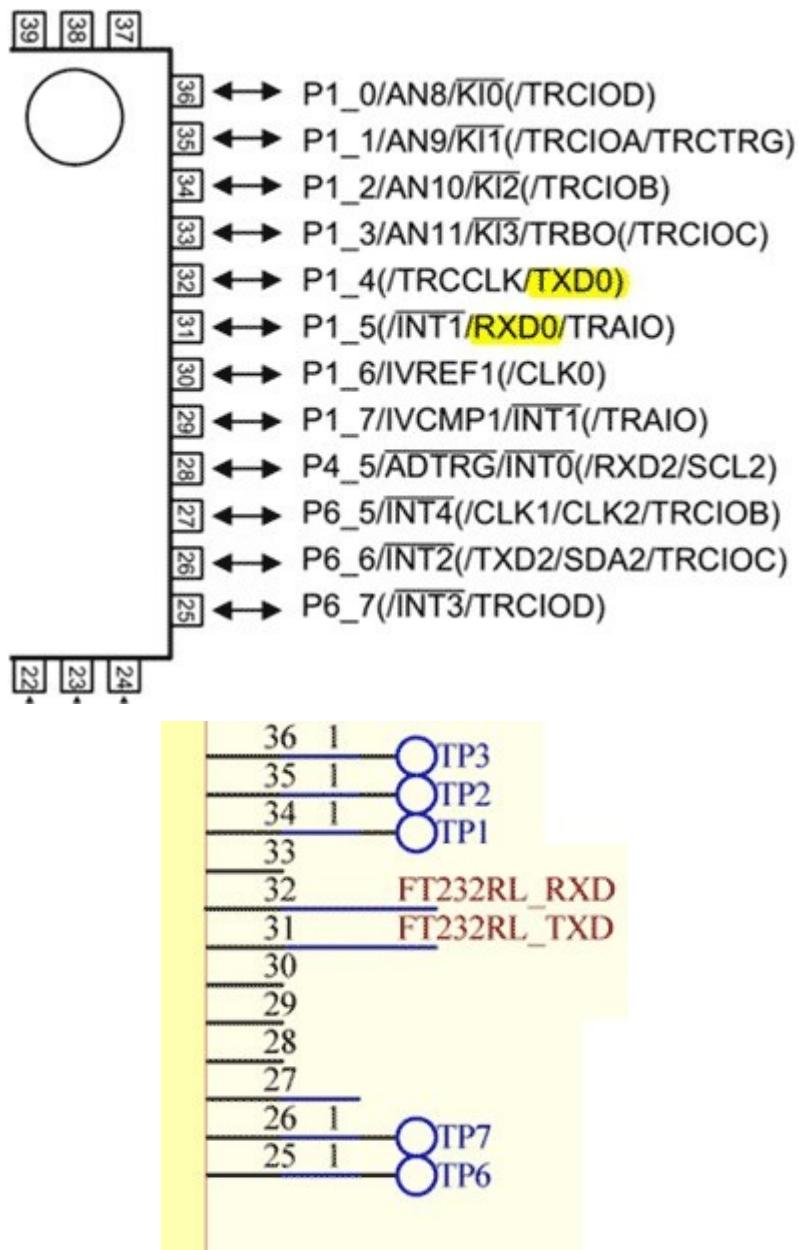


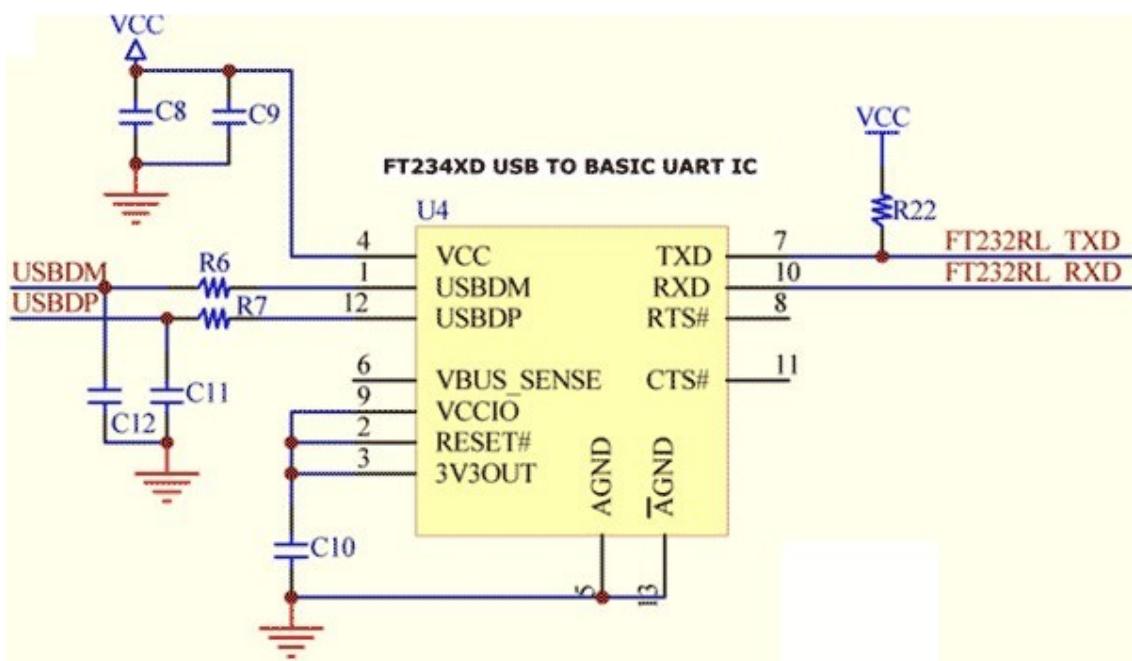
[9] UART

When you run the line tracer, you'll think that you want to know more in detail the internal situation of it.

Communication port is available, and you can use the cable that was used to write, let's send the internal state to the PC.

(1) First, we prepare so that communication is usable.





(2) Add a function in “Hwsetup()” as follows,

```
static void uart_init(void);
```

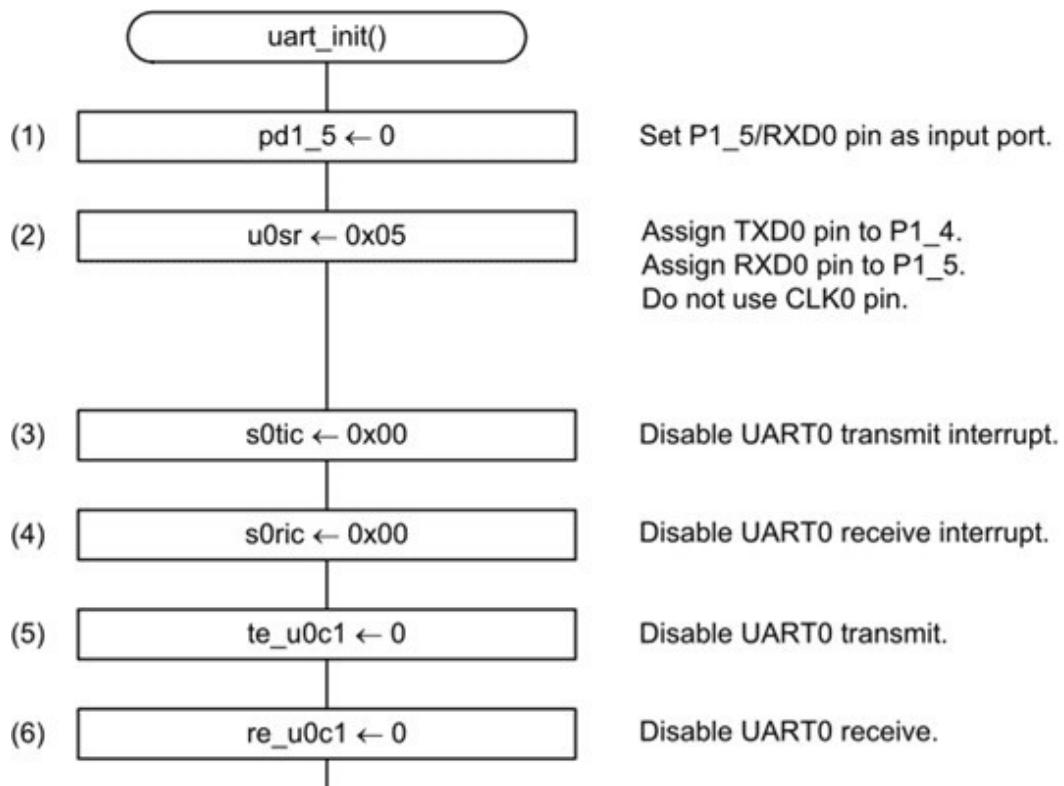
```
void Hwsetup()
{
    mcu_init();
    port_init();
    ad_init();
    motor_init();
    uart_init();
}
//
```

```
static void uart_init()
{}
```

Described in “uart_init()”.

[1 uart_init]

We set in accordance with the flow chart of “RENESAS”.



(1)Port Direction

7.4.1 Port Pi Direction Register (PDi) (i = 0 to 4 or 6)

Address 00E2h (PD0 ⁽¹⁾), 00E3h (PD1), 00E6h (PD2), 00E7h (PD3 ⁽²⁾), 00EAh (PD4 ⁽³⁾), 00EE

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	PDi_7	PDi_6	PDi_5	PDi_4	PDi_3	PDi_2	PDi_1	PDi_0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	PDi_0	Port Pi_0 direction bit	0: Input mode (functions as an input port) 1: Output mode (functions as an output port)
b1	PDi_1	Port Pi_1 direction bit	
b2	PDi_2	Port Pi_2 direction bit	
b3	PDi_3	Port Pi_3 direction bit	
b4	PDi_4	Port Pi_4 direction bit	
b5	PDi_5	Port Pi_5 direction bit	
b6	PDi_6	Port Pi_6 direction bit	
b7	PDi_7	Port Pi_7 direction bit	

pd1_5 = 0; /* Pd1_5/RXD0 = input */

(2)UART Pin Select

7.4.10 UART0 Pin Select Register (U0SR)

Address 0188h

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	CLK0SEL0	—	RXD0SEL0	—	TXD0SEL0
After Reset	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Function
b0	TXD0SEL0	TXD0 pin select bit	0: TXD0 pin not used 1: P1_4 assigned
b1	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b2	RXD0SEL0	RXD0 pin select bit	0: RXD0 pin not used 1: P1_5 assigned
b3	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b4	CLK0SEL0	CLK0 pin select bit	0: CLK0 pin not used 1: P1_6 assigned

u0sr = 0x05; /* TXDo pin : P1_4 */

/* RXDo pin : P1_5 */

/* CLK0 pin : not used */

11.2.1 Interrupt Control Register

(TREIC, S2TIC, S2RIC, KUPIC, ADIC, SOTIC, SORIC, S1TIC, S1RIC,
TRAIC, TRBIC, U2BCNIC, VCMP1IC, VCMP2IC)

Address 004Ah (TREIC), 004Bh (S2TIC), 004Ch (S2RIC), 004Dh (KUPIC), 004Eh (ADIC),
0051h (SOTIC), 0052h (SORIC), 0053h (S1TIC), 0054h (S1RIC), 0056h (TRAIC),
0058h (TRBIC), 005Eh (U2BCNIC), 0072h (VCMP1IC), 0073h (VCMP2IC),

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	IR	ILVL2	ILVL1	ILVL0
After Reset	X	X	X	X	X	0	0	0

Bit	Symbol	Bit Name	Function
b0	ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0: Level 0 (interrupt disabled) 0 0 1: Level 1 0 1 0: Level 2 0 1 1: Level 3 1 0 0: Level 4 1 0 1: Level 5 1 1 0: Level 6 1 1 1: Level 7
b1	ILVL1		
b2	ILVL2		

```
sotic = 0x04; /* UART0 transmit interrupt level 4 */
soric = 0x06; /* UART0 receive interrupt level 6 */
```

22.2.5 UART*i* Transmit/Receive Control Register 1 (UiC1) (*i* = 0 or 1)

Address 00A5h (U0C1), 0165h (U1C1)

Symbol	Bit b7	b6	b5	b4	b3	b2	b1	b0
After Reset	—	—	UiRRM	UiIRS	RI	RE	TI	TE

Bit	Symbol	Bit Name	Function
b0	TE	Transmit enable bit	0: Transmission disabled 1: Transmission enabled
b1	TI	Transmit buffer empty flag	0: Data present in the UiTB register 1: No data in the UiTB register
b2	RE	Receive enable bit	0: Reception disabled 1: Reception enabled
b3	RI	Receive complete flag (1)	0: No data in the UiRB register 1: Data present in the UiRB register
b4	UiIRS	UART <i>i</i> transmit interrupt source select bit	0: Transmission buffer empty (TI = 1) 1: Transmission completed (TXEPT = 1)
b5	UiRRM	UART <i>i</i> continuous receive mode enable bit (2)	0: Continuous receive mode disabled 1: Continuous receive mode enabled

```
te_uoc1 = 0;      /* Transmit enable : disabled */
re_uoc1 = 0;      /* Receive enable : disabled */
```

22.2.1 UART*i* Transmit/Receive Mode Register (UiMR) (*i* = 0 or 1)

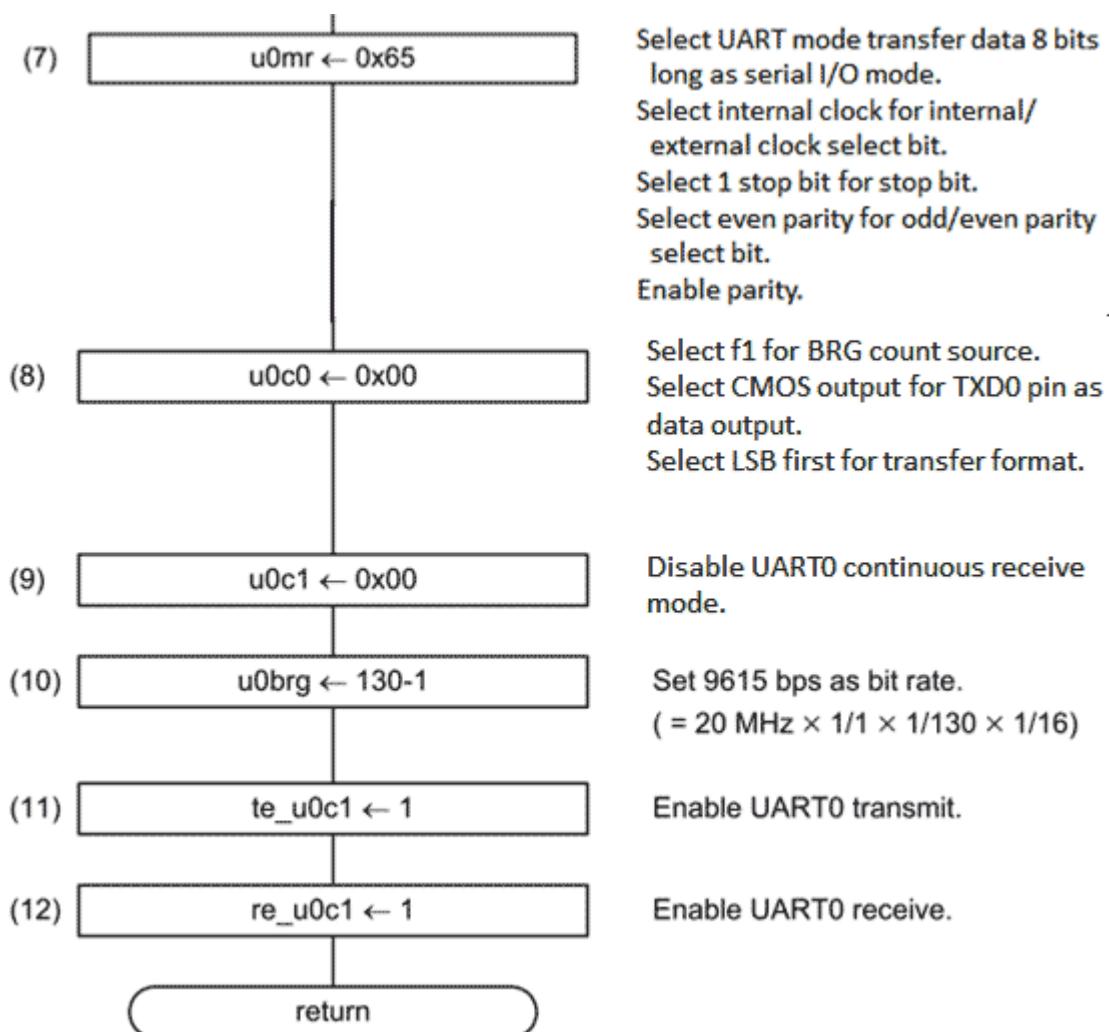
Address 00A0h (U0MR), 0160h (U1MR)

Symbol	Bit b7	b6	b5	b4	b3	b2	b1	b0
After Reset	—	PRYE	PRY	STPS	CKDIR	SMD2	SMD1	SMD0

Bit	Symbol	Bit Name	Function
b0	SMD0	Serial I/O mode select bit	b2 b1 b0 0 0 0: Serial interface disabled 0 0 1: Clock synchronous serial I/O mode 1 0 0: UART mode, transfer data 7 bits long 1 0 1: UART mode, transfer data 8 bits long 1 1 0: UART mode, transfer data 9 bits long Other than above: Do not set.
b1	SMD1		
b2	SMD2		
b3	CKDIR	Internal/external clock select bit	0: Internal clock 1: External clock
b4	STPS	Stop bit length select bit	0: One stop bit 1: Two stop bits
b5	PRY	Odd/even parity select bit	Enabled when PRYE = 1 0: Odd parity 1: Even parity
b6	PRYE	Parity enable bit	0: Parity disabled 1: Parity enabled

(3) UART Mode

```
uomr = 0x65; /* Serial I/O mode : UART mode, */
               /* transfer data 8 bits long */
               /* Internal/external clock : Internal clock */
               /* Stop bit length : One stop bit */
               /* Odd/even parity : Even */
               /* Parity enable : Parity enabled */
```



```
uoco = 0x00; /* BRG count source : F1 */
              /* Data output : TXD0 pin set to */
              /* CMOS output */
              /* Transfer format : LSB first */
```

```

UOC1 = 0x00;      /* Transmit enable : disabled */
                  /* Receive enable : disabled */
                  /* UART0 transmit interrupt source : */
                  /* Transmission completed (TI= 1) */
                  /* UART0 continuous receive mode : */
                  /* Continuous receive mode disabled */

```

(4) UART Control

22.2.4 **UART*i* Transmit/Receive Control Register 0 (UiC0) (i = 0 or 1)**

Address 00A4h (U0C0), 0164h (U1C0)

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	UFORM	CKPOL	NCH	—	TXEPT	—	CLK1	CLK0
After Reset	0	0	0	0	1	0	0	0

Bit	Symbol	Bit Name	Function
b0	CLK0	BRG count source select bit (1)	b1 b0 0 0: f1 selected 0 1: f8 selected 1 0: f32 selected 1 1: fC selected
b1	CLK1		
b2	—	Reserved bit	Set to 0.
b3	TXEPT	Transmit register empty flag	0: Data present in the transmit register (transmission in progress) 1: No data in the transmit register (transmission completed)
b4	—	Nothing is assigned. If necessary, set to 0. When read, the content is 0.	
b5	NCH	Data output select bit	0: TXDi pin set to CMOS output 1: TXDi pin set to N-channel open-drain output
b6	CKPOL	CLK polarity select bit	0: Transmit data output at the falling edge and data input at the rising edge of the transfer 1: Transmit data output at the rising edge and data input at the falling edge of the transfer
b7	UFORM	Transfer format select bit	0: LSB first 1: MSB first

22.2.2 **UART*i* Bit Rate Register (UiBRG) (*i* = 0 or 1)**

Address 00A1h (U0BRG), 0161h (U1BRG)

Bit	b7	b6	b5	b4	b3	b2	b1	b0
Symbol	—	—	—	—	—	—	—	—
After Reset	X	X	X	X	X	X	X	X

Bit	Function	Setting Range
b7 to b0	If the setting value is n, UiBRG divides the count source by n+1.	00h to FFh

```
uobrg = 130-1; /* 20MHz * 1/1 * 1/(129+1) * 1/16 = 9615bps */
//te_uoc1 = 1; /* Transmit enable : enabled */
//re_uoc1 = 1; /* Receive enable : enabled */
```

[2.Summary]

```
static void uart_init()
{
    pd1_5 = 0;      /* Pd1_5/RXDo = input */
    uosr = 0x05;    /* TXDo pin : P1_4 */
                    /* RXDo pin : P1_5 */
                    /* CLKo pin : not used */
    sotic = 0x04;
                    /* UARTo transmit interrupt level 4 */
    soric = 0x06; /* UARTo receive interrupt level 6 */
    te_uoc1 = 0;   /* Transmit enable : disabled */
    re_uoc1 = 0;   /* Receive enable : disabled */
    uomr = 0x65;  /* Serial I/O mode : UART mode,
                    /*transfer data 8 bits long */
                    /* Internal/external clock: Internal clock */
                    /* Stop bit length: One stop bit */
                    /* Odd/even parity: Even */
                    /* Parity enable: Parity enabled */
    uoco = 0x00;   /* BRG count source: F1 */
                    /* Data output: TXDo pin set to CMOS output */
                    /* Transfer format: LSB first */
```

```
UOC1 = 0x00;      /* Transmit enable: disabled */
                  /* Receive enable: disabled */
                  /* UARTo transmit interrupt source : */
                  /*Transmission completed (TI= 1) */
                  /* UARTo continuous receive mode : */
                  /*Continuous receive mode disabled */

uobrg = 130-1;
                  /* 20MHz * 1/1 * 1/(129+1) * 1/16 = 9615bps */

//  te_uoc1 = 1;      /* Transmit enable : enabled */
//  re_uoc1 = 1;      /* Receive enable : enabled */
}
```

[3.Communication protocol]

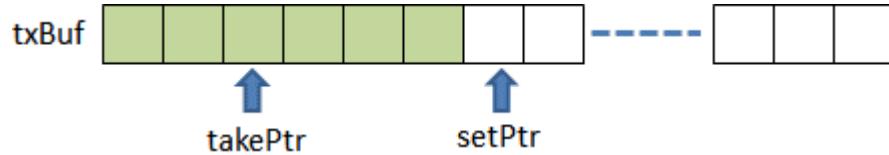
(1)For instance, transmit the sensor data of Line Tracer to PC with the following format.

@	sensor0	sensor1	sensor2	sensor3	FCS	*	CR
---	---------	---------	---------	---------	-----	---	----

[4.Uarto.cpp]

(1) Prepare a file named “Uarto.cpp”.

(2) And, describe variables for transmission.



```
#include <string.h>
#include "LineTracer.h"

#define TXBFSZ      40
static char txBuf[TXBFSZ] ; //Send buffer
static char *takePtr ;     //Pointer
static char *setPtr ;     // Pointer
static bool txBusy ;      //Busy flag
```

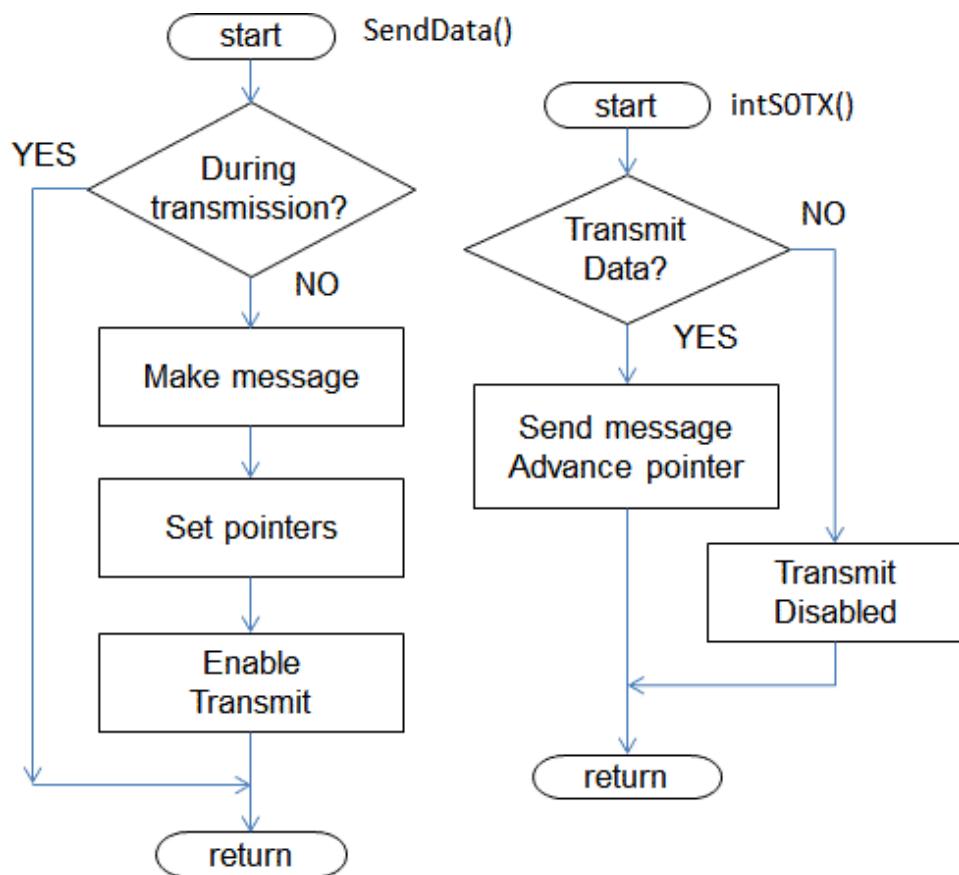
```

static void MakeText(void);

//Initialize
void InitUarto()
{
    takePtr = setPtr = txBuf ;
    txBusy = false ;
}

```

(3) Flow chart



(4) Transmission

```
voidSendData()
{
    if ( txBusy == true )
        return ;
    if ( txept_uoco == 0 )    //On sending
        return ;
    MakeText();
    int n = strlen( txBuf ) ;
    txBusy = true ;
    takePtr = txBuf ;
    setPtr = &txBuf[n] ;
    uotb = *takePtr++ ;
    te_uoc1 = 1;      /* Transmit enable : enabled */
}
```

(5) Subroutine

```
//Convert to 2-byte hex code
void itoh2( unsigned short n , char *s )
{
    unsigned char b ;

    for ( int m = 2 ; --m >= 0 ; ){
        b = n & 0x000f ;
        n >>= 4 ;
        if ( b <= 9 )
            *( s + m ) = '0' + b ;
        else
            *( s + m ) = 'A' + b - 10 ;
    }
    *( s + 2 ) = 0 ;
}
```

```
// Convert to 4-byte hex code
void itoh4( unsigned short n , char *s )
{
    unsigned char b ;

    for ( int m = 4 ; --m >= 0 ; ){
        b = n & 0x000f ;
        n >>= 4 ;
        if ( b <= 9 )
            *( s + m ) = '0' + b ;
        else
            *( s + m ) = 'A' + b - 10 ;
    }

    *( s + 4 ) = 0 ;
}
```

```
//Make text
static void MakeText()
{
    int sensor[4];
    GetValues(sensor);
    char *t = txBuf ;
    strcpy( t,"@" );
    t++ ;
    int n = sensor[0];
    itoh4( n, t );
    t += 4 ;
    *t++ = ' ' ;
    n = sensor[1];
    itoh4( n, t );
    t += 4 ;
    *t++ = ' ' ;
    n = sensor[2];
    itoh4( n, t );
    t += 4 ;
    *t++ = ' ' ;
    n = sensor[3];
    itoh4( n, t );
    t += 4 ;
    *t++ = ' ' ;
```

```

char *s = txBuf ;
char c = *s++ ;
while ( s < t )
    c ^= *s++ ;      /*EOR*/
itoH2( c, t );
t += 2 ;
*t++ = '*' ;      //terminator
*t++ = 'Yr' ;
*t = 0 ;
}

```

(6) Interrupt

```

#pragma interrupt intSoTX(vect=17)
void intSoTX(void)
{
    if ( takePtr != setPtr )
        uotb = *takePtr++ ;
    else {
        txBusy = false ;
        te_uoc1 = 0;      /* Transmit enable : disabled */
    }
}

```

```
#pragma interrupt intSoRX(vect=18)
void intSoRX(void)
{
}
```

(7) Periodically send

In order to periodically send the data to PC, use the timer.

At first, make the definition.

Add code in “LineTracer.h” file.

```
//For timer
enum {
    TM00,      //SCI timer
    TM01,      //
    TMRNUM
};
```

[5.Timer.cpp]

- (1) Prepare new file named “Timer.cpp”.
Insert code in the file.

```
#include "LineTracer.h"

//Timer table
// 100msec
const unsigned short timeBase[TMRNUM] = {
    5,      //TM00 SCI  0.5sec
    10,     //TM01
};

struct timer {
    unsigned short stat ;      //State
    unsigned short rest ;     //Time left
    unsigned short time ;     //Set time
};
static struct timer timeTbl[TMRNUM] ;
static int tick ;
```

```

void InitTimer()
{
    struct timer *sp = timeTbl ;
    for ( int n = 0 ; n < TMRNUM ; n++ ) {
        sp->stat = OFF ;
        sp->rest = 0 ;
        sp->time = timeBase[n] ;
        sp++ ;
    }
    tick = 0 ;
}

```

- (2) Please move the timer RA interrupt using the trigger of the A/D to “Timer.cpp” file.

In “Analog.cpp” file

```

//#pragma interrupt intTRAIC(vect=22)
//void intTRAIC(void)
//{
//    adst = 1 ;    //ADC Start
//}

```

(3) In “Timer.cpp” file

```
#pragma interruptintTRAIC(vec=22)
void intTRAIC(void)
{
    adst = 1; //ADC Start
}
```

Add code in “intTRAIC” function.

(4) Interrupt //10ms

```
#pragma interruptintTRAIC(vec=22)
void intTRAIC(void)
{
    adst = 1; //ADC Start
    if ( ++tick >= 10 ){ //100ms
        tick = 0;
        struct timer *sp = timeTbl;
        for ( int n = 0 ; n < TMRNUM ; n++ ) {
            if ( sp->stat == ON ) {
                if ( sp->rest )
                    sp->rest--;
                if ( sp->rest == 0 )
                    sp->stat = UP;
            }
            sp++;
        }
    }
}
```

(5) Let's provide a method to use the timer.

```
// Start the specified timer
void OnTimer( int n )
{
    timeTbl[n].rest = timeTbl[n].time ;
    timeTbl[n].stat = ON ;
}
// Stop the specified timer
void OffTimer( int n )
{
    timeTbl[n].stat = OFF ;
}
// Check the specified timer
int ChkTimer( int n )
{
    return timeTbl[n].stat ;
}
```

(6) Add code in “Prototype.h” file as follows.

```
void InitTimer(void);
void OnTimer( int n );
void OffTimer( int n );
int ChkTimer( int n );
```

(7) Using these, for example, periodically send the value of the sensor to the PC is as follows.

```
void main(void)
{
:
InitUarto();
InitTimer();
bool runFlag = false ;
OnTimer( TMoo );      // 0.5sec
//Loop of main
while(1)
{
    Analog();
    Input();
    if( runFlag == false )
    {
        if( IsStartPB() == true )
        {
            InitFuzzy();
            MotorStart();
            runFlag = true ;
        }
    }
}
```

```
if ( ChkTimer( TMoo ) == UP )
{
    OnTimer( TMoo );
    SendData();
}
else
{
    if ( IsStartPB() == true )
    {
        MotorStop();
        runFlag = false ;
    }
    else
    {
        MotorControl();
    }
}
}
```

Using the communication software in the PC, and you can see the received data.

```
@017F 01B4 0240 03B5 35*
@017B 01B2 023F 03AC 33*
@0179 01B1 023D 03AD 4E*
@017F 01B3 0240 03AD 40*
@017F 01B4 023F 03B0 41*
@017D 01B3 023F 03B3 47*
@017F 01B4 0240 0393 48*
@017F 01B4 0240 03B4 34*
@0179 01B1 023E 03B3 3B*
@017B 01B2 023E 03B3 43*
@017F 01B3 0240 03B4 33*
@017E 01B3 023F 03AD 32*
@017A 01B1 023E 03B6 46*
@0180 01B4 0240 03AE 3F*
@017C 01B2 023E 03AD 36*
@017A 01B1 023D 03AE 37*
```

{Exercise 4}

Please complete the receiving part of the UART.

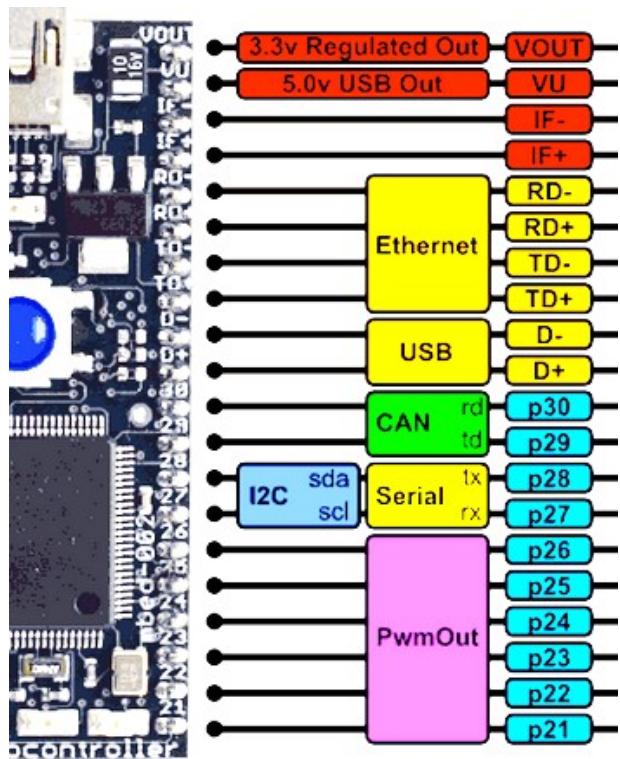
{Exercise 5}

From the PC, can you change the coefficient of Fuzzy ?

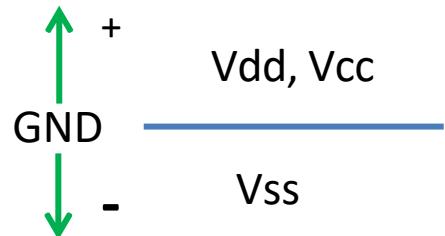
EMBED SYSTEM

LPC1768

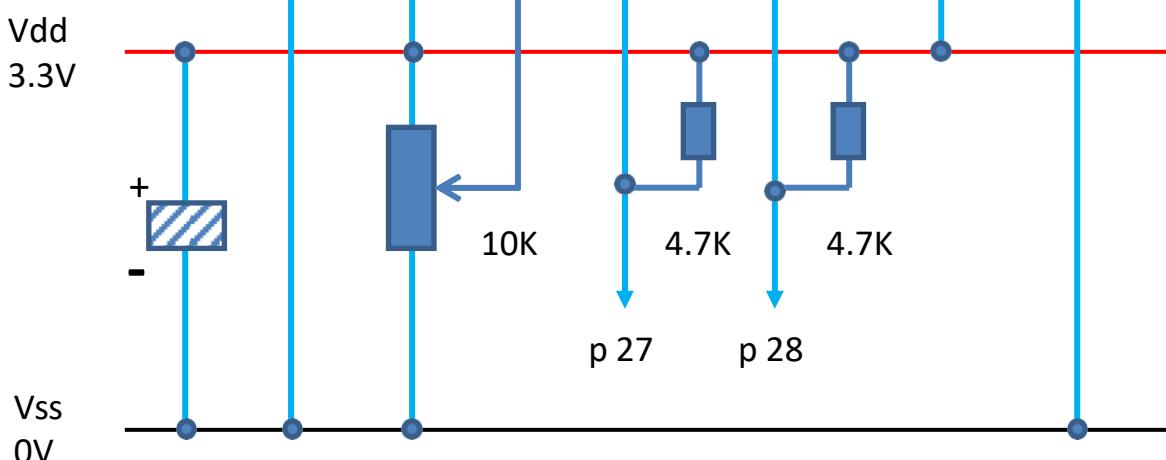
ACM1602NI-FLW-FBW



Pin No.	Symbol	Function
1	VSS	Ground
2	VDD	3.3V
3	V _O	LCD contrast adjust
4	SCL	SERIAL CLOCK INPUT
5	SDA	SERIAL DATA INPUT
6	BL+	Power Supply for BL+
7	BL-	Power Supply for BL-



1. Vss 2.Vdd 3.Vo 4.SCL 5.SDA 6.BL+ 7.BL-



CMOS

Vdd: Voltage Drain(+)

Vss: Voltage Source(-)

TTL

Vcc: Voltage Collector (+)

GND: GrouND

To be used to mbed-style

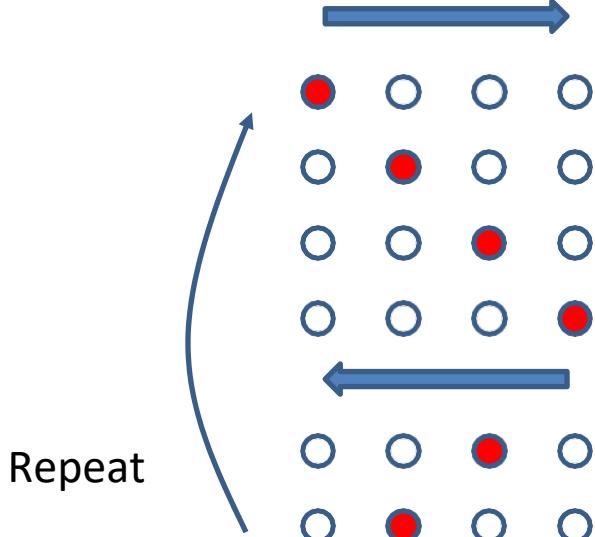
Example

```
include "mbed.h"
```

```
DigitalOut led1(LED1);  
DigitalOut led2(LED2);  
DigitalOut led3(LED3);  
DigitalOut led4(LED4);
```

```
void main()  
{  
    while(1) {  
        led1 = 1;  
        wait(0.2);  
        led1 = 0;  
        wait(0.2);  
  
    }  
}
```

Next challenge



We can exchange messages between this board and a PC.
Terminal emulator software is necessary for the PC side.
This is useful for doing debugging.

```
include "mbed.h"
```

```
DigitalOut led1(LED1);  
DigitalOut led2(LED2);  
DigitalOut led3(LED3);  
DigitalOut led4(LED4);
```

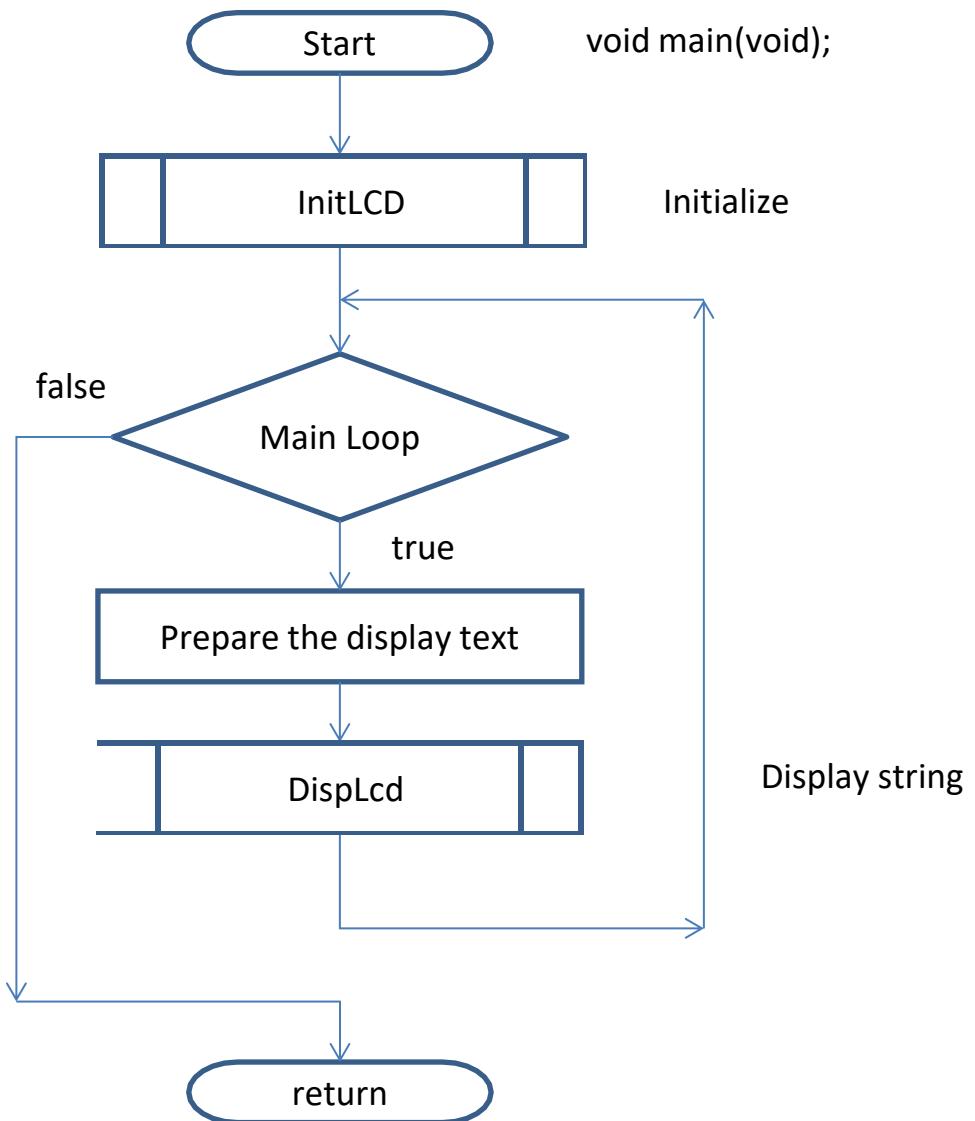
→ Serial pc(USBTX, USBRX);

```
int main()  
{  
    pc.printf("Echoes back to the screen anything you type\n");  
    while(1) {  
        if (pc.readable())  
            pc.putc(pc.getc());  
        led1 = 1;  
        wait(0.2);  
        led1 = 0;  
        wait(0.2);  
    }  
}
```

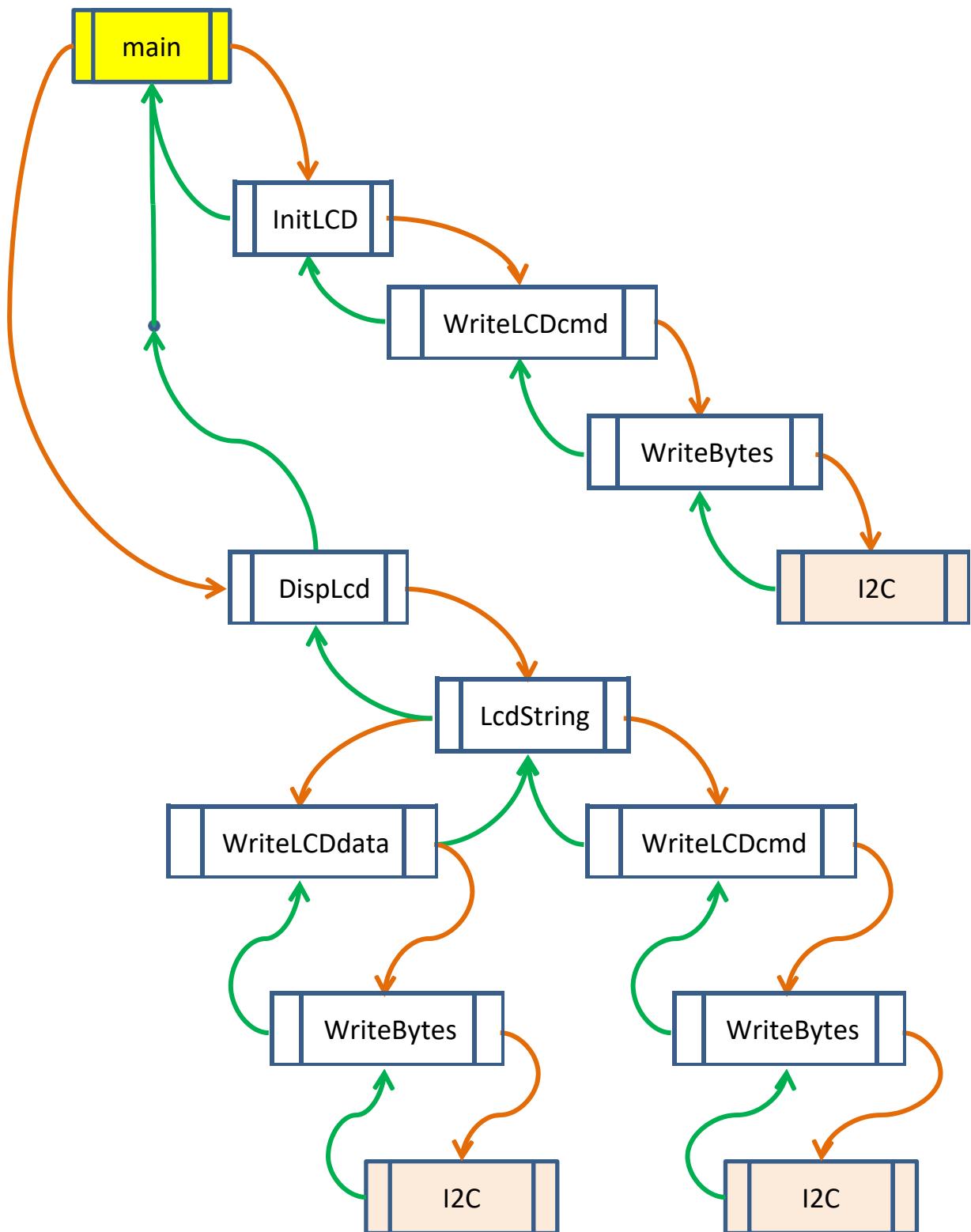
Question

- 1) Baud rate ?
- 2) Where is it described?

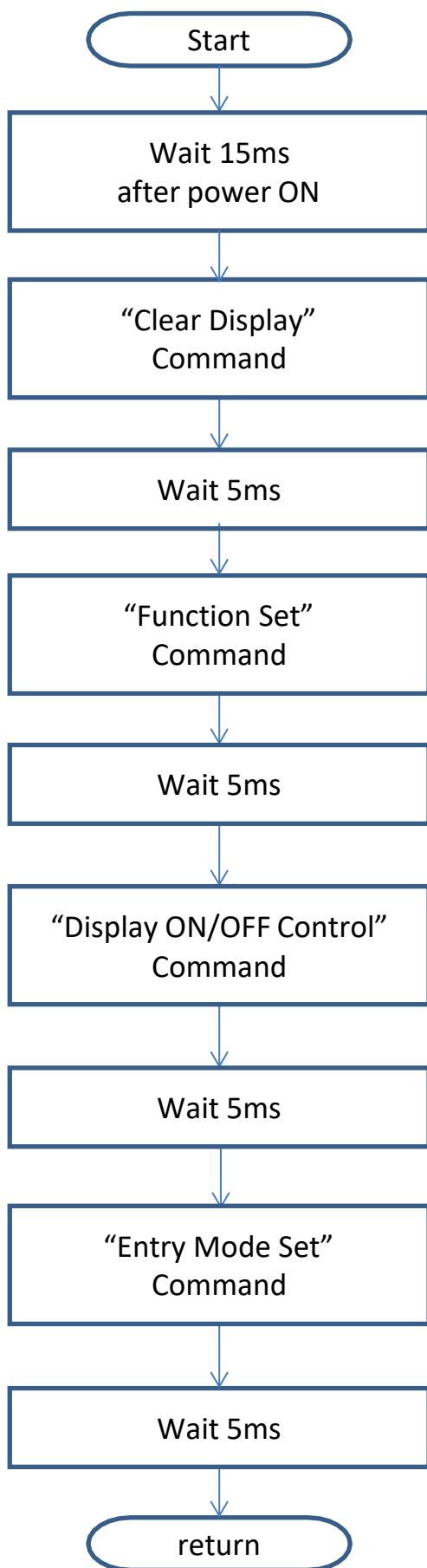
- 1) First, we have to initialize the LCD unit.
- 2) Thereafter, we send data to display to the LCD unit.



Call of the subprogram



```
void InitLCD();
```



ACM1602NI-FLW-FBW

Please refer to “11.0 DISPLAY INSTRUCTION TABLE” of the manual.

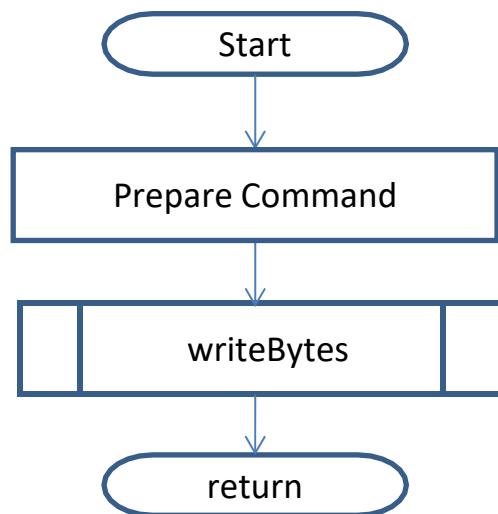
0x01;

0x38;
0b001x0000; DL
Interface Data length. 8bits=1/4bits=0
0b0010x000; N
Numbers of display line. 2-line=1/1-line=0
0b00100x00; F
Display font type. 5x10dots=1/5x8dots=0

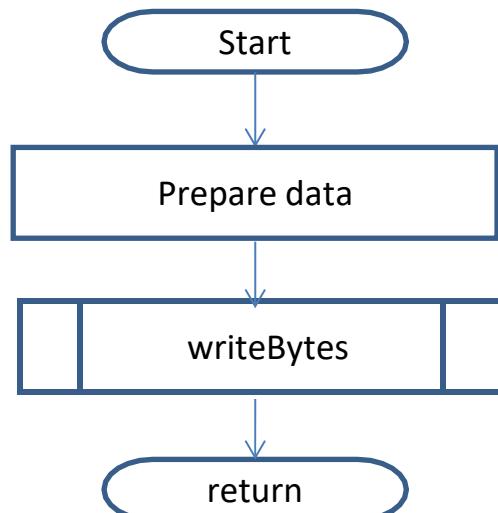
0x0c;
0b00001x00; D
Set display on=1/off=0
0b000010x0; C
Set cursor on=1/off=0
0b0000100x; B
Set blinking of cursor on=1/off=0

0x06;
0b000001x0; I/D
Assign cursor moving direction I=1,D=0
0b0000010x; S
Enable the shift of entire display

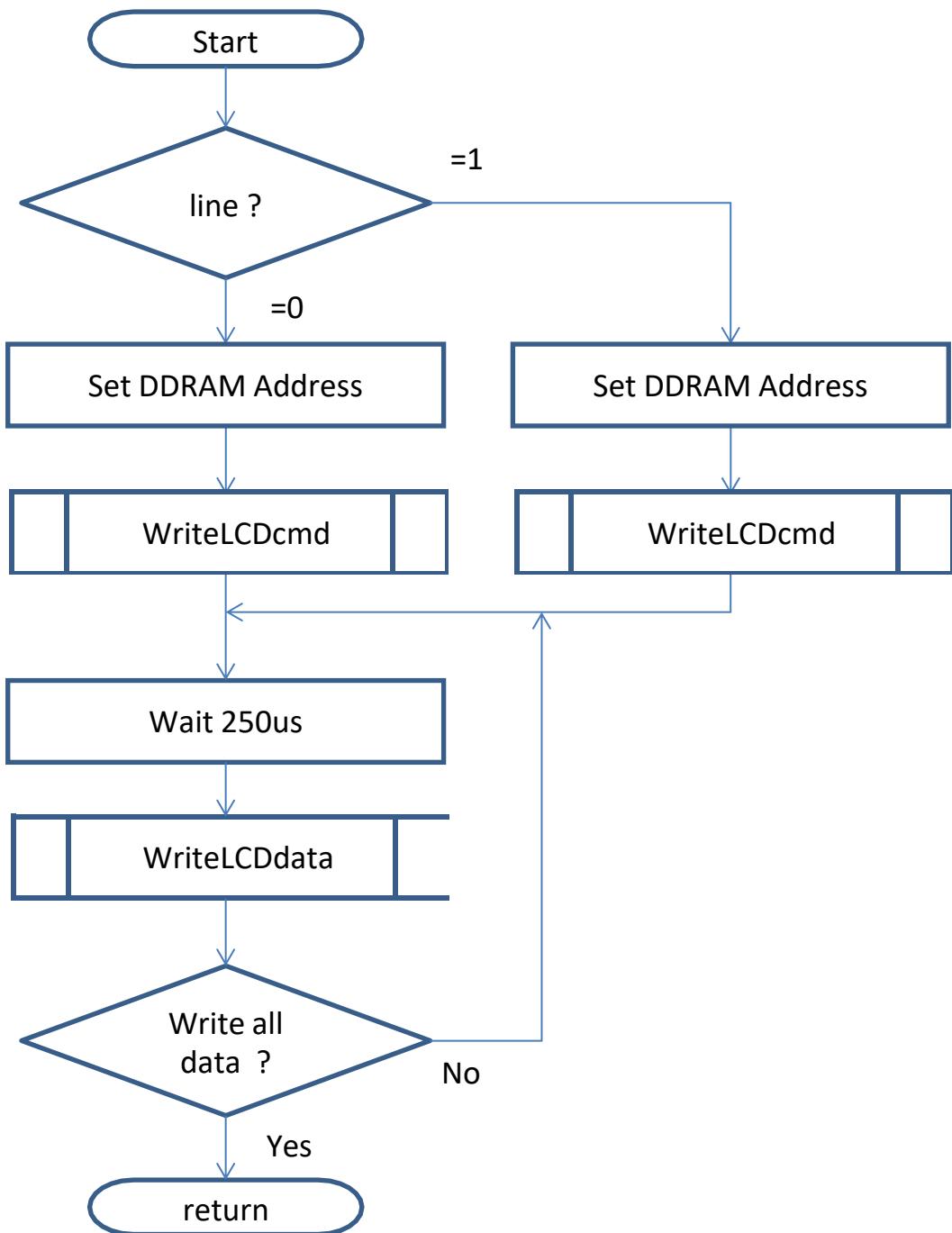
```
void WriteLCDcmd(unsigned char command);
```



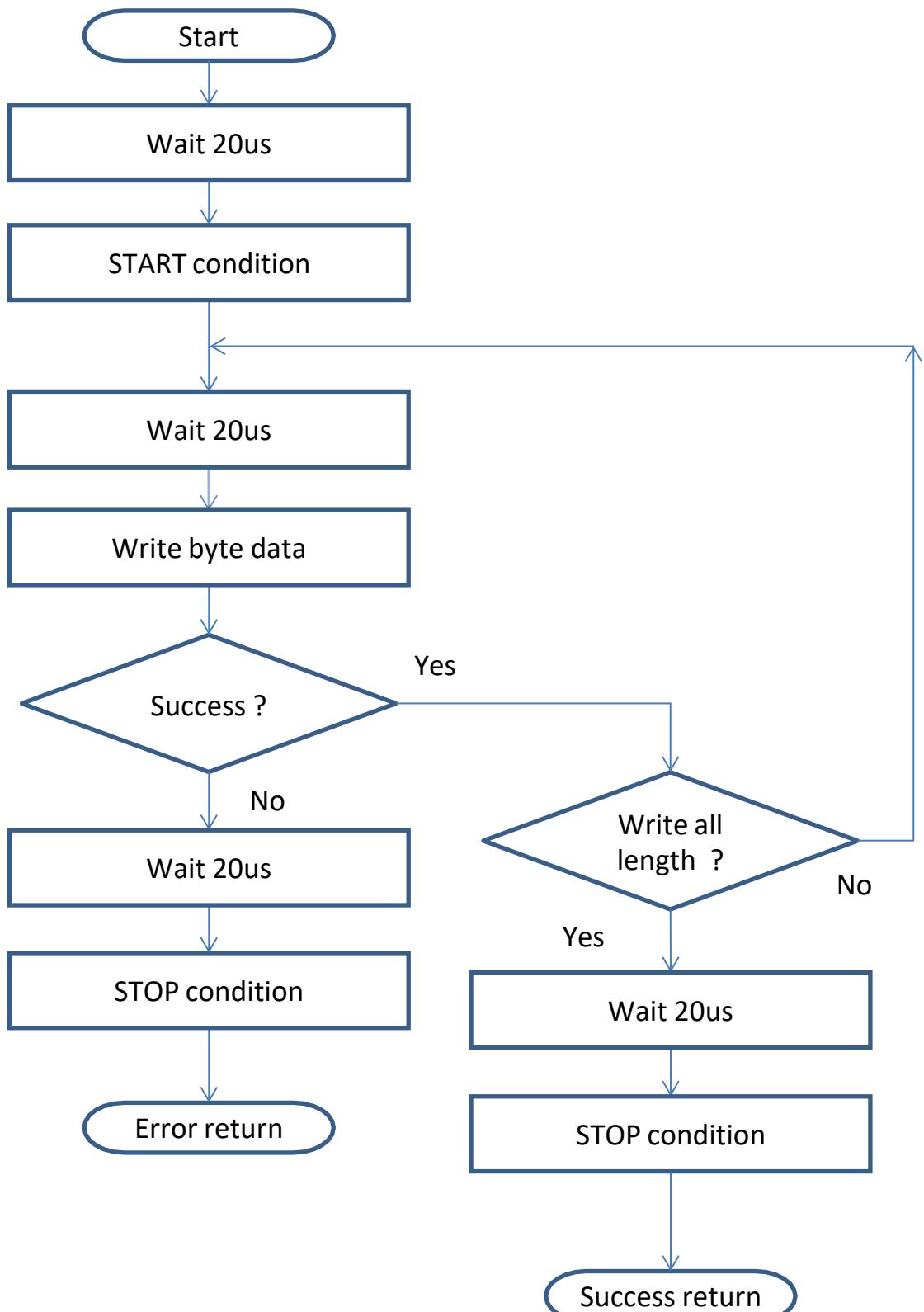
```
void WriteLCDdata(unsigned char data);
```



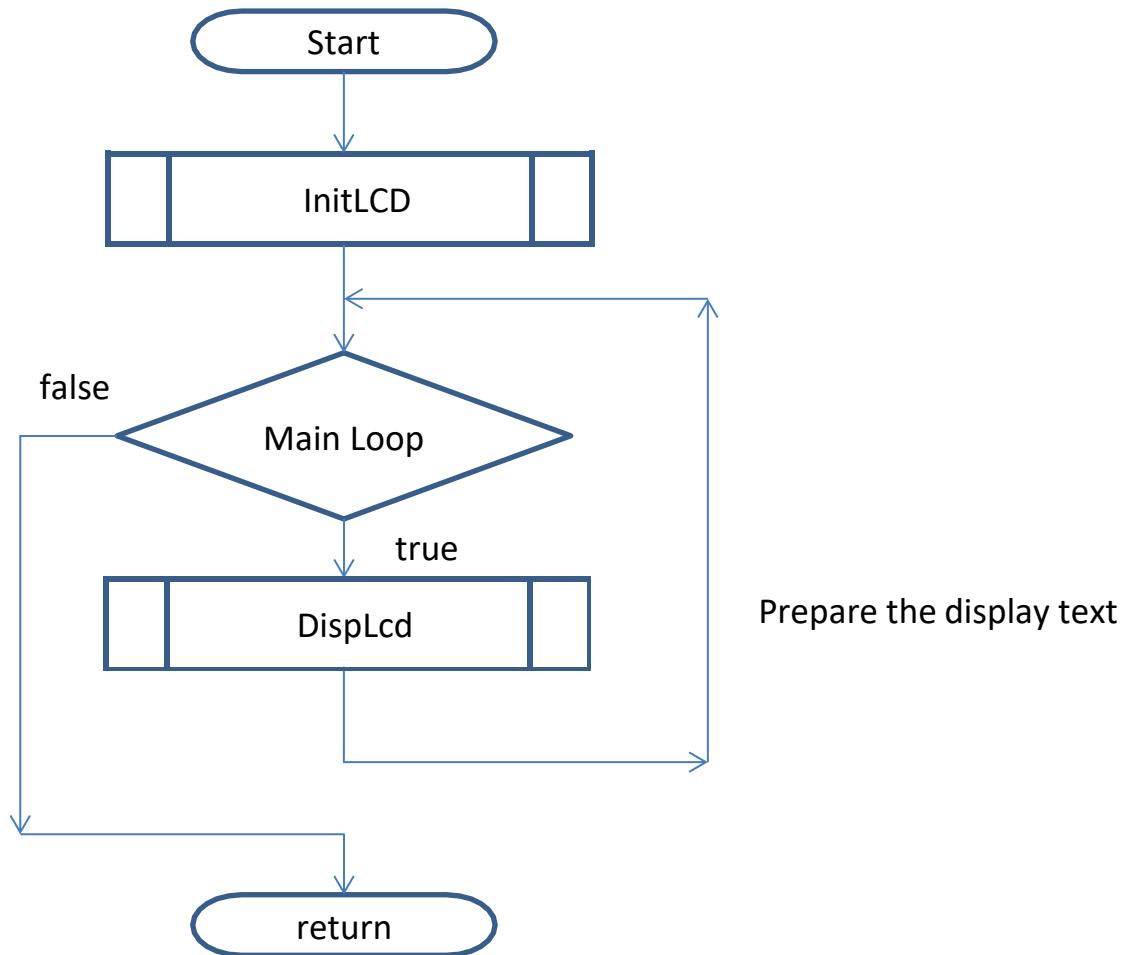
```
void LcdString( int line, char* s );
```



```
bool writeBytes(const char *data, int length);
```



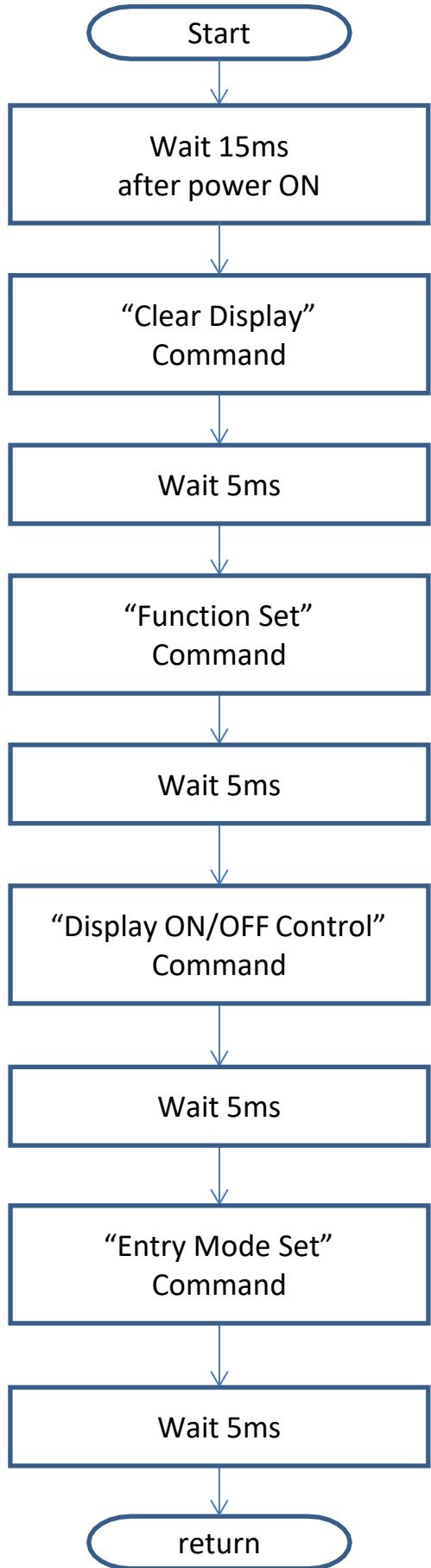
```
void main(void);
```



```
I2C i2c(p28, p27);
const int addr = 0xA0;
```

```
void main()
{
    int ctr = 0;
    InitLCD();
    while (1) {
        DispLCD( ctr );
        if ( ++ctr >= 1000 )
            ctr = 0;
        wait( 0.25 );
        led1 = !led1;
    }
}
```

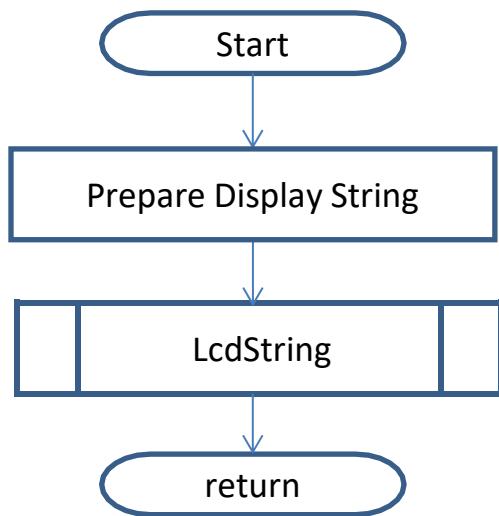
```
void InitLCD();
```



ACM1602NI-FLW-FBW

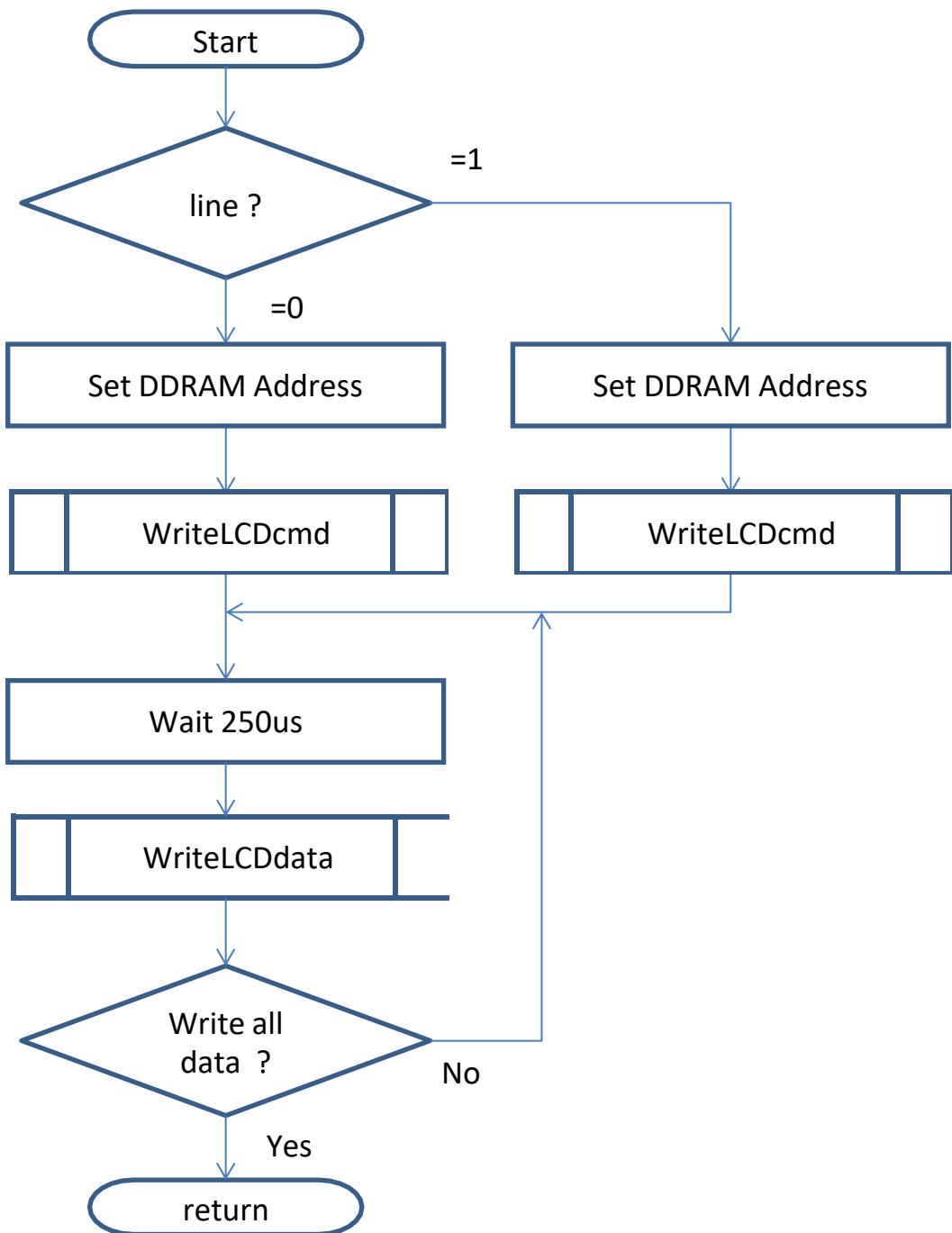
```
void InitLcd()
{
    wait_ms( 15 ); //15ms
    WriteLCDcmd( 0x01 );// Clear Diplay
    wait_ms( 5 ); //5ms
        //Function set DL=1,N=1,F=0 0011 1000
        //b5 = 1
        // 0b001x000; DL; Interface Data length.
        8bits/4bits
        // 0b0010x000; N; Numbers of display line. 2-
        line/1-line
        // 0b00100x00; F; Display font type.
        5x10dots/5x8dots
    WriteLCDcmd( 0x38 );
    wait_ms( 5 ); //5ms
        //Display ON/OFF Control
        //b3 = 1
        // 0b00001x00; D; Set display on/off.
        // 0b000010x0; C; Set curthor on/off.
        // 0b0000100x; B; Set blinking of curthor
        on/off.
    WriteLCDcmd( 0x0c );
    wait_ms( 5 ); //5ms
        // Entry Mode Set
        //b2 = 1
        // 0b000001x0; I/D;Assign curthor moving
        direction.
        // 0b0000010x; I/D;Enable the shift of entire
        display.
    WriteLCDcmd( 0x06 );
    wait_ms( 5 ); //5ms
}
```

```
void DispLcd(int n);
```



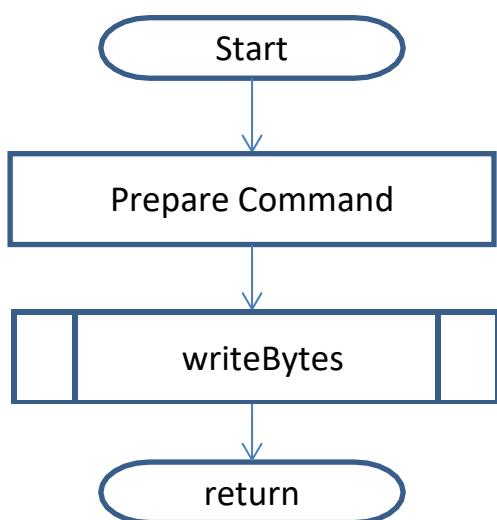
```
void DispLcd( int n )
{
    char buff[18];
    sprintf( buff, "counter=% 4d", n);
    LcdString( 0, buff );
    led2 = !led2;
}
```

```
void LcdString( int line, char* s );
```



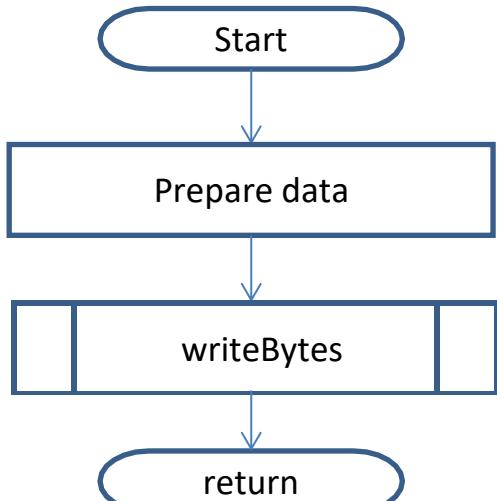
```
void LcdString( int line, char* s )
{
    char c ;
    if ( line == 0 ) {
        //Set DDRAM Address//first line
        WriteLCDcmd(0x80);
    } else {
        //Set DDRAM Address
        WriteLCDcmd(0x80 | 0x40); //second line 0xC0
    }
    while (( c = *s++ ) > 0 ) {
        wait_ms( 5 ); //5ms
        //Write Data to RAM
        WriteLCDdata(c);
    }
    wait_ms( 5 ); //5ms
}
```

```
void WriteLCDcmd(unsigned char command);
```



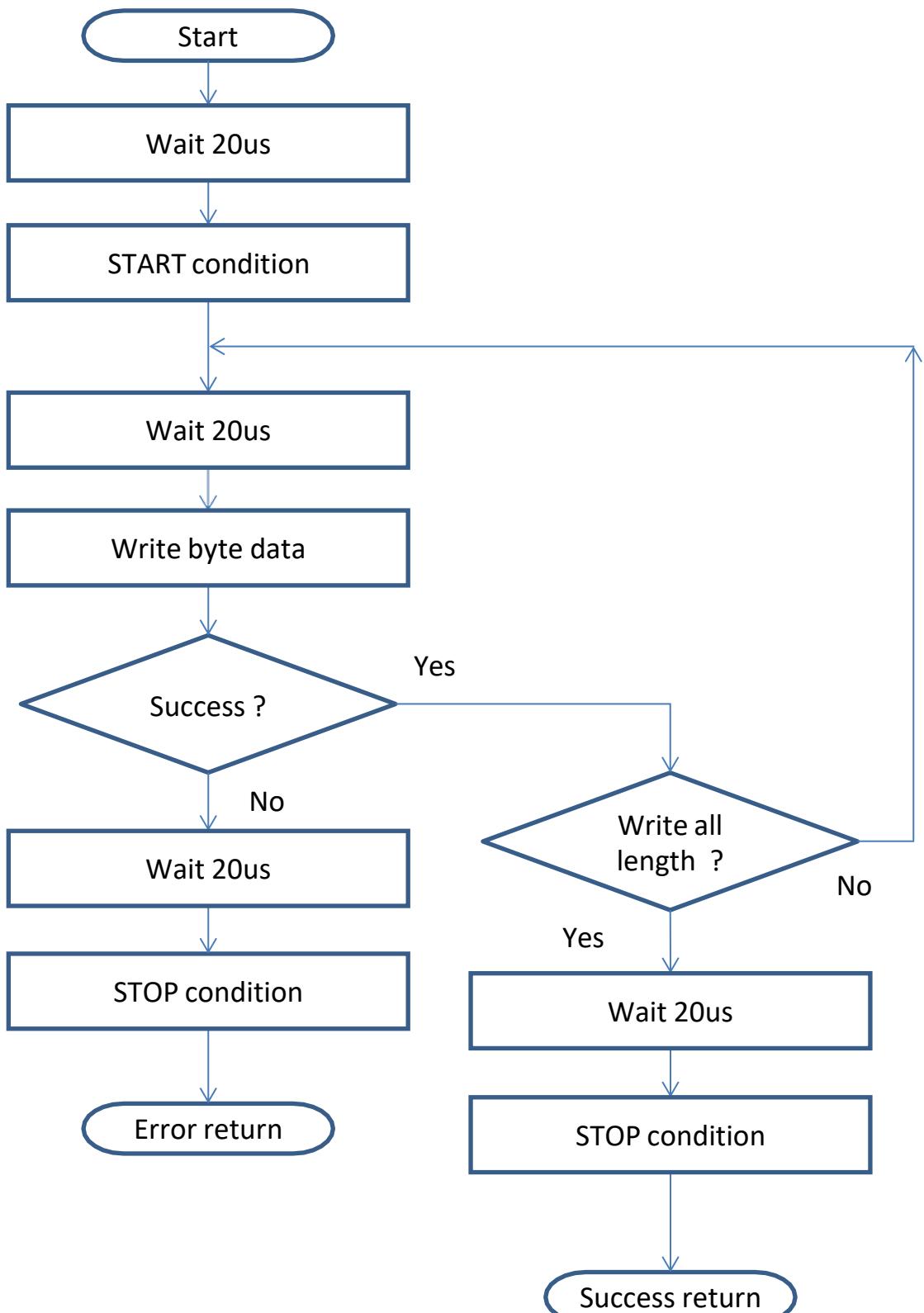
```
void WriteLCDcmd(unsigned char  
command)  
{  
    char cmd[3];  
    cmd[0] = addr;  
    cmd[1] = 0x00;  
    cmd[2] = command;  
    WriteBytes(cmd, 3);  
}
```

```
void WriteLCDdata(unsigned char data);
```



```
void WriteLCDdata(unsigned char data)  
{  
    char cmd[3];  
    cmd[0] = addr;  
    cmd[1] = 0x80;  
    cmd[2] = data;  
    WriteBytes(cmd, 3);  
}
```

```
bool writeBytes(const char *data, int length);
```



Program Workspace

- mbed
 - Classes
 - AnalogIn
 - AnalogOut
 - BusInOut
 - BusIn
 - BusOut
 - CANMessage
 - CAN
 - DigitalInOut
 - DigitalIn
 - DigitalOut
 - DirHandle
 - Ethernet
 - FileHandle
 - FileSystemL
 - FunctionPoint
 - I2CSlave
 - I2C**
 - InterruptIn

I2C Class Reference

```
#include <I2C.h>
```

Public Member Functions

- I2C (PinName sda, PinName scl)**
Create an **I2C** Master interface, connected to the specified pins.
- void frequency (int hz)**
Set the frequency of the **I2C** interface.
- int read (int address, char *data, int length, bool repeated=false)**
Read from an **I2C** slave.
- int read (int ack)**
Read a single byte from the **I2C** bus.
- int write (int address, const char *data, int length, bool repeated=false)**

Compile output for program: I2C_LCD1602

Description
—

void start (void)
Creates a start condition on the **I2C** bus.

void stop (void)
Creates a stop condition on the **I2C** bus.

int write (int data)
Write single byte out on the **I2C** bus.

Parameters:
data data to write out on bus

Returns:
'1' if an ACK was received, '0' otherwise

```
int write( int address,  
          const char * data,  
          int length,  
          bool repeated = false  
    )
```

Write to an **I2C** slave.

Performs a complete write transaction. The bottom bit of the address is forced to 0 to indicate a write.

Parameters:

address 8-bit **I2C** slave address [addr | 0]
data Pointer to the byte-array data to send
length Number of bytes to send
repeated Repeated start, true - do not send stop at end

Returns:

0 on success (ack), non-0 on failure (nack)

```
bool WriteBytes(const char *data, int length) {  
    wait_us(20);  
    i2c.start();  
    for (int n = 0; n < length; n++) {  
        wait_us(20);  
        if (i2c.write(data[n]) != 1) { // '1' if an ACK was received, '0' otherwise  
            wait_us(20);  
            i2c.stop();  
            led4 = 1;  
            return false;  
        }  
    }  
    wait_us(20);  
    i2c.stop();  
    led3 = !led3;  
    return true;  
}
```

Search mbed.org...

[Handbook](#) » [Windows serial configuration](#)

Windows serial configuration

The mbed serial port works by default on Mac and Linux, but Windows needs a driver. These instructions explain how to setup the mbed Microcontroller to use the USB serial port on Windows.

1. Download the mbed Windows serial port driver

Download the installer to your PC, e.g. your desktop.

[Download latest driver](#)

2. Run the installer

Troubleshooting

If you have multiple mbed microcontrollers, but the serial port only appears for one of them:

- Make sure you run the installer for every mbed; windows loads the driver based on the serial number, so it needs to be run for each mbed you use

If the installer fails because "No mbed Microcontrollers were found":

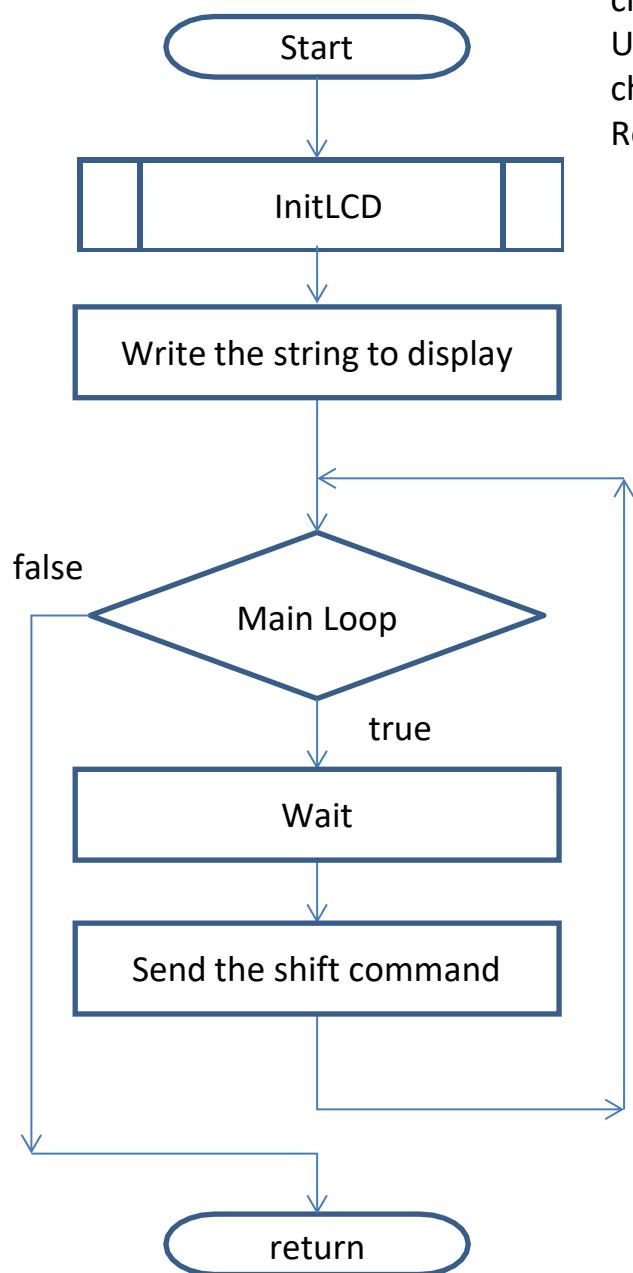
- Check your mbed Microcontroller is plugged in

If the installer reports the message "mbedWinSerial_nnnnn.exe is not a valid Win32 application":

- It is likely you are using Internet Explorer to download the installer file, which sometimes seems to only download part of the installer application for an unknown reason
- To solve this, download the installer with a different browser (Firefox, Chrome), and try again; this should solve the issue.

Scroll Sample

```
void main(void);
```



```
char message1[] = "<-Daffodil International  
University-->";//Max 40 char  
char message2[] = "Robotics Club Robotics Club  
Robotics ";
```

```
int main() {  
    int ctr = 0;  
    bool flag = false;  
    int len = strlen(message1);  
    LcdString(0,message1);  
    LcdString(1,message2);  
    while (1) {  
        wait(0.5);  
        if ( ++ctr > len ) {  
            ctr = 0;  
            flag = (flag == true) ? false : true;  
            led1 = 0;  
            led4 = 0;  
        }  
        //Display Shift  
        if ( flag == false ) {  
            WriteLCDcmd(0x18);  
            led1 = !led1;  
        }  
        else {  
            WriteLCDcmd(0x1C);  
            led4 = !led4;  
        }  
    }  
}
```

DISPLAYTRONIC

XIAMEN ZETTLER ELECTRONICS CO., LTD.

SPECIFICATIONS FOR LIQUID CRYSTAL DISPLAY

CUSTOMER APPROVAL

CUSTOMER APPROVAL			
※ PART NO. : <u>ACM1602NI-FLW-FBW-M01(DISPLAYTRONIC) VER1.2</u>			
APPROVAL		COMPANY CHOP	
CUSTOMER COMMENTS			

DISPLAYTRONIC ENGINEERING APPROVAL

DESIGNED BY	CHECKED BY	APPROVED BY
SDF	JFM	GZH

REVISION RECORD

REVISION	REVISION DATE	PAGE	CONTENTS
VER1.0	31/10-2011		FIRST ISSUE
VER1.1	02/02-2012	19	ADD DDRAM ADDRESS
VER1.2	21/03-2012	3&7	1、3.2.1 ELECTRICAL-OPTICAL CHARACTERISTICS OF LED BACKLIGHT: CHANGED PEAK WAVE LENGTH TO CHROMA COORDINATE 2、CHANGED PIN ASSIGNMENT——VDD: FROM 5.0V TO 3.3V

※ CONTENTS

- 1.0 GENERAL SPECS
- 2.0 ABSOLUTE MAXIMUM RATINGS
- 3.0 ELECTRICAL CHARACTERISTICS
- 4.0 OPTICAL CHARACTERISTICS
- 5.0 BLOCK DIAGRAM AND POWER SUPPLY DIAGRAM
- 6.0 PIN ASSIGNMENT
- 7.0 I2C TIMING CHARACTERISTICS
- 8.0 THE REFERENCED CODE
- 9.0 MECHANICAL DIAGRAM
- 10.0 RELIABILITY TEST
- 11.0 DISPLAY INSTRUCTION TABLE
- 12.0 STANDARD CHARACTER PATTERNS
- 13.0 PRECAUTION FOR USING LCM

1.0 GENERAL SPECS

1. Display Format	16*2 Character
2. Power Supply	3.3V(Single power supply with integrated DC-DC,adjustable Vop)
3. Module outline dimension	85.0mm(W) x 33.5mm(H) x max 13.5mm(D)
1. Viewing Area(W*H)	64.5mm(W) x 16.4mm(H)
2. Dot Size (W*H)	0.56mm(W) x 0.61mm(H)
3. Dot Pitch (W*H)	0.61mm(W) x 0.66mm(H)
4. Character Size (W*H)	3.00mm(W) x 5.23mm(H)
5. Character Pitch (W*H)	3.51mm(W) x 5.75mm(H)
6. Viewing Direction	6:00 O'Clock
7. Driving Method	1/16Duty,1/5Bias
8. Control IC	PIC16F689 and ST7066U-0A
9. Display Mode	FSTN /Positive/Transflective
10. Backlight	White LED/Side
11. Operating temperature	-20°C ~ 70°C
12. Storage temperature	-30°C ~ 80°C
13. Interface	I2C
14. ROHS	ROHS compliant

2.0 ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Min	Typ	Max	Unit
Operating temperature	Top	-20	--	70	°C
Storage temperature	Tst	-30	--	80	°C
Input voltage	Vin	Vss-0.3	--	Vdd+0.3	V
Supply voltage for logic	Vdd- Vss	2.7	--	5.5	V
Supply voltage for LCD driving	Vdd- V0	3.0	--	8.0	V

3.0 ELECTRICAL CHARACTERISTICS

3.1 Electrical Characteristics Of LCM

Item	Symbol	Condition	Min	Typ	Max	Unit
Power Supply Voltage	Vdd	25°C	3.1	3.3	3.5	V
Power Supply Current	Idd	Vdd=3.3V, fosc=270kHz	--	3.5	5	mA
Input voltage (high)	Vih	Pins:(E,RS,R/ W,DB0-DB7) VDD=3.3V	0.7Vdd	--	Vdd	V
Input voltage (low)	Vil		-0.3	--	0.6	V
Recommended Driving Voltage	LC Vdd -V0	-20°C	4.6	4.8	5.0	V
		25°C	4.3	4.5	4.7	
		70°C	3.9	4.1	4.3	

3.2 The Characteristics Of LED Backlight

3.2.1 Electrical-Optical Characteristics Of LED Backlight ($T_a=25^\circ C$)

Item	Symbol	Condition	Min	Typ	Max	Unit
Forward Voltage ⁽¹⁾	Vf	If=15mA	2.9	3.1	3.3	V
Reverse Voltage	Vr	-	--	--	5	V
Luminance ⁽²⁾	Lv	If=15mA	240	300	--	cd/m ²
Uniformity ⁽³⁾	Δ	(Lvmin/Lvmax)%	70%	--	--	-
Peak wave length	λ_p	-	--	--	--	nm
Chroma coordinate	x	If=15mA	0.26	--	0.30	um
	y	If=15mA	0.27	--	0.31	um
Lifetime ⁽⁴⁾	-	If=15mA	-	20000	-	Hours

NOTE:

(1) Forward voltage means voltage applied directly to the LED, please refer to the backlight diagram.

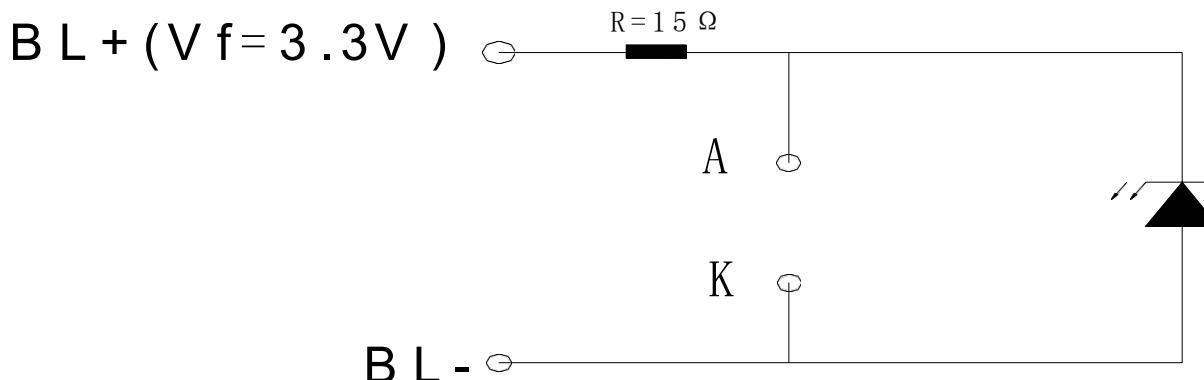
(2)The luminance is the average value of 5 points,The measurement instrument is BM-7 luminance colorimeter.The diameter of aperture is $\Phi 5\text{mm}$

(3) Luminance means the backlight brightness without LCD.

(4) Backlight lifetime means luminance value larger than half of the original after 20000 hours' continuous working.

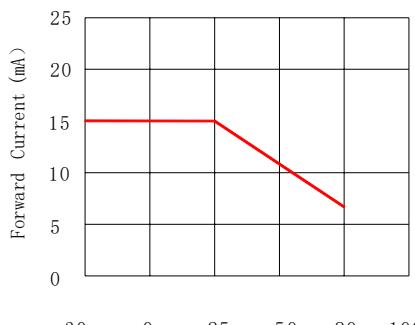
(5) Please apply the backlight current as the table recommend. If LCM surface luminance is acceptable, please apply the driving current as lower as possible. Any time, do not apply the driving current higher than 20mA.

3.2.2 Backlight Control Circuit FOR LCM (1*1=1 pcs LED)

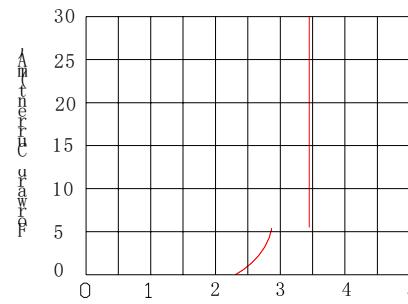


3.2.3 LED Characteristics Curves (for single led)

1. Forward current VS. Ambient Temperature



2. Forward current VS. Forward Voltage



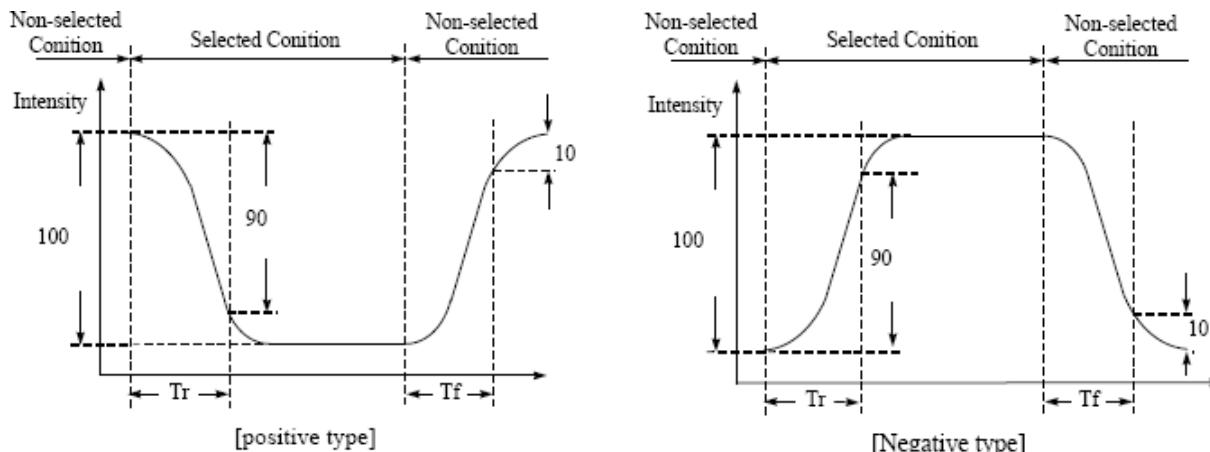
Ambient Temperature

Forward Voltage

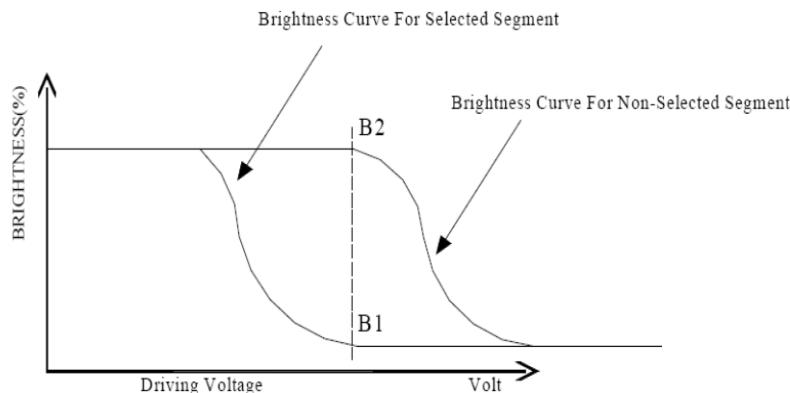
4.0 OPTICAL CHARACTERISTICS ($T_a=25^\circ C$)

Item	Symbol	Condition	Min	Typ	Max	Unit
Viewing angle (Left - right)	θ_2	$Cr \geq 2.0$	-35	-	35	deg
Viewing angle (Up-down)	θ_1	$Cr \geq 2.0$	-25	-	40	deg
Contrast Ratio	Cr	$\theta_1=0^\circ, \theta_2=0^\circ$	-	6	-	-
Response time (rise)	Tr	$\theta_1=0^\circ, \theta_2=0^\circ$	-	180	300	ms
Response time (fall)	Tf	$\theta_1=0^\circ, \theta_2=0^\circ$	-	150	250	ms

(1). Definition of Optical Response Time

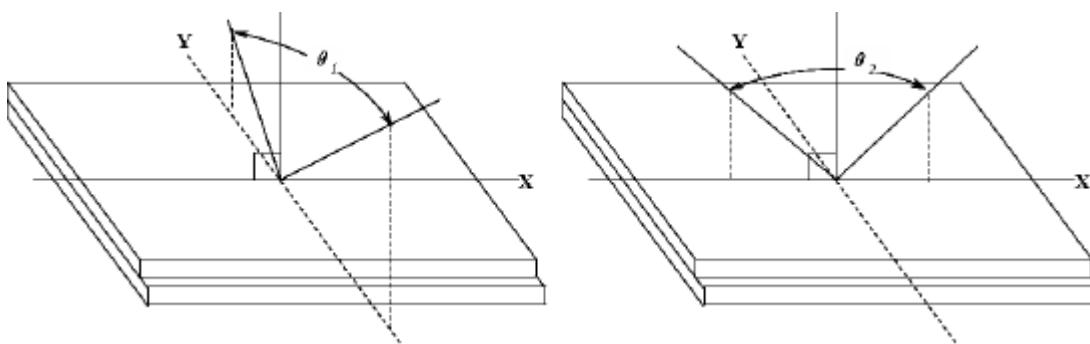


(2). Definition of Contrast Ratio



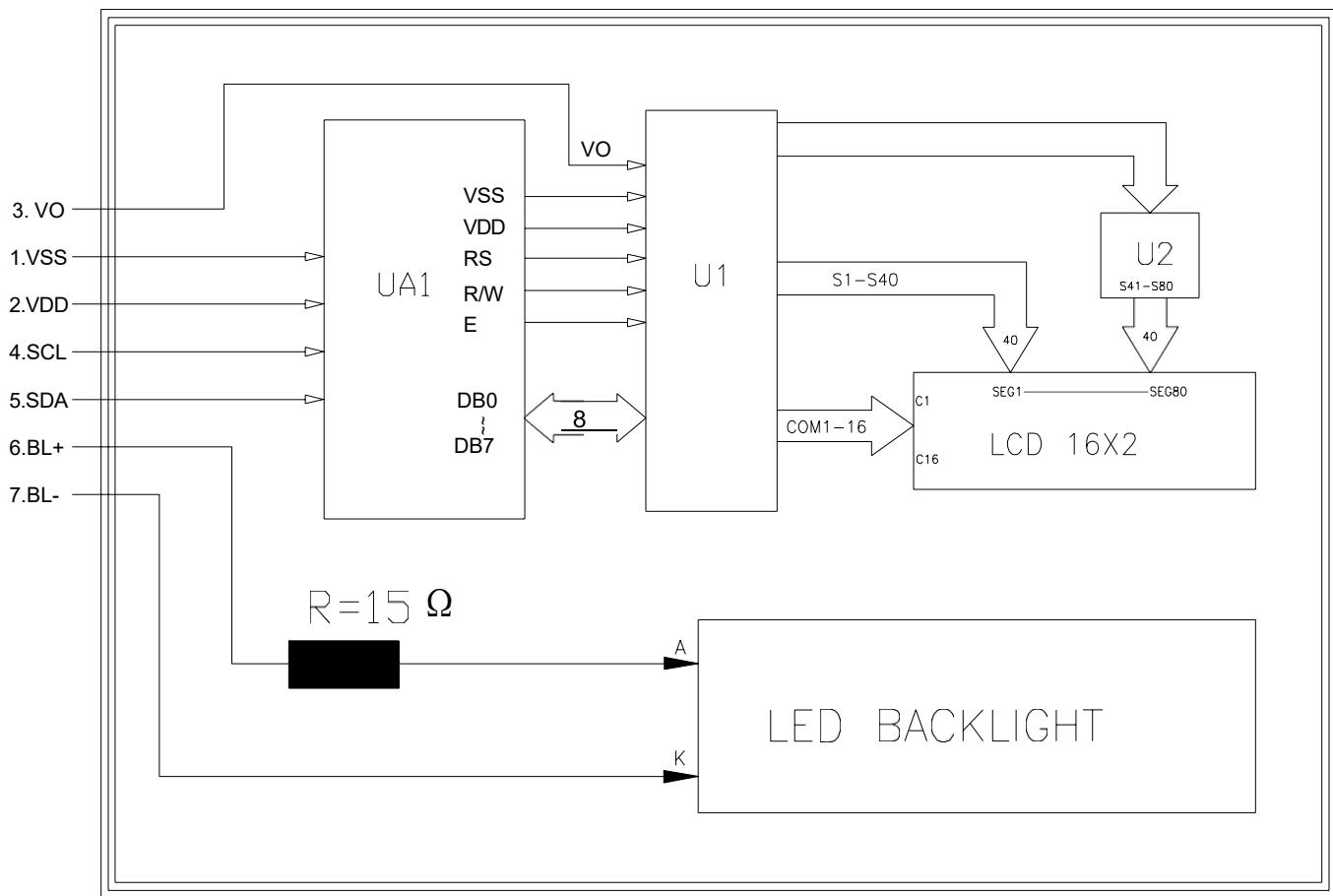
$$Cr = \frac{\text{Brightness of Non-selected Segment}(B2)}{\text{Brightness of selected Segment}(B1)}$$

(3). Definition of Viewing Angle θ_2 and θ_1

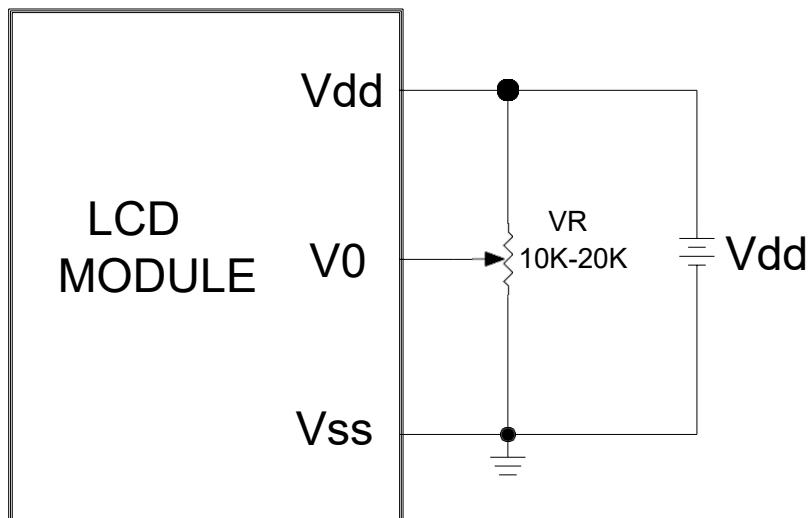


5.0 BLOCK DIAGRAM AND POWER SUPPLY DIAGRAM

5.0.1 BLOCK DIAGRAM



5.0.2 POWER SUPPLY DIAGRAM



6.0 PIN ASSIGNMENT

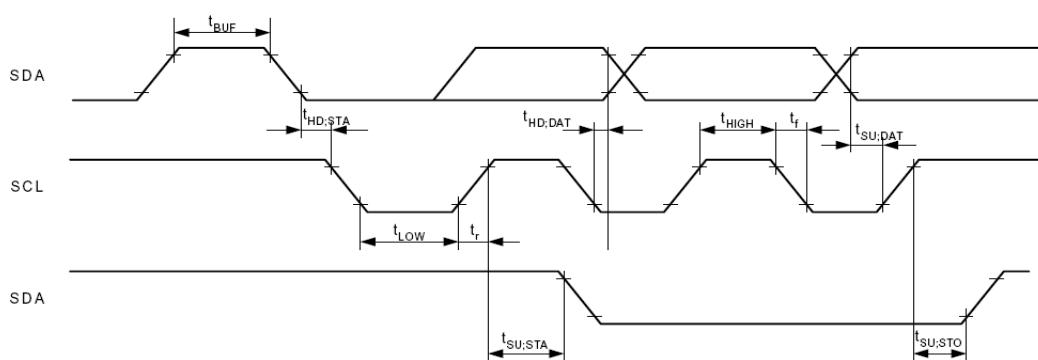
Pin No.	Symbol	Function
1	VSS	Ground
2	VDD	3.3V
3	V ₀	LCD contrast adjust
4	SCL	SERIAL CLOCK INPUT
5	SDA	SERIAL DATA INPUT
6	BL+	Power Supply for BL+
7	BL-	Power Supply for BL-

7.0 I²C TIMING CHARACTERISTICS

7.0.1 TIMING

$V_{DD} = 5V$, $V_{SS} = 0V$, $T_A = 25^\circ C$, unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
Timing characteristics: I ² C-bus interface (input capacitance $C_i = 10pF$)						
f_{SCL}	SCL clock frequency	–	–	100	KHz	
t_{LOW}	SCL clock low period	5.2	–	–	μs	
t_{HIGH}	SCL clock high period	2.4	–	–	μs	
$t_{SU;DAT}$	data set-up time	400	–	–	ns	
$t_{HD;DAT}$	data hold time	0	–	–	ns	
t_r	SCL, SDA rise time	–	–	400	ns	
t_f	SCL, SDA fall time	–	–	400	ns	
C_B	capacitive bus line load	–	–	400	pF	
$t_{SU;STA}$	set-up time for a repeated START condition	2.4	–	–	μs	
$t_{HD;STA}$	START condition hold time	2.4	–	–	μs	
$t_{SU;STO}$	set-up time for STOP condition	2.4	–	–	μs	
t_{SW}	tolerable spike width on bus	–	–	80	ns	



7.0.2 I2C interface:

It just only could write Data or Instruction to LCM by the IIC Interface.

It could not read Data or Instruction from LCM (except Acknowledge signal).

SCL: serial clock input

SDA: serial data input

Slaver address could only set to

The I2C interface send RAM data and executes the commands sent via the I2C Interface. It could send data bit to the RAM. The I2C Interface is two-line communication between different ICs or modules. The two lines are a Serial Data line (SDA) and a Serial Clock line (SCL). Data transfer may be initiated only when the bus is not busy.

BIT TRANSFER:

One data bit is transferred during each clock pulse. The data on the SDA line must remain stable during the HIGH period of the clock pulse because changes in the data line at this time will be interpreted as a control signal. Bit transfer is illustrated in Fig.1.

START AND STOP CONDITIONS:

Both data and clock lines remain HIGH when the bus is not busy. A HIGH-to-LOW transition of the data line, while the clock is HIGH is defined as the START condition (S). A LOW-to-HIGH transition of the data line while the clock is HIGH is defined as the STOP condition (P). The START and STOP conditions are illustrated in Fig.2.

SYSTEM CONFIGURATION:

The system configuration is illustrated in Fig.3.

- Transmitter: the device, which sends the data to the bus
- Master: the device, which initiates a transfer, generates clock signals and terminates a transfer
- Slave: the device addressed by a master
- Multi-Master: more than one master can attempt to control the bus at the same time without corrupting the message
 - Arbitration: procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted
 - Synchronization: procedure to synchronize the clock signals of two or more devices.

ACKNOWLEDGE:

Acknowledge is not Busy Flag in I2C interface.

Each byte of eight bits is followed by an acknowledge bit. The acknowledge bit is a HIGH signal put on the bus by the transmitter during which time the master generates an extra acknowledge related clock pulse. receiver which is addressed must generate an acknowledge after the reception of each byte. A master receiver must also generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges must pull-down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse (set-up and hold times must be taken into consideration). A master receiver must signal an end-of-data to the transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave. In this event the transmitter must leave the data line HIGH to enable the master to generate a STOP condition. Acknowledgement on the I2C Interface is illustrated in Fig.4.

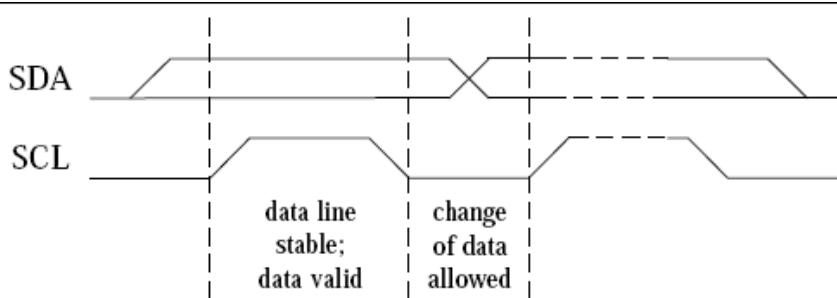


Figure 1. Bit transfer

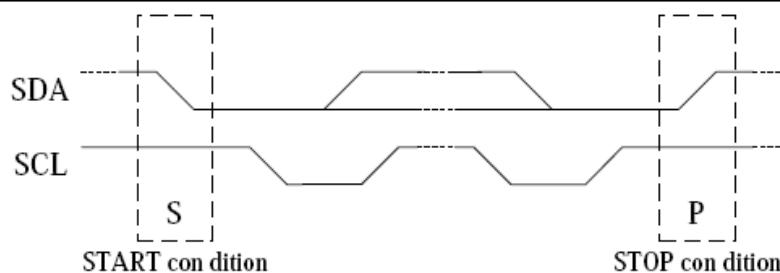


Figure 2. Definition of START and STOP conditions

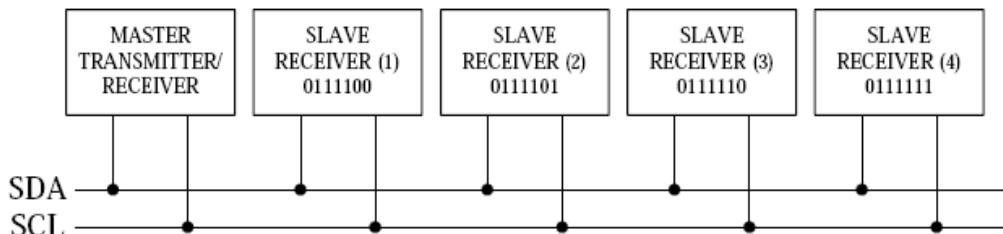


Figure 3. System configuration

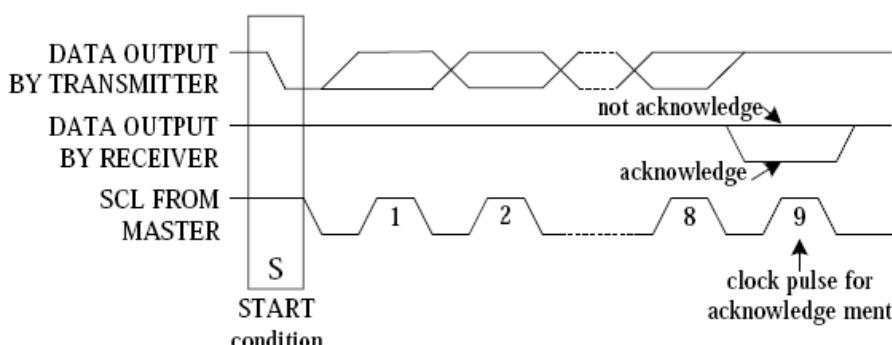


Figure 4. Acknowledgement on the 2-line Interface

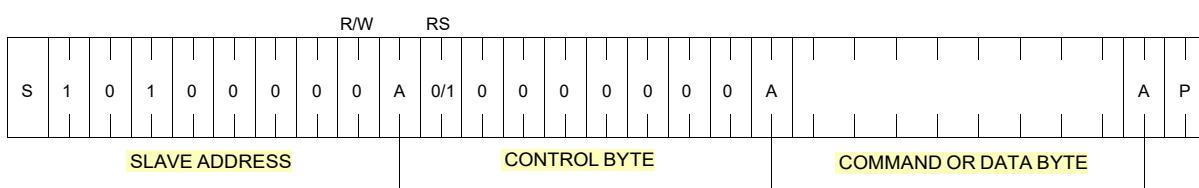
7.0.3 I2C Interface protocol:

The LCM supports command, data write addressed slaves on the bus.

Before any data is transmitted on the I2C Interface, the device, which should respond, is addressed first. Only one 7-bit slave addresses (1010000) is reserved for the LCM. The R/W is assigned to 0 for Write only. The I2C Interface protocol is illustrated in Fig.5.

The sequence is initiated with a START condition (S) from the I2C Interface master, which is followed by the slave address. All slaves with the corresponding address acknowledge in parallel, all the others will ignore the I2C Interface transfer. After acknowledgement, a control byte will be followed which defines RS bit. The state of the RS bit defines whether the data byte is interpreted as a command or as RAM data.

next following byte is command byte. When the control byte is 0x80 ,the next following byte is data byte. Only the addressed slave makes the acknowledgement after each byte. At the end of the transmission the I2C INTERFACE-bus master issues a STOP condition (P).



Acknowledgement from lcm

Acknowledgement from lcm

Acknowledgement from lcm

8.0 THE REFERENCED CODE

8.0.1 PICC REFERENCED CODE FOR PIC SERIES MCU

```

//PIC16F877A,11.0592MHZ,PICC
//*****
#include <pic.h>
#include <pic16f87x.h>
//*****
#define uchar unsigned char
//*****
#define sda RC4
#define sclk RC3
//*****
__CONFIG(HS & WDTDIS & PWRTDIS & BORDIS & LVPDIS & DUNPROT & UNPROTECT);
//*****
const uchar table[]=
{
    "This is program for "
    "The Master is:16f877"
    "i2c interface.      "
    "The Slave is:16f689 "
};

void long_delay(void)
{
    uchar i,j;
    for(i=0;i<25;i++)
        for(j=0;j<255;j++)
            ;
}

void short_delay(void)
{
    uchar i,j;
    for(i=0;i<1;i++)
        for(j=0;j<1;j++)
            ;
}

void start(void)
{
    sclk=1;
    sda=1;
    sda=0;
    sclk=0;
}

void stop(void)
{
    sclk=1;
    sda=0;
    sda=1;
    sclk=0;
}

void write_byte(byte)
{
    uchar i;
    for(i=8;i>0;i--)
    {

```

```

if((byte & 0x80)==0x80)
    sda=1;
else
    sda=0;
sclk=1;
sclk=0;
byte=byte<<1;
}
sda=1;
sclk=1;
TRISC4=1;
while(sda);
sclk=0;
TRISC4=0;
short_delay();           //wait for a moment,you can adjust it.
}
//*********************************************************************
void WtiteCommand(command)
{
start();
write_byte(0xa0);
write_byte(0x00);
write_byte(command);
stop();
}
//*********************************************************************
void WriteData(data)
{
start();
write_byte(0xa0);
write_byte(0x80);
write_byte(data);
stop();
}
//*********************************************************************
main()
{
uchar i;
long_delay();           //at least 15ms
long_delay();
POR=1;
BOR=1;
TRISC=0x00;
sda=1;
sclk=1;
WtiteCommand(0x01);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x38);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x0f);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x06);
long_delay();           //at least 5ms
long_delay();
while(1)
{
    WtiteCommand(0x80);
    long_delay();           //delay at least 5ms
    for(i=0;i<16;i++)
}

```

```

{
WriteData(table[i]);
long_delay();           //delay at least 5ms
}
WtiteCommand(0xC0);
long_delay();           //delay at least 5ms
for(i=0;i<16;i++)
{
WriteData(table[i]);
long_delay();           //delay at least 5ms
}

WtiteCommand(0x01);
}
}
//*****

```

8.0.2 KEILC REFERENCED CODE FOR 51 SERIES MCU

```

//AT89S52,12MHZ,KEIL
//*****
#include"at89x51.h"
//*****
#define sclk P2_0
#define sda P2_1
//*****
#define uchar unsigned char
//*****
uchar code table[]=
{
"This is program for "
"The Master is:89S52 "
"i2c interface.      "
"The Slave is:16f689 "
};
//*****
void long_delay(void)
{
uchar i,j;
for(i=0;i<25;i++)
for(j=0;j<255;j++)
;
}
//*****
void short_delay(void)
{
uchar i,j;
for(i=0;i<1;i++)
for(j=0;j<1;j++)
;
}
//*****
void start(void)
{
sclk=1;
sda=1;
sda=0;
sclk=0;
}
//*****
void stop(void)
{
sclk=1;

```

```

sda=0;
sda=1;
sclk=0;
}
//*****
void write_byte(byte)
{
uchar i;
for(i=8;i>0;i--)
{
if((byte & 0x80)==0x80)
sda=1;
else
sda=0;
sclk=1;
sclk=0;
byte=byte<<1;
}
sda=1;
sclk=1;
while(sda);
sclk=0;
short_delay();           //wait for a moment ,you can adjust it.
}
//*****
void WtiteCommand(command)
{
start();
write_byte(0xa0);
write_byte(0x00);
write_byte(command);
stop();
}
//*****
void WriteData(dat)
{
start();
write_byte(0xa0);
write_byte(0x80);
write_byte(dat);
stop();
}
//*****
main()
{
uchar i;
long_delay();           //at least 15ms
long_delay();
sda=1;
sclk=1;
WtiteCommand(0x01);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x38);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x0f);
long_delay();           //at least 5ms
long_delay();
WtiteCommand(0x06);
long_delay();           //at least 5ms
long_delay();
}

```

```

while(1)
{
    WtiteCommand(0x80);
    long_delay();           //delay at least 5ms
    for(i=0;i<16;i++)
    {
        WriteData(table[i]);
        long_delay();           //delay at least 5ms
    }
    WtiteCommand(0xC0);
    long_delay();           //delay at least 5ms
    for(i=0;i<16;i++)
    {
        WriteData(table[i]);
        long_delay();           //delay at least 5ms
    }
    WtiteCommand(0x01);
}
}

```

8.0.3 PICC REFERENCED CODE FOR PIC SERIES MCU(USING MSSP MODULE)

```

//PIC16F877A,11.0592MHZ,PICC
//*********************************************************************//
#include <pic.h>
#include <pic1687x.h>
//*********************************************************************//
#define uchar unsigned char
//*********************************************************************//
#define sda RC4
#define sclk RC3
//*********************************************************************//
__CONFIG(HS & WDTDIS & PWRTDIS & BORDIS & LVPDIS & DUNPROT & UNPROTECT);
//*********************************************************************//
const uchar table[]=
{
    "This is program for "
    "The Master is:16f877"
    "i2c interface.      "
    "The Slave is:16f689 "
};

void long_delay(void)
{
    uchar i,j;
    for(i=0;i<25;i++)
    for(j=0;j<255;j++)
        ;
}

void short_delay(void)
{
    uchar i,j;
    for(i=0;i<1;i++)
    for(j=0;j<1;j++)
        ;
}

void start(void)
{
    SEN=1;
    while(!SSPIF);
}

```

```

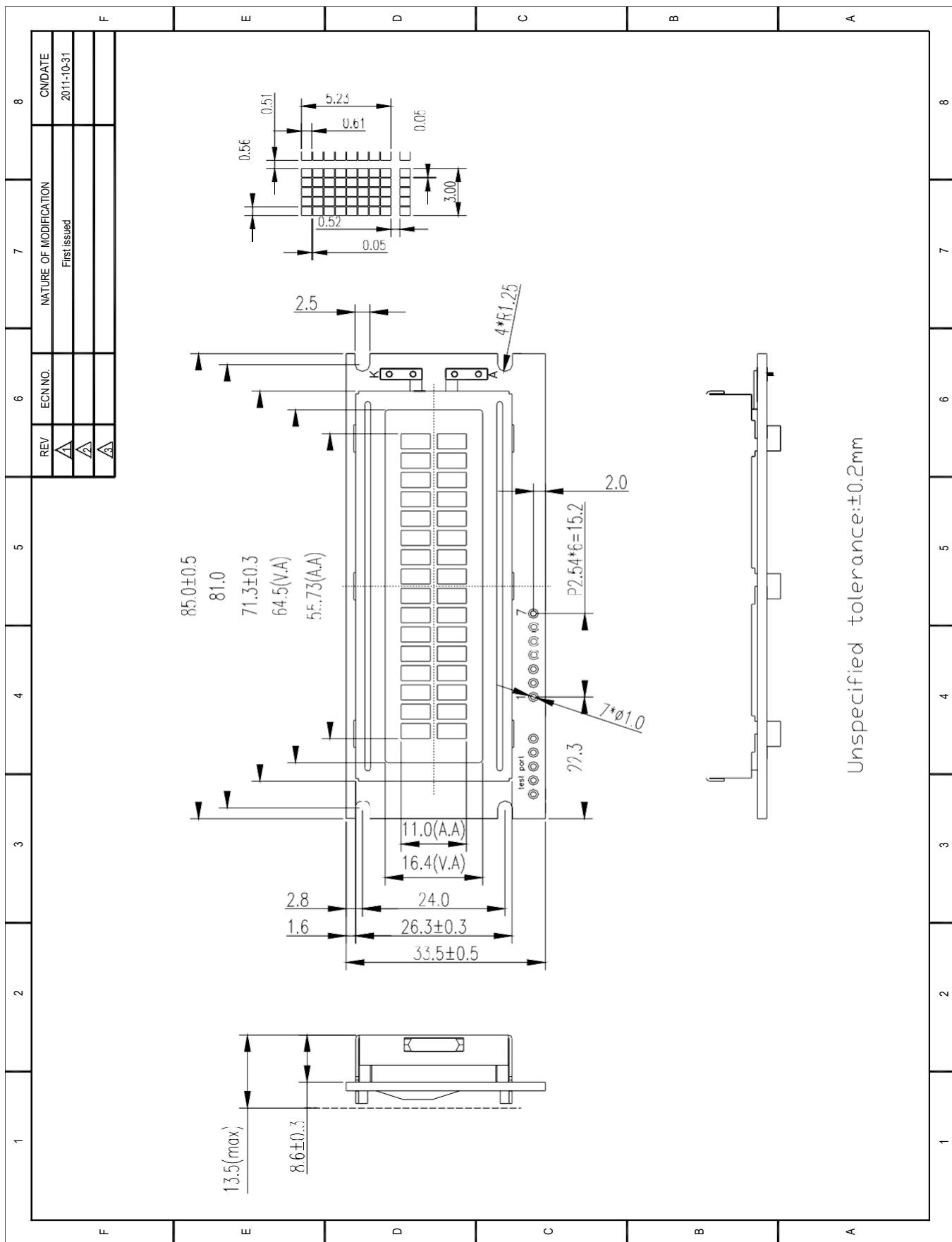
SSPIF=0;
}
//*****
void stop(void)
{
PEN=1;
while(!SSPIF);
SSPIF=0; }
//*****
void write_byte(byte)
{
SSPBUF=byte;
while(!SSPIF);
while(ACKSTAT);
SSPIF=0; }
//*****
void WtiteCommand(command)
{
start();
write_byte(0xa0);
write_byte(0x00);
write_byte(command);
stop();
}
//*****
void WriteData(data)
{
start();
write_byte(0xa0);
write_byte(0x80);
write_byte(data);
stop();
}
//*****
main()
{
uchar i;
long_delay();           //delay at least 15ms
long_delay();
POR=1;
BOR=1;
SSPSTAT=0x00;
TRISC=0x18;
SSPADD=0x09;
SSPIE=0;
SSPCON=0x28;
INTCON=0x00;
WtiteCommand(0x01);
long_delay();           //delay at least 5ms
WtiteCommand(0x38);
long_delay();           //delay at least 5ms
WtiteCommand(0x0f);
long_delay();           //delay at least 5ms
WtiteCommand(0x06);
long_delay();           //delay at least 5ms
while(1)
{
WtiteCommand(0x80);   //delay at least 5ms
for(i=0;i<16;i++)
{
WriteData(table[i]);
}
}
}

```

```
long_delay();           //delay at least 5ms
}
WtiteCommand(0xC0);
long_delay();           //delay at least 5ms
for(i=0;i<16;i++)
{
    WriteData(table[i]);
    long_delay();           //delay at least 5ms
}

WtiteCommand(0x01);
}
}
//*****
```

9.0 MECHANICAL DIAGRAM



10.0 RELIABILITY TEST

NO	Test Item	Description	Test Condition	Remark
1	Environmental Test	High temperature storage Applying the high storage temperature Under normal humidity for a long time Check normal performance	80 °C 96hrs	
2		Low temperature storage Applying the low storage temperature Under normal humidity for a long time Check normal performance	-30°C 96hrs	
3		High temperature Operation Apply the electric stress(Voltage and current) Under high temperature for a long time	70 °C 96hrs	Note1
4		Low temperature Operation Apply the electric stress Under low temperature for a long time	-20°C 96hrs	Note1 Note2
5		High temperature/High Humidity Storage Apply high temperature and high humidity storage for a long time	90% RH 40°C 96hrs	Note2
6		Temperature Cycle Apply the low and high temperature cycle $-30^{\circ}\text{C} <> 25^{\circ}\text{C} <> 80^{\circ}\text{C} <> 25^{\circ}\text{C}$ 30min 10min 30min 10min ↔ → 1 cycle Check normal performance	-30°C/80°C 10 cycle	
7	Mechanical Test	Vibration test(Package state) Applying vibration to product check normal performance	Freq:10~55~10Hz Amplitude:0.75mm 1cycle time:1min X.Y.Z every direction for 15 cycles	
8		Shock test(package state) Applying shock to product check normal performance	Drop them through 70cm height to strike horizontal plane	
9	Other			

Remark

Note1:Normal operations condition ($25^{\circ}\text{C} \pm 5^{\circ}\text{C}$).

Note2:Pay attention to keep dewdrops from the module during this test.

11.0 DISPLAY INSTRUCTION TABLE AND DDRAM ADDRESS

Instruction	Instruction Code											Description		Execution time (Temp = 25°C)		
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Fosc= 190KHz	Fosc= 270KHz	Fosc= 350KHz			
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address to "00H" from AC		2.16ms	1.52ms	1.18ms	
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.		2.16ms	1.52ms	1.18ms	
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Assign cursor moving direction and enable the shift of entire display		53μs	38μs	29μs	
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor(C), and blinking of cursor(B) on/off control bit.		53μs	38μs	29μs	
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.		53μs	38μs	29μs	
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5x10 dots/5x8 dots)		53μs	38μs	29μs	
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.		53μs	38μs	29μs	
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter		53μs	38μs	29μs	
Read Busy Flag and Address Counter	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.					
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).		53μs	38μs	29μs	
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).		53μs	38μs	29μs	

Display

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

12.0 STANDARD CHARACTER PATTERNS

Lower 4 Bits Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000 CG RAM (1)	CG		あ	P	^	P					—	う	ミ	エ	ピ	
xxxx0001 (2)	!	1	A	Q	あ	q				□	ア	チ	4	ä	q	
xxxx0010 (3)	"	2	B	R	b	r				「	イ	リ	×	þ	ø	
xxxx0011 (4)	#	3	C	S	c	s				」	ウ	テ	モ	€	œ	
xxxx0100 (5)	\$	4	D	T	d	t				、	エ	ト	ト	₩	₩	
xxxx0101 (6)	%	5	E	U	e	u				・	オ	ナ	ユ	€	ø	
xxxx0110 (7)	&	6	F	V	f	v				ヲ	カ	ニ	ヨ	Þ	Σ	
xxxx0111 (8)	'	7	G	W	g	w				ア	キ	ヌ	ラ	₪	₪	
xxxx1000 (1)	(8	H	X	h	x				イ	ク	ネ	リ	¤	¤	
xxxx1001 (2))	9	I	Y	i	y				ウ	ケ	ル	ル	-'	¥	
xxxx1010 (3)	*	:	J	Z	j	z				エ	コ	ハ	レ	J	₩	
xxxx1011 (4)	+	;	K	C	k	c				オ	サ	ヒ	ロ	*	¤	
xxxx1100 (5)	,	<	L	¥	l	l				ヤ	シ	フ	ワ	◊	¤	
xxxx1101 (6)	-	=	M	J	m	}				ユ	ズ	ス	ン	£	÷	
xxxx1110 (7)	.	>	N	^	n	†				ミ	セ	ホ	ン	ñ		
xxxx1111 (8)	/	?	O	_	o	†				々	々	々	々	ö	■	

Note: The character generator RAM is the RAM with which the user can rewrite character patterns by program.

13.0 PRECAUTION FOR USING LCM

1. When design the product with this LCD Module, make sure the viewing angle matches to its purpose of usage.
2. As LCD panel is made of glass substrate, Dropping the LCD module or banging it against hard objects may cause cracking or fragmentation. Especially at corners and edges.
3. Although the polarizer of this LCD Module has the anti-glare coating, always be careful not to scratch its surface. Use of a plastic cover is recommended to protect the surface of polarizer.
4. If the LCD module is stored at below specified temperature, the LC material may freeze and be deteriorated. If it is stored at above specified temperature, the molecular orientation of the LC material may change to Liquid state and it may not revert to its original state. Excessive temperature and humidity could cause polarizer peel off or bubble. Therefore, the LCD module should always be stored within specified temperature range.
5. Saliva or water droplets must be wiped off immediately as those may leave stains or cause color changes if remained for a long time. Water vapor will cause corrosion of ITO electrodes.
6. If the surface of LCD panel needs to be cleaned, wipe it swiftly with cotton or other soft cloth. If it is not still clean enough, blow a breath on the surface and wipe again.
7. The module should be driven according to the specified ratings to avoid malfunction and permanent damage. Applying DC voltage cause a rapid deterioration of LC material. Make sure to apply alternating waveform by continuous application of the M signal. Especially the power ON/OFF sequence should be kept to avoid latch-up of driver LSIs and DC charge up to LCD panel.
8. Mechanical Considerations
 - a) LCM are assembled and adjusted with a high degree of precision. Avoid excessive shocks and do not make any alterations or modifications. The following should be noted.
 - b) Do not tamper in any way with the tabs on the metal frame.
 - c) Do not modify the PCB by drilling extra holes, changing its outline, moving its components or modifying its pattern.
 - d) Do not touch the elastomer connector; especially insert a backlight panel (for example, EL).
 - e) When mounting a LCM makes sure that the PCB is not under any stress such as bending or twisting. Elastomer contacts are very delicate and missing pixels could result from slight dislocation of any of the elements.
 - f) Avoid pressing on the metal bezel, otherwise the elastomer connector could be deformed and lose contact, resulting in missing pixels.
9. Static Electricity
 - a) Operator

**Ware the electrostatics shielded clothes because human body may be statically charged if not wear shielded clothes.
Never touch any of the conductive parts such as the LSI pads; the copper leads on the PCB and the interface terminals with any parts of the human body.**

b) Equipment

There is a possibility that the static electricity is charged to the equipment, which has a function of peeling or friction action (ex: conveyer, soldering iron, working table). Earth the equipment through proper resistance (electrostatic earth: 1×10^8 ohm).

Only properly grounded soldering irons should be used.

If an electric screwdriver is used, it should be well grounded and shielded from commutator sparks.

The normal static prevention measures should be observed for work clothes and working benches; for the latter conductive (rubber) mat is recommended.

c) Floor

Floor is the important part to drain static electricity, which is generated by operators or equipment.

There is a possibility that charged static electricity is not properly drained in case of insulating floor. Set the electrostatic earth (electrostatic earth: 1×10^8 ohm).

d) Humidity

Proper humidity helps in reducing the chance of generating electrostatic charges. Humidity should be kept over 50%RH.

e) Transportation/storage

The storage materials also need to be anti-static treated because there is a possibility that the human body or storage materials such as containers may be statically charged by friction or peeling.

The modules should be kept in antistatic bags or other containers resistant to static for storage.

f) Soldering

Solder only to the I/O terminals. Use only soldering irons with proper grounding and no leakage.

Soldering temperature : $280^\circ\text{C} \pm 10^\circ\text{C}$

Soldering time: 3 to 4 sec.

Use eutectic solder with resin flux fill.

If flux is used, the LCD surface should be covered to avoid flux spatters. Flux residue should be removed afterwards.

g) Others

The laminator (protective film) is attached on the surface of LCD panel to prevent it from scratches or stains. It should be peeled off slowly using static eliminator.

Static eliminator should also be installed to the workbench to prevent LCD module from static charge.

10. Operation

- a) Driving voltage should be kept within specified range; excess voltage shortens display life.
 - b) Response time increases with decrease in temperature.
 - c) Display may turn black or dark blue at temperatures above its operational range; this is (however not pressing on the viewing area) may cause the segments to appear "fractured".
 - d) Mechanical disturbance during operation (such as pressing on the viewing area) may cause the segments to appear "fractured".
11. If any fluid leaks out of a damaged glass cell, wash off any human part that comes into contact with soap and water. The toxicity is extremely low but caution should be exercised at all the time.
12. Disassembling the LCD module can cause permanent damage and it should be strictly avoided.
13. LCD retains the display pattern when it is applied for long time (Image retention). To prevent image retention, do not apply the fixed pattern for a long time. Image retention is not a deterioration of LCD. It will be removed after display pattern is changed.
14. Do not use any materials, which emit gas from epoxy resin (hardener for amine) and silicone adhesive agent (dealcohol or deoxym) to prevent discoloration of polarizer due to gas.
15. Avoid the exposure of the module to the direct sunlight or strong ultraviolet light for a long time.

```

LCD
main.cpp
#include "mbed.h"
#include "Lcd.h"

LCD lcd;

DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

int main() {
    int ctr = 0;
    char buff[18];

    while (1) {
        sprintf(buff, "counter=%d",ctr);
        lcd.LcdString2(0, buff);
        if (++ctr >= 1000)
            ctr = 0;
        wait(0.25);
        led1 = !led1;
    }
}

lcd.cpp

#include "mbed.h"
#include "Lcd.h"

I2C i2cLcd(p28, p27);

static const int devAddr = 0xA0;

//constructor
LCD::LCD()
{
    InitLcd();
}

bool LCD::WriteBytes(const char *data, int length, bool repeated = false)
{
    wait_us(20);
    i2cLcd.start();
    for (int i = 0; i < length; i++) {
        wait_us(20);
        if (i2cLcd.write(data[i]) != 1) { // '1' if an ACK was
received, '0' otherwise
            wait_us(20);
            i2cLcd.stop();
            return false;
        }
    }
}

```

```

        if (repeated == false) {
            wait_us(20);
            i2cLcd.stop();
        }
        return true;
    }

void LCD::WriteLCDcmd(unsigned char command) {
    char cmd[3];
    cmd[0] = devAddr;
    cmd[1] = 0x00;
    cmd[2] = command;
    WriteBytes(cmd, 3, false);
}

void LCD::WriteLCDdata(unsigned char data) {
    char cmd[3];
    cmd[0] = devAddr;
    cmd[1] = 0x80;
    cmd[2] = data;
    WriteBytes(cmd, 3, false);
}

void LCD::LcdString(int line, char* s) {
    if (line == 0) {
        //Set DDRAM Address //first line
        WriteLCDcmd(0x80);
    } else {
        //Set DDRAM Address
        WriteLCDcmd(0x80 | 0x40);    //second line 0xC0
    }
    char c;
    while ((c = *s++) > 0) {
        wait_ms(5);    //5ms
        //Write Data to RAM
        WriteLCDdata(c);
    }
    wait_ms(5);    //5ms
}

void LCD::LcdString2(int line, char* s) {
    if (line == 0) {
        //Set DDRAM Address //first line
        WriteLCDcmd(0x80);
    } else {
        //Set DDRAM Address
        WriteLCDcmd(0x80 | 0x40);    //second line 0xC0
    }
    char c;
    int n = 0;
    while ((c = *s++) > 0) {
        wait_ms(5);    //5ms
        //Write Data to RAM
        WriteLCDdata(c);
        n++;
    }
    for ( ; n < 16 ; n++ ){
        wait_ms(5);    //5ms
    }
}

```

```

        //Write Data to RAM
        WriteLCDdata(' ');
    }
    wait_ms(5); //5ms
}
void LCD::InitLcd() {
    wait_ms(15); //15ms
    WriteLCDcmd(0x01); // Clear Diplay
    wait_ms(5); //5ms
    //ãf•ã, ;ãf³ã, -ã, ·ãf$ãf³ã, »ãffãf^ DL=1,N=1,F=0 0011 1000
    //b5 = 1
    // 0b001x0000; DL; Interface Data length. 8bits/4bits
    // 0b0010x000; N; Numbers of display line. 2-line/1-line
    // 0b00100x00; F; Display font type. 5x10dots/5x8dots
    WriteLCDcmd(0x38);
    wait_ms(5); //5ms
    //Display ON/OFF Control
    //b3 = 1
    // 0b00001x00; D; Set display on/off.
    // 0b000010x0; C; Set curthor on/off.
    // 0b0000100x; B; Set blinking of curthor on/off.
    WriteLCDcmd(0x0c);
    wait_ms(5); //5ms
    // Entry Mode Set
    //b2 = 1
    // 0b000001x0; I/D;Assign curthor moving direction.
    // 0b0000010x; I/D;Enable the shift of entire display.
    WriteLCDcmd(0x06);
    wait_ms(5); //5ms
}
void LCD::ClearDisplay() {
    WriteLCDcmd(0x01); // Clear Diplay
    wait_ms(5); //5ms
}
void LCD::ClearLine(int line) {
    if (line == 0) {
        //Set DDRAM Address //first line
        WriteLCDcmd(0x80);
    } else {
        //Set DDRAM Address
        WriteLCDcmd(0x80 | 0x40); //second line 0xC0
    }
    for (int n = 0; n < 16; n++) {
        wait_ms(5); //5ms
        //Write Data to RAM
        WriteLCDdata(' ');
    }
    wait_ms(5); //5ms
}

```

LCD.h

```
#ifndef LCD_H_
#define LCD_H_
```

```
class LCD {
    bool WriteBytes(const char *data, int length, bool repeated);
    void WriteLCDcmd(unsigned char command);
    void WriteLCDdata(unsigned char data);

public:
    LCD();
    void InitLcd(void);
    void LcdString(int line, char* s);
    void LcdString2(int line, char* s);
    void ClearDisplay(void);
    void ClearLine(int line);
};

#endif /* LCD_H_ */
```

HC-SR04 User Guide

1. Ultrasonic Distance Measurement Principles

The transmitter emits a 8 bursts of an directional 40KHz ultrasonic wave when triggered and starts a timer. Ultrasonic pulses travel outward until they encounter an object, The object causes the the wave to be reflected back towards the unit. The ultrasonic receiver would detect the reflected wave and stop the stop timer. The velocity of the ultrasonic burst is 340m/sec. in air. Based on the number of counts by the timer, the distance can be calculated between the object and transmitter. The TRD Measurement formula is expressed as: $D = C \times T$ which is known as the time/rate/distance measurement formula where D is the measured distance, and R is the propagation velocity (Rate) in air (speed of sound) and T represents time. In this application T is divided by 2 as T is double the time value from transmitter to object back to receiver.

2. Product Features

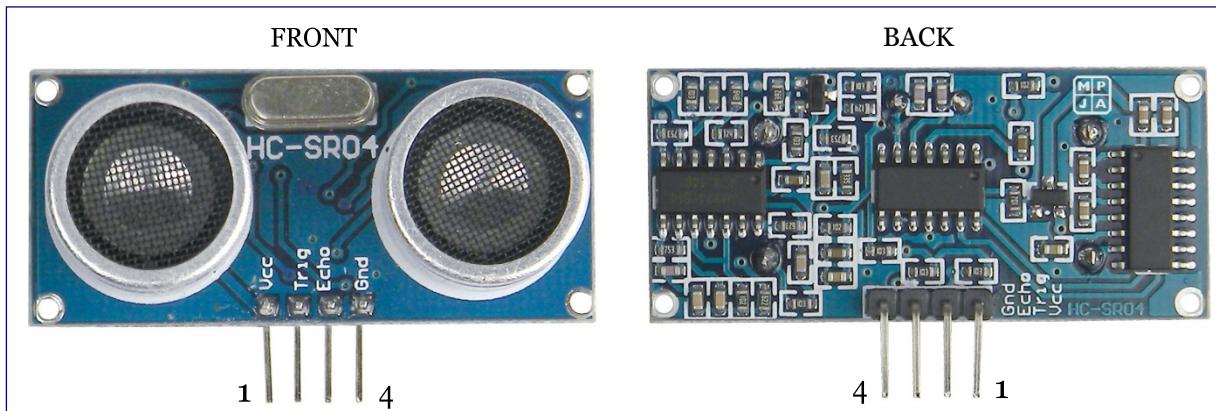
Features

- Stable performance (Xtal.)
- Accurate distance measurement
- High-density SMD Board
- Close Range (2cm)

Uses

- Robotics barrier
- Object distance measurement
- Level detection
- Security systems
- Vehicle detection/avoidance

3. Product Views



4. Module Pin Assignments

	Pin Symbol	Pin Function Description
1	VCC	5V power supply
2	Trig	Trigger Input pin
3	Echo	Receiver Output pin
4	GND	Power ground

5. Electrical Specifications

WARARNING

Do Not connect Module with Power Applied! Always apply power after connecting Connect "GND" Terminal first

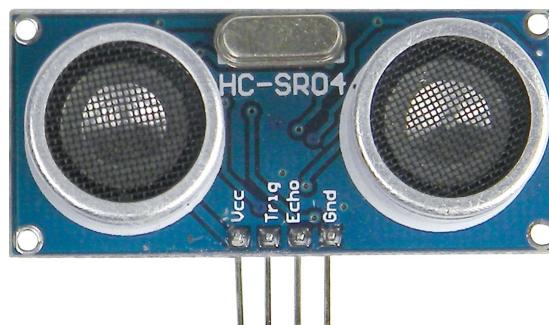
Electrical Parameters	HC-SR04 Ultrasonic Module
Operating Voltage	5VDC
Operating Current	15mA
Operating Frequency	40KHz
Max. Range	4m
Nearest Range	2cm
Measuring Angle	15 Degrees
Input Trigger Signal	10us min. TTL pulse
Output Echo Signal	TTL level signal, proportional to distance
Board Dimensions	1-13/16" X 13/16" X 5/8"
Board Connections	4 X 0.1" Pitch Right Angle Header Pins

6. Module Operation

Set Trig and Echo Low to initialize module. Place a minimum 10us High level pulse to "Trigger" (module will automatically send eight 40KHz acoustic bursts). At the same time, Gate the microcontroller timer to start timing.

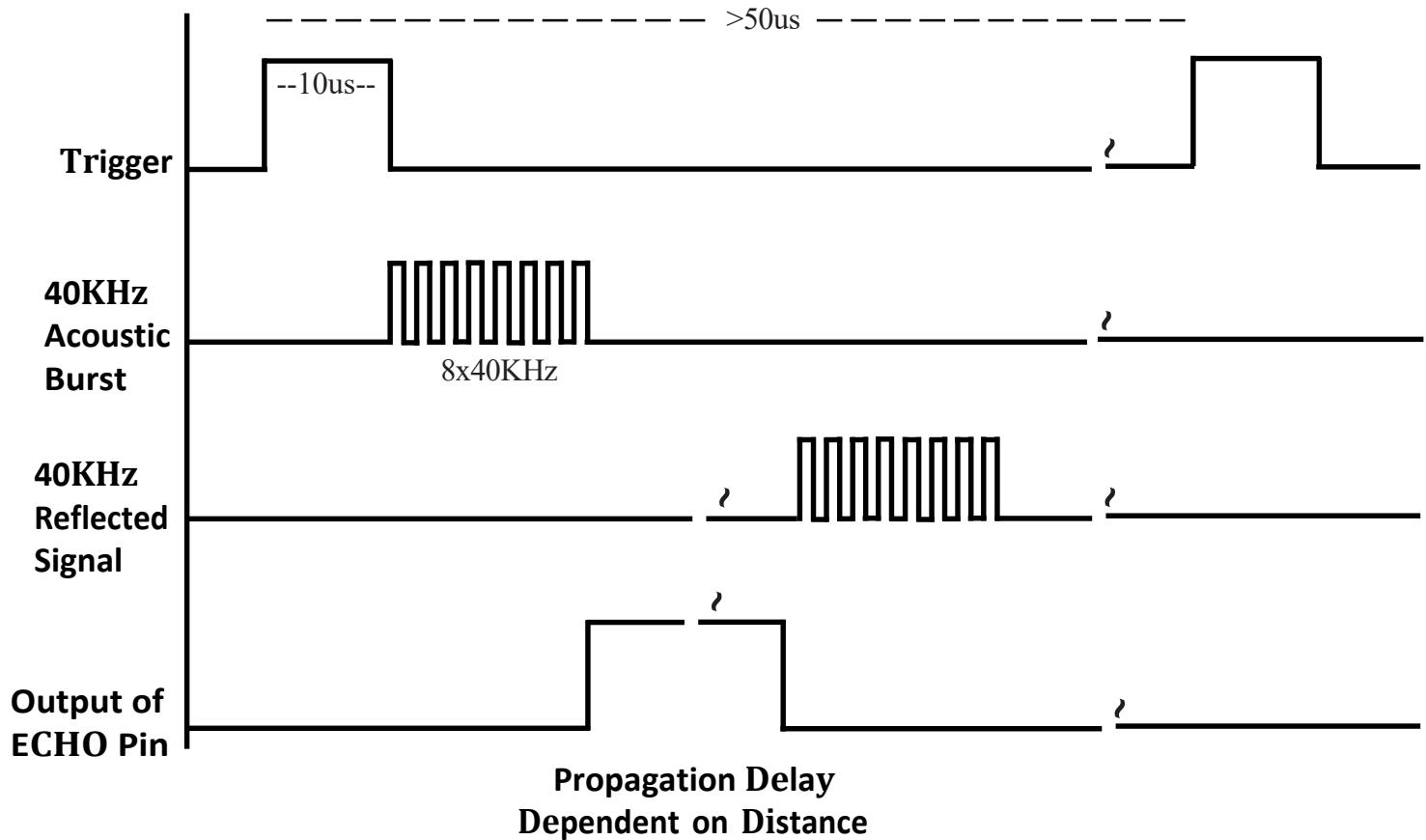
Wait to capture the rising edge output of ECHO port to stop the timer. Now read the time of the counter, which is the ultrasonic propagation time in the air. According to the formula: Distance = (ECHO high level time X ultrasonic velocity (Speed of Sound in air 340m/sec) / 2, you can calculate the distance to the obstacle.

For best results and maximum range, the Object should be larger than 0.5M² the nearer the target object, the smaller it may be



7. Module Timing

HC-SR04 ULTRASONIC MODULE



Trigger 10us min. start measurement from microcontroller.

Max Rep. Rate: 50us

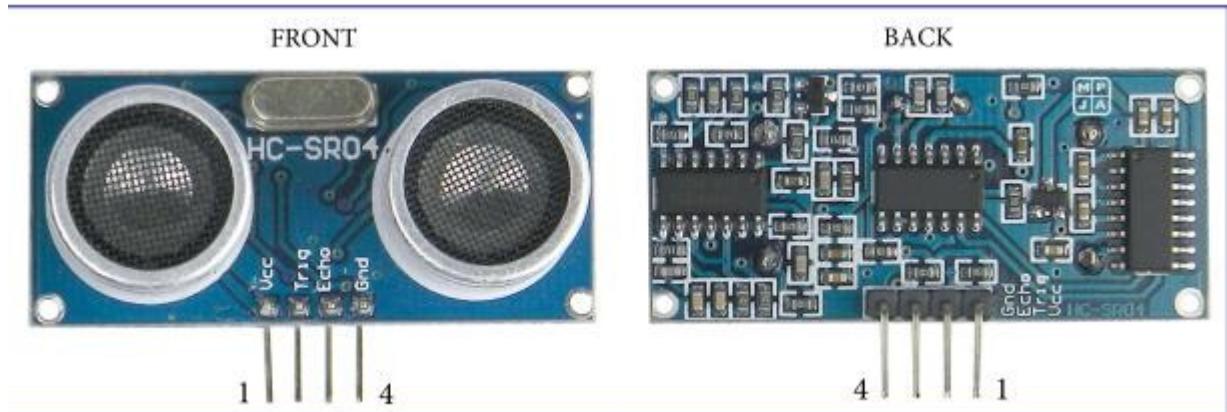
ECHO Output pulse to microcontroller, width is the time from last of 8 40KHz bursts to detected reflected signal (microcontroller Timer gate signal)

Distance in cm = echo pulse width in uS/58

Distance in inch = echo pulse width in uS/148

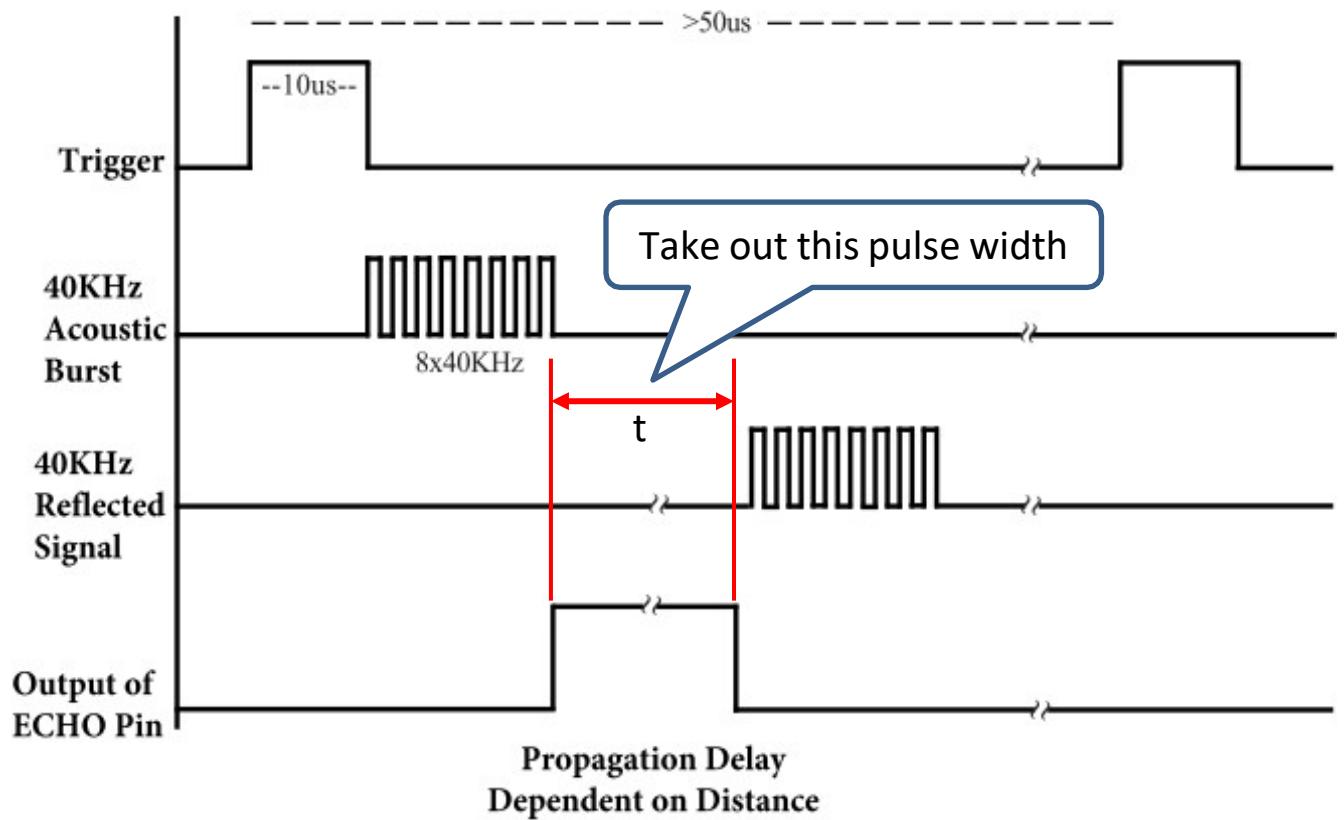
Information obtained from or supplied by Mpja.com or Marlin P. Jones and Associates inc. is supplied as a service to our customers and accuracy is not guaranteed nor is it definitive of any particular part or manufacturer. Use of information and suitability for any application is at users own discretion and user assumes all risk.

Ultrasonic Distance Measurement



	Pin Symbol	Pin Function Description
1	VCC	5V power supply
2	Trig	Trigger Input pin
3	Echo	Receiver Output pin
4	GND	Power ground

Electrical Parameters	HC-SR04 Ultrasonic Module
Operating Voltage	5VDC
Operating Current	15mA
Operating Frequency	40KHz
Max. Range	4m
Nearest Range	2cm
Measuring Angle	15 Degrees
Input Trigger Signal	10us min. TTL pulse
Output Echo Signal	TTL level signal, proportional to distance



Trigger 10us min. start measurement from microcontroller.

Max Rep. Rate: 50us

ECHO Output pulse to microcontroller, width is the time from last of 8 40KHz bursts to detected reflected signal (microcontroller Timer gate signal)

Distance in cm = echo pulse width in uS/58

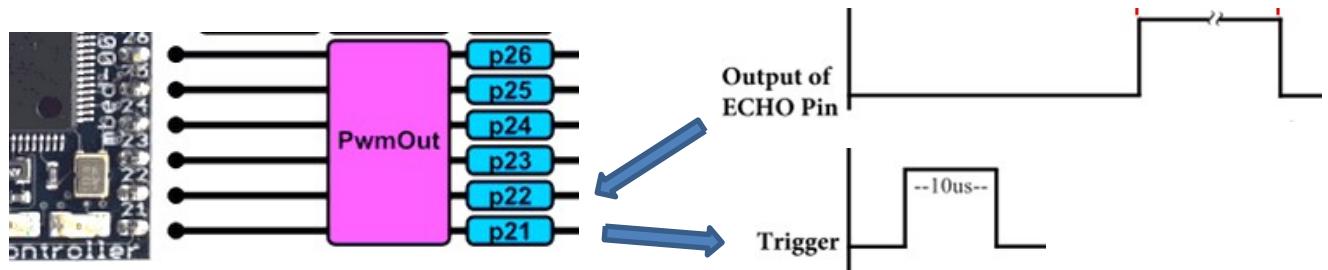
Distance in inch = echo pulse width in uS/148

T = temperature(degree on the centigrade)

t = pulse width(sec)

$$\text{Distance}(m) = t * (331.5 + 0.61 * T) / 2 ;$$

Ex. T=15, t=25ms Distance = 4.25m



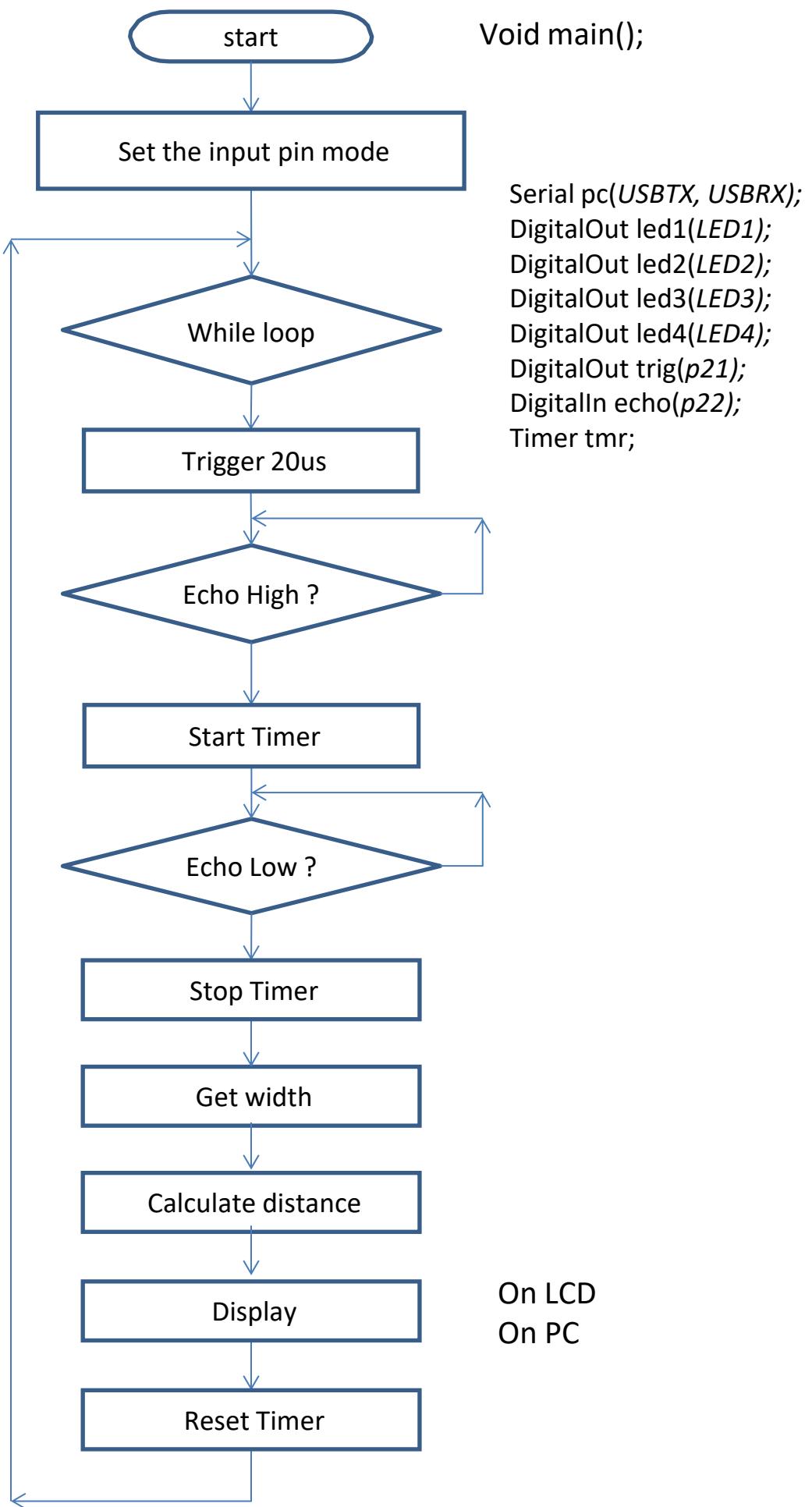
```
DigitalOut trig(p21);  
DigitalIn echo(p22);
```

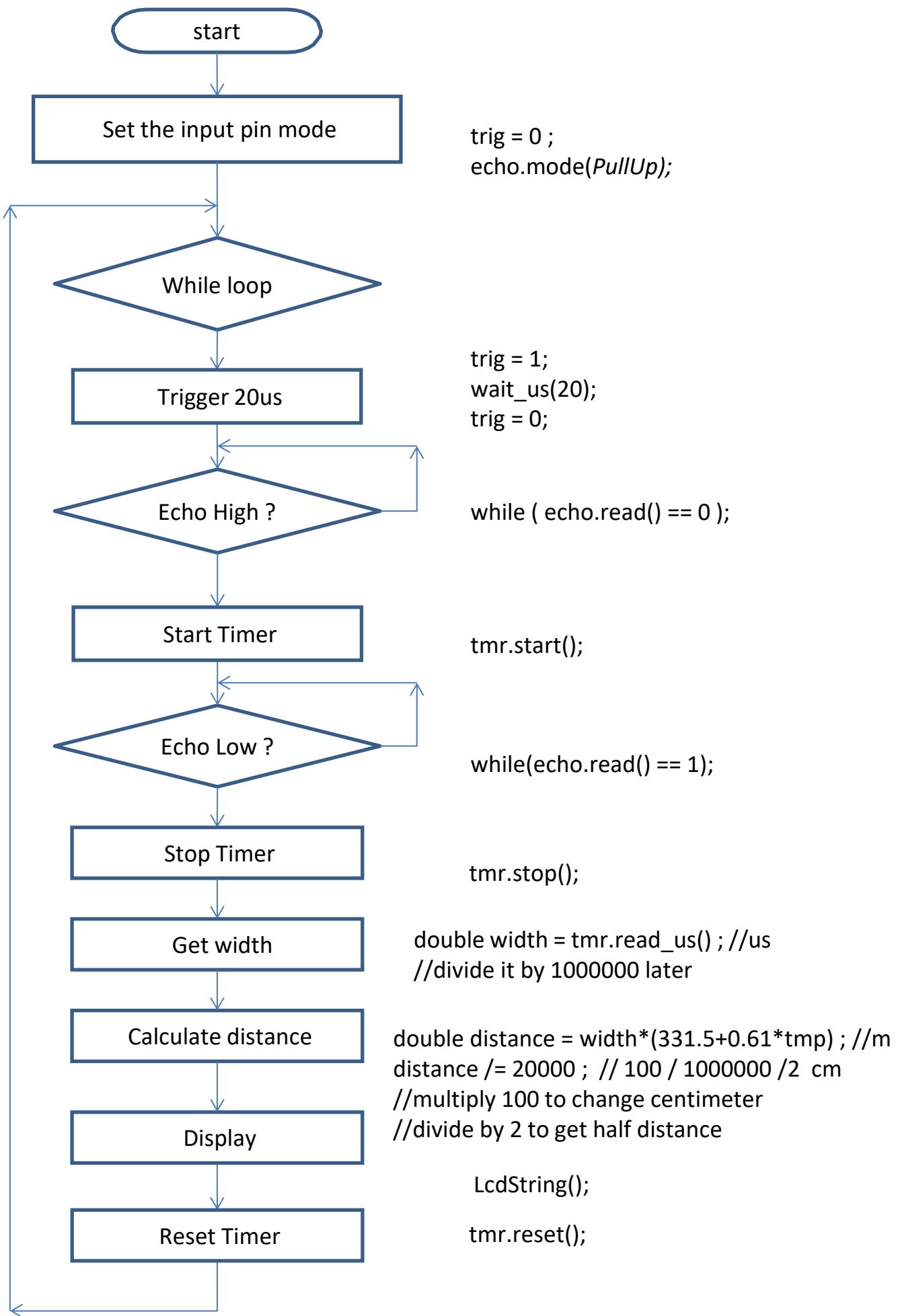
DigitalIn

```
/** Set the input pin mode  
 * @param mode PullUp, PullDown, PullNone, OpenDrain */  
void mode(PinMode pull);  
  
/** Read the input, represented as 0 or 1 (int)  
 * @returns  
 * An integer representing the state of the input pin,  
 * 0 for logical 0, 1 for logical 1 */  
int read();
```

Timer

```
/** Start the timer */  
void start();  
  
/** Stop the timer */  
void stop();  
  
/** Get the time passed in micro-seconds */  
int read_us();  
  
/** Reset the timer to 0.  
 * If it was already counting, it will continue */  
void reset();
```





To avoid an infinite loop, calculate width at the length of 5m max.

When T=30

$$(331.5 + 0.61 * T) / 2 = 175$$

$$\text{From } D = \text{width} * (331.5 + 0.61 * T) / 2$$

When D=5m

$$\text{width} = 5 / 175 \text{ sec} = 28.6 \text{ ms} = 28600 \text{ us}$$

Instead of “while (echo.read() == 0);”

```
int ctr = 0 ;
bool flag = true;
tmr.start();
while ( echo.read() == 0 ){
    if (++ctr >= 50 ){
        ctr=0;
        if (tmr.read_us() >= 28600 ){      //5m
            flag = false;
            break;
        }
    }
}
tmr.stop();
tmr.reset();
if ( flag == false ){
    error processing;
}
```

Revise likewise

“while (echo.read() == 1);”

Think about three application of the distance sensor.

- 1) Automatic door
- 2) Invasion detection
- 3) The near miss detection

LM35

Precision Centigrade Temperature Sensors

General Description

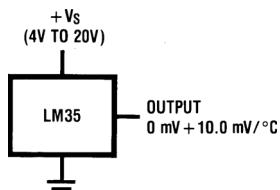
The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of $\pm 1^\circ\text{C}$ at room temperature and $\pm 3^\circ\text{C}$ over a full -55 to $+150^\circ\text{C}$ temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only $60\ \mu\text{A}$ from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a -55 to $+150^\circ\text{C}$ temperature range, while the LM35C is rated for a -40 to $+110^\circ\text{C}$ range (-10° with improved accuracy). The LM35 series is available pack-

aged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

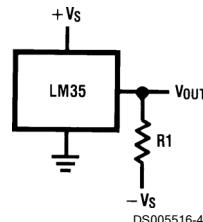
Features

- Linear $+10.0\ \text{mV}/^\circ\text{C}$ scale factor
- 0.5°C accuracy guaranteeable (at $+25^\circ\text{C}$)
- Rated for full -55 to $+150^\circ\text{C}$ range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than $60\ \mu\text{A}$ current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only $\pm 1^\circ\text{C}$ typical
- Low impedance output, $0.1\ \Omega$ for 1 mA load

Typical Applications



DS005516-3

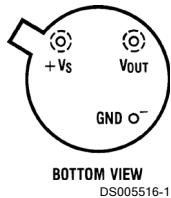
Choose $R_1 = -V_s/50\ \mu\text{A}$

$$\begin{aligned} V_{\text{OUT}} &= +1,500\ \text{mV} \text{ at } +150^\circ\text{C} \\ &= -550\ \text{mV} \text{ at } -55^\circ\text{C} \end{aligned}$$

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

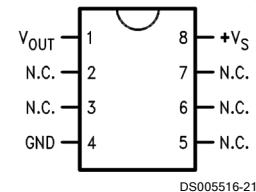
TO-46
Metal Can Package*



*Case is connected to negative pin (GND)

Order Number LM35H, LM35AH, LM35CH, LM35CAH or

SO-8
Small Outline Molded Package



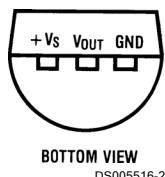
N.C. = No Connection

Top View

LM35DH
See NS Package Number H03H

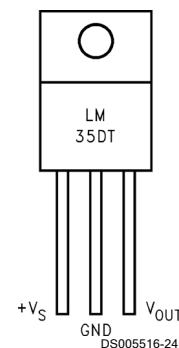
Order Number LM35DM
See NS Package Number M08A

TO-92
Plastic Package



Order Number LM35CZ,
LM35CAZ or LM35DZ
See NS Package Number Z03A

TO-220
Plastic Package*



*Tab is connected to the negative pin (GND).

Note: The LM35DT pinout is different than the discontinued LM35DP.

Order Number LM35DT
See NS Package Number TA03F

Absolute Maximum Ratings (Note 10)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	+35V to -0.2V
Output Voltage	+6V to -1.0V
Output Current	10 mA
Storage Temp.; TO-46 Package,	-60°C to +180°C
TO-92 Package,	-60°C to +150°C
SO-8 Package,	-65°C to +150°C
TO-220 Package,	-65°C to +150°C
Lead Temp.: TO-46 Package, (Soldering, 10 seconds)	300°C

TO-92 and TO-220 Package, (Soldering, 10 seconds)	260°C
SO Package (Note 12)	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
ESD Susceptibility (Note 11)	2500V
Specified Operating Temperature Range: T _{MIN} to T _{MAX} (Note 2)	
LM35, LM35A	-55°C to +150°C
LM35C, LM35CA	-40°C to +110°C
LM35D	0°C to +100°C

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35A			LM35CA			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy (Note 7)	T _A =+25°C	±0.2	±0.5		±0.2	±0.5		°C
	T _A =-10°C	±0.3			±0.3		±1.0	°C
	T _A =T _{MAX}	±0.4	±1.0		±0.4	±1.0		°C
	T _A =T _{MIN}	±0.4	±1.0		±0.4		±1.5	°C
Nonlinearity (Note 8)	T _{MIN} ≤T _A ≤T _{MAX}	±0.18		±0.35	±0.15		±0.3	°C
Sensor Gain (Average Slope)	T _{MIN} ≤T _A ≤T _{MAX}	+10.0	+9.9, +10.1		+10.0		+9.9, +10.1	mV/°C
Load Regulation (Note 3) 0≤I _L ≤1 mA	T _A =+25°C	±0.4	±1.0		±0.4	±1.0		mV/mA
	T _{MIN} ≤T _A ≤T _{MAX}	±0.5		±3.0	±0.5		±3.0	mV/mA
Line Regulation (Note 3)	T _A =+25°C	±0.01	±0.05		±0.01	±0.05		mV/V
	4V≤V _S ≤30V	±0.02		±0.1	±0.02		±0.1	mV/V
Quiescent Current (Note 9)	V _S =+5V, +25°C	56	67		56	67		µA
	V _S =+5V	105		131	91		114	µA
	V _S =+30V, +25°C	56.2	68		56.2	68		µA
	V _S =+30V	105.5		133	91.5		116	µA
Change of Quiescent Current (Note 3)	4V≤V _S ≤30V, +25°C	0.2	1.0		0.2	1.0		µA
	4V≤V _S ≤30V	0.5		2.0	0.5		2.0	µA
Temperature Coefficient of Quiescent Current		+0.39		+0.5	+0.39		+0.5	µA/°C
Minimum Temperature for Rated Accuracy	In circuit of <i>Figure 1</i> , I _L =0	+1.5		+2.0	+1.5		+2.0	°C
Long Term Stability	T _J =T _{MAX} , for 1000 hours	±0.08			±0.08			°C

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35			LM35C, LM35D			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy, LM35, LM35C (Note 7)	T _A =+25°C	±0.4	±1.0		±0.4	±1.0		°C
	T _A =-10°C	±0.5			±0.5		±1.5	°C
	T _A =T _{MAX}	±0.8	±1.5		±0.8		±1.5	°C
	T _A =T _{MIN}	±0.8		±1.5	±0.8		±2.0	°C
Accuracy, LM35D (Note 7)	T _A =+25°C				±0.6	±1.5		°C
	T _A =T _{MAX}				±0.9		±2.0	°C
	T _A =T _{MIN}				±0.9		±2.0	°C
Nonlinearity (Note 8)	T _{MIN} ≤T _A ≤T _{MAX}	±0.3		±0.5	±0.2		±0.5	°C
Sensor Gain (Average Slope)	T _{MIN} ≤T _A ≤T _{MAX}	+10.0	+9.8, +10.2		+10.0		+9.8, +10.2	mV/°C
Load Regulation (Note 3) 0≤I _L ≤1 mA	T _A =+25°C	±0.4	±2.0		±0.4	±2.0		mV/mA
	T _{MIN} ≤T _A ≤T _{MAX}	±0.5		±5.0	±0.5		±5.0	mV/mA
Line Regulation (Note 3)	T _A =+25°C	±0.01	±0.1		±0.01	±0.1		mV/V
	4V≤V _S ≤30V	±0.02		±0.2	±0.02		±0.2	mV/V
Quiescent Current (Note 9)	V _S =+5V, +25°C	56	80		56	80		µA
	V _S =+5V	105		158	91		138	µA
	V _S =+30V, +25°C	56.2	82		56.2	82		µA
	V _S =+30V	105.5		161	91.5		141	µA
Change of Quiescent Current (Note 3)	4V≤V _S ≤30V, +25°C	0.2	2.0		0.2	2.0		µA
	4V≤V _S ≤30V	0.5		3.0	0.5		3.0	µA
Temperature Coefficient of Quiescent Current		+0.39		+0.7	+0.39		+0.7	µA/°C
Minimum Temperature for Rated Accuracy	In circuit of <i>Figure 1</i> , I _L =0	+1.5		+2.0	+1.5		+2.0	°C
Long Term Stability	T _J =T _{MAX} , for 1000 hours	±0.08			±0.08			°C

Note 1: Unless otherwise noted, these specifications apply: -55°C≤T_J≤+150°C for the LM35 and LM35A; -40°C≤T_J≤+110°C for the LM35C and LM35CA; and 0°C≤T_J≤+100°C for the LM35D. V_S=+5Vdc and I_{LOAD}=50 µA, in the circuit of *Figure 2*. These specifications also apply from +2°C to T_{MAX} in the circuit of *Figure 1*. Specifications in **boldface** apply over the full rated temperature range.

Note 2: Thermal resistance of the TO-46 package is 400°C/W, junction to ambient, and 24°C/W junction to case. Thermal resistance of the TO-92 package is 180°C/W junction to ambient. Thermal resistance of the small outline molded package is 220°C/W junction to ambient. Thermal resistance of the TO-220 package is 90°C/W junction to ambient. For additional thermal resistance information see table in the Applications section.

Note 3: Regulation is measured at constant junction temperature, using pulse testing with a low duty cycle. Changes in output due to heating effects can be computed by multiplying the internal dissipation by the thermal resistance.

Note 4: Tested Limits are guaranteed and 100% tested in production.

Note 5: Design Limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels.

Note 6: Specifications in **boldface** apply over the full rated temperature range.

Note 7: Accuracy is defined as the error between the output voltage and 10mV/°C times the device's case temperature, at specified conditions of voltage, current, and temperature (expressed in °C).

Note 8: Nonlinearity is defined as the deviation of the output-voltage-versus-temperature curve from the best-fit straight line, over the device's rated temperature range.

Note 9: Quiescent current is defined in the circuit of *Figure 1*.

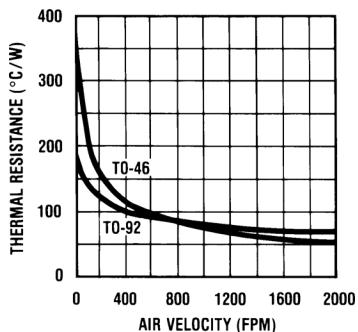
Note 10: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions. See Note 1.

Note 11: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Note 12: See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" or the section titled "Surface Mount" found in a current National Semiconductor Linear Data Book for other methods of soldering surface mount devices.

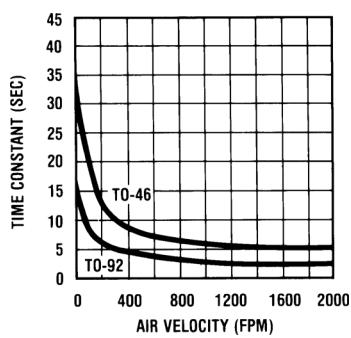
Typical Performance Characteristics

Thermal Resistance
Junction to Air



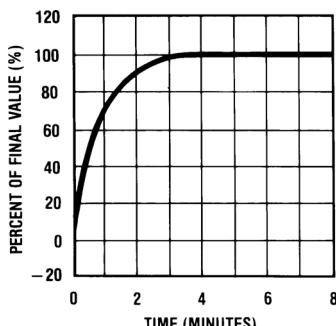
DS005516-25

Thermal Time Constant



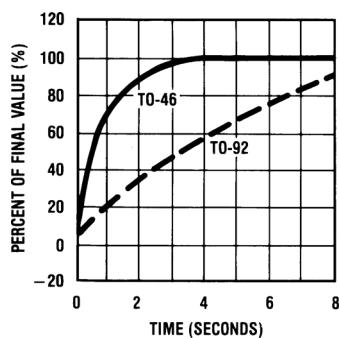
DS005516-26

Thermal Response
in Still Air



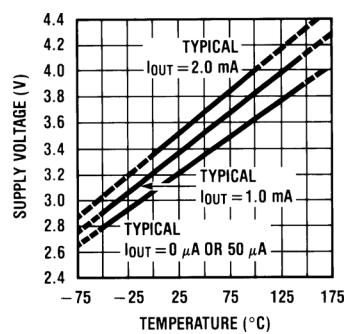
DS005516-27

Thermal Response in
Stirred Oil Bath



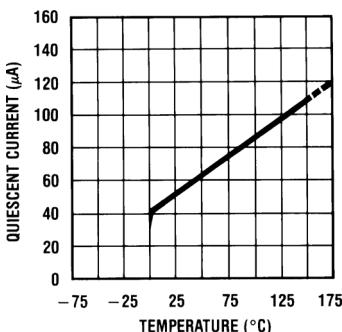
DS005516-28

Minimum Supply
Voltage vs. Temperature



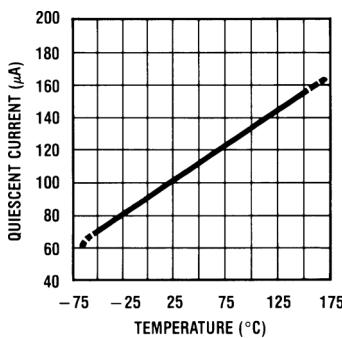
DS005516-29

Quiescent Current
vs. Temperature
(In Circuit of Figure 1.)



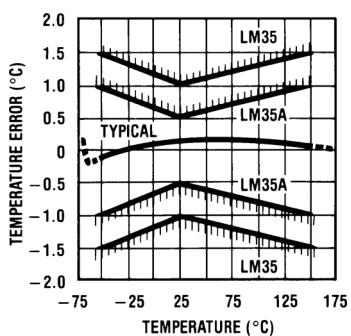
DS005516-30

Quiescent Current
vs. Temperature
(In Circuit of Figure 2.)



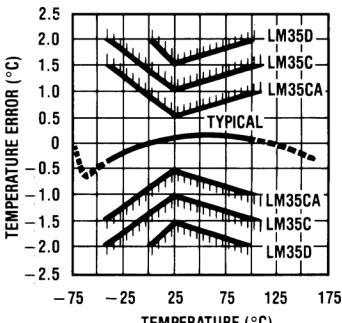
DS005516-31

Accuracy vs. Temperature
(Guaranteed)



DS005516-32

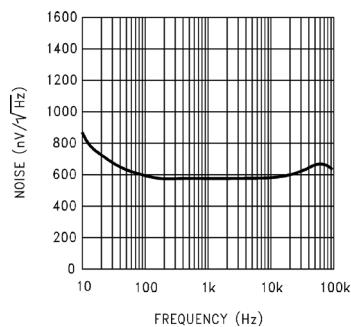
Accuracy vs. Temperature
(Guaranteed)



DS005516-33

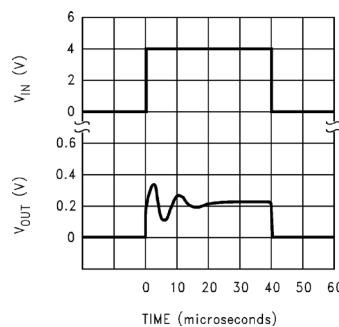
Typical Performance Characteristics (Continued)

Noise Voltage



DS005516-34

Start-Up Response



DS005516-35

Applications

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01°C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is especially true for the TO-92 plastic package, where the copper leads are the principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up these wires with a bead of epoxy which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die's temperature will not be affected by the air temperature.

The TO-46 metal package can also be soldered to a metal surface or pipe without damage. Of course, in that case the

V- terminal of the circuit will be grounded to that metal. Alternatively, the LM35 can be mounted inside a sealed-end metal tube, and can then be dipped into a bath or screwed into a threaded hole in a tank. As with any IC, the LM35 and accompanying wiring and circuits must be kept insulated and dry, to avoid leakage and corrosion. This is especially true if the circuit may operate at cold temperatures where condensation can occur. Printed-circuit coatings and varnishes such as Humiseal and epoxy paints or dips are often used to insure that moisture cannot corrode the LM35 or its connections.

These devices are sometimes soldered to a small light-weight heat fin, to decrease the thermal time constant and speed up the response in slowly-moving air. On the other hand, a small thermal mass may be added to the sensor, to give the steadiest reading despite small deviations in the air temperature.

Temperature Rise of LM35 Due To Self-heating (Thermal Resistance, θ_{JA})

	TO-46, no heat sink	TO-46*, small heat fin	TO-92, no heat sink	TO-92**, small heat fin	SO-8 no heat sink	SO-8** small heat fin	TO-220 no heat sink
Still air	400°C/W	100°C/W	180°C/W	140°C/W	220°C/W	110°C/W	90°C/W
Moving air	100°C/W	40°C/W	90°C/W	70°C/W	105°C/W	90°C/W	26°C/W
Still oil	100°C/W	40°C/W	90°C/W	70°C/W	-	-	-
Stirred oil	50°C/W	30°C/W	45°C/W	40°C/W	-	-	-
(Clamped to metal, Infinite heat sink)		(24°C/W)				(55°C/W)	

*Wakefield type 201, or 1" disc of 0.020" sheet brass, soldered to case, or similar.

**TO-92 and SO-8 packages glued and leads soldered to 1" square of 1/16" printed circuit board with 2 oz. foil or similar.

Typical Applications

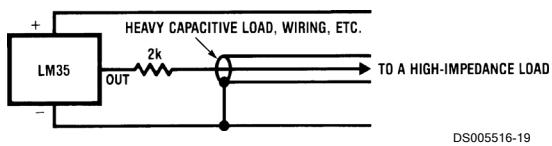


FIGURE 3. LM35 with Decoupling from Capacitive Load

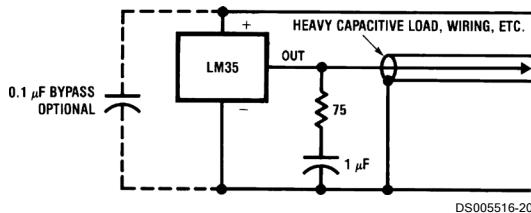


FIGURE 4. LM35 with R-C Damper

CAPACITIVE LOADS

Like most micropower circuits, the LM35 has a limited ability to drive heavy capacitive loads. The LM35 by itself is able to drive 50 pF without special precautions. If heavier loads are anticipated, it is easy to isolate or decouple the load with a resistor; see *Figure 3*. Or you can improve the tolerance of capacitance with a series R-C damper from output to ground; see *Figure 4*.

When the LM35 is applied with a 200Ω load resistor as shown in *Figure 5*, *Figure 6* or *Figure 8* it is relatively immune to wiring capacitance because the capacitance forms a by-pass from ground to input, not on the output. However, as with any linear circuit connected to wires in a hostile environment, its performance can be affected adversely by intense electromagnetic sources such as relays, radio transmitters, motors with arcing brushes, SCR transients, etc., as its wiring can act as a receiving antenna and its internal junctions can act as rectifiers. For best results in such cases, a bypass capacitor from V_{IN} to ground and a series R-C damper such as 75Ω in series with 0.2 or 1 μF from output to ground are often useful. These are shown in *Figure 13*, *Figure 14*, and *Figure 16*.

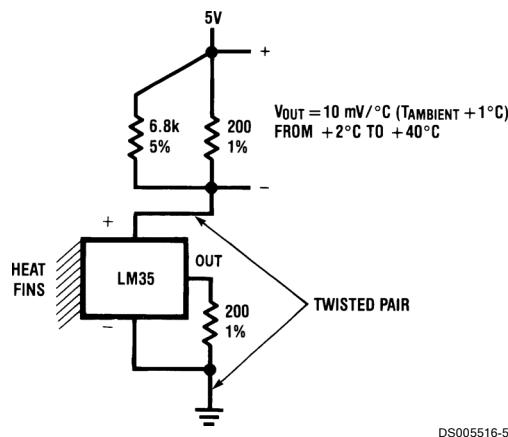


FIGURE 5. Two-Wire Remote Temperature Sensor (Grounded Sensor)

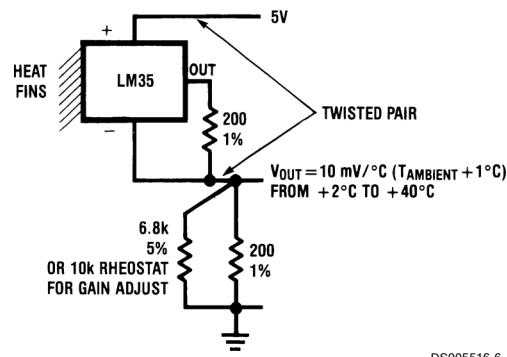


FIGURE 6. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

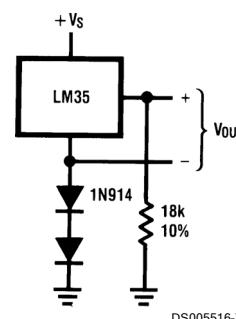


FIGURE 7. Temperature Sensor, Single Supply, -55° to +150°C

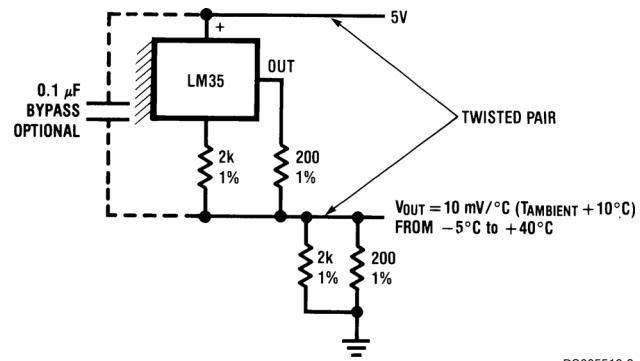


FIGURE 8. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

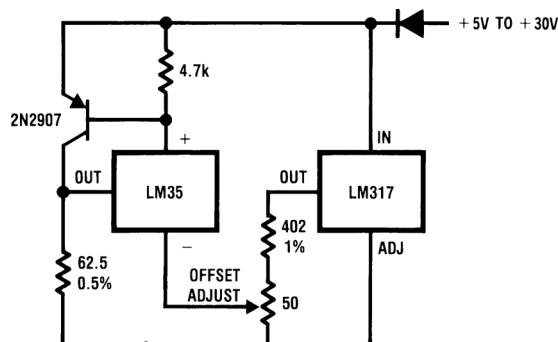


FIGURE 9. 4-To-20 mA Current Source (0°C to +100°C)

Typical Applications (Continued)

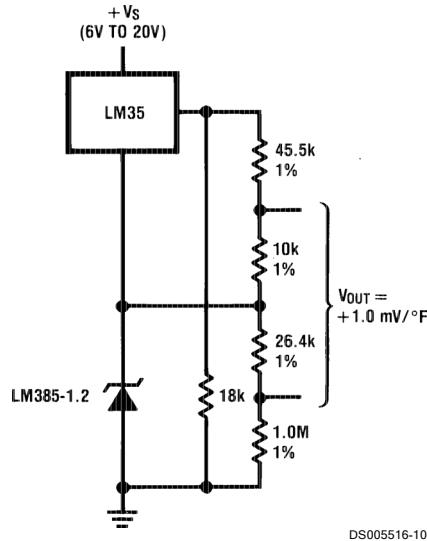


FIGURE 10. Fahrenheit Thermometer

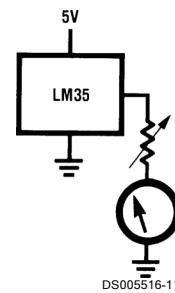
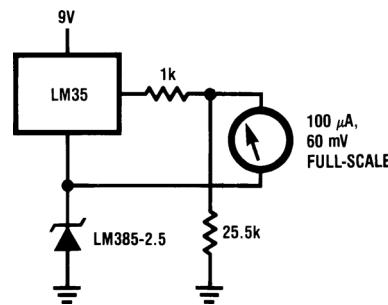


FIGURE 11. Centigrade Thermometer (Analog Meter)



**FIGURE 12. Fahrenheit Thermometer Expanded Scale Thermometer
(50° to 80° Fahrenheit, for Example Shown)**

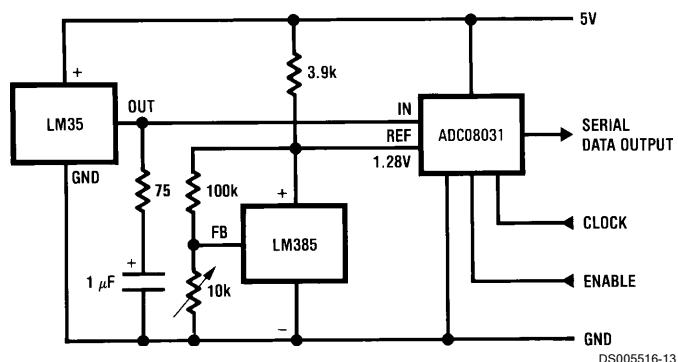


FIGURE 13. Temperature To Digital Converter (Serial Output) (+128°C Full Scale)

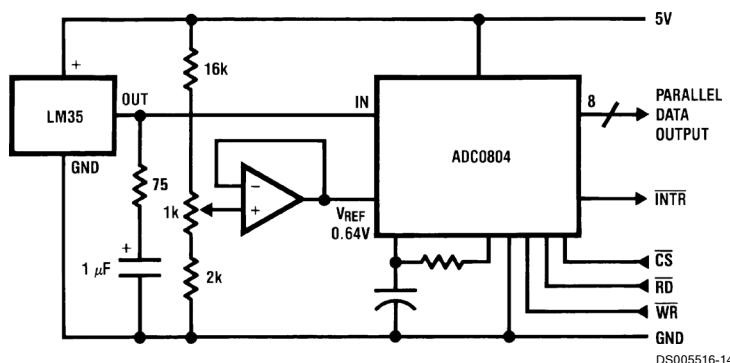
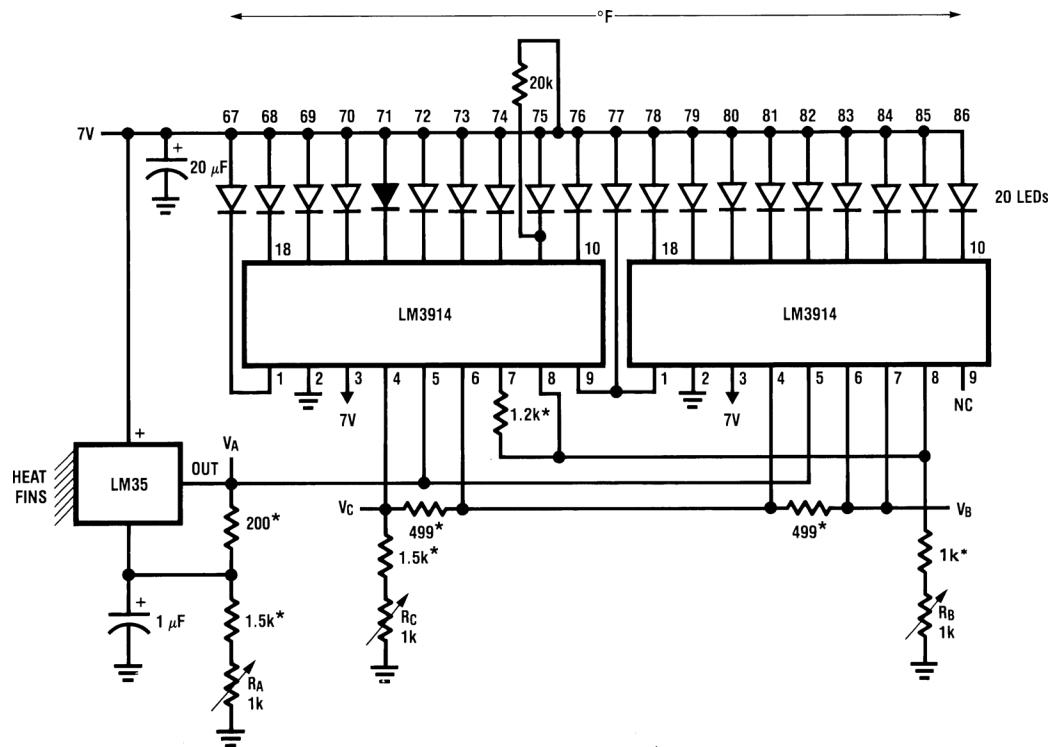


FIGURE 14. Temperature To Digital Converter (Parallel TRI-STATE™ Outputs for Standard Data Bus to μP Interface) (128°C Full Scale)

Typical Applications (Continued)

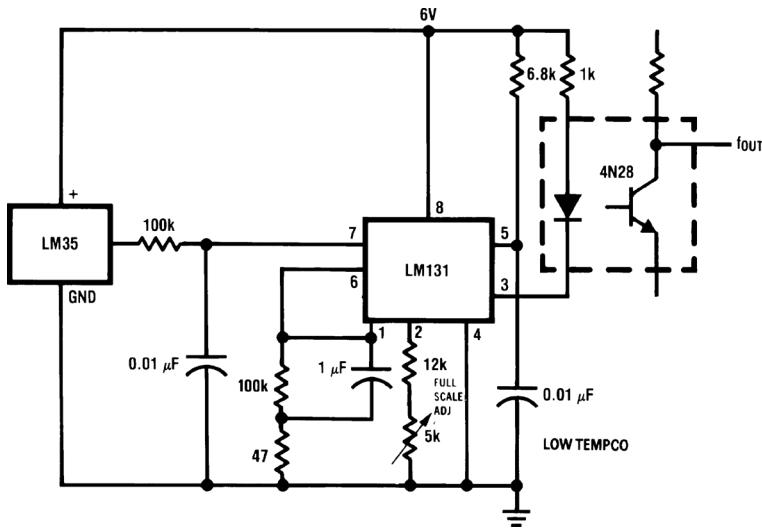


DS005516-16

*=1% or 2% film resistor

Trim R_B for V_B=3.075VTrim R_C for V_C=1.955VTrim R_A for V_A=0.075V + 100mV/°C × TambientExample, V_A=2.275V at 22°C

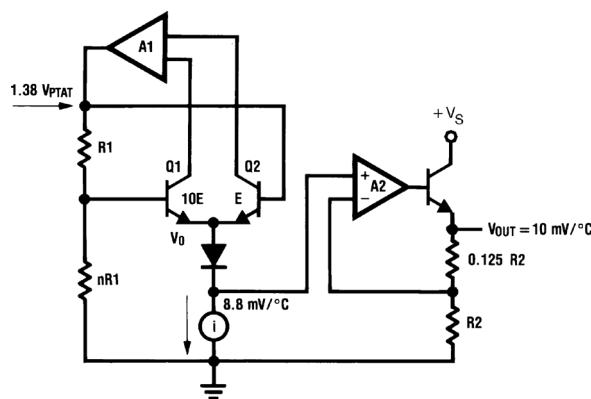
FIGURE 15. Bar-Graph Temperature Display (Dot Mode)



DS005516-15

**FIGURE 16. LM35 With Voltage-To-Frequency Converter And Isolated Output
(2°C to +150°C; 20 Hz to 1500 Hz)**

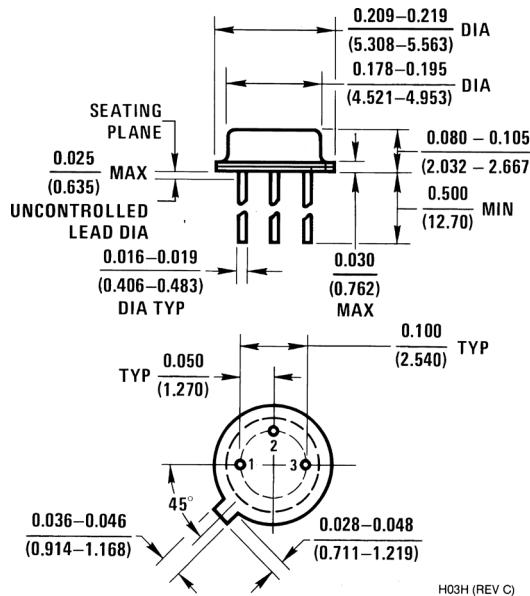
Block Diagram



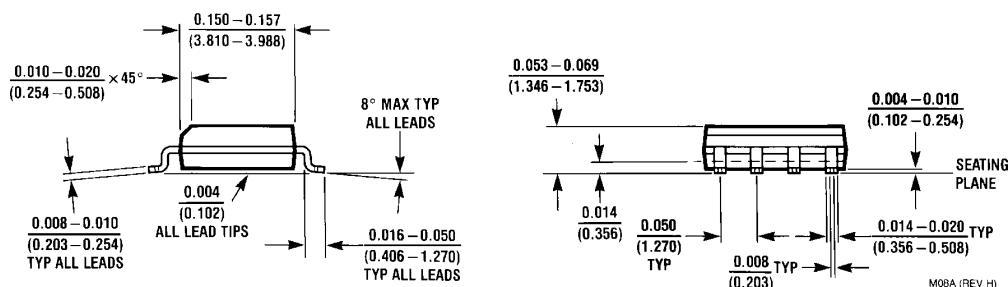
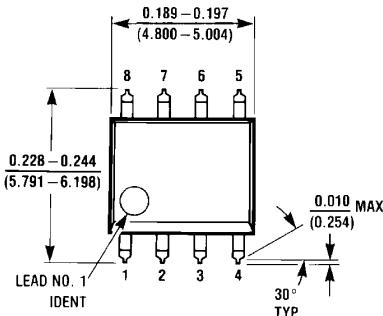
DS005516-23

Physical Dimensions

inches (millimeters) unless otherwise noted

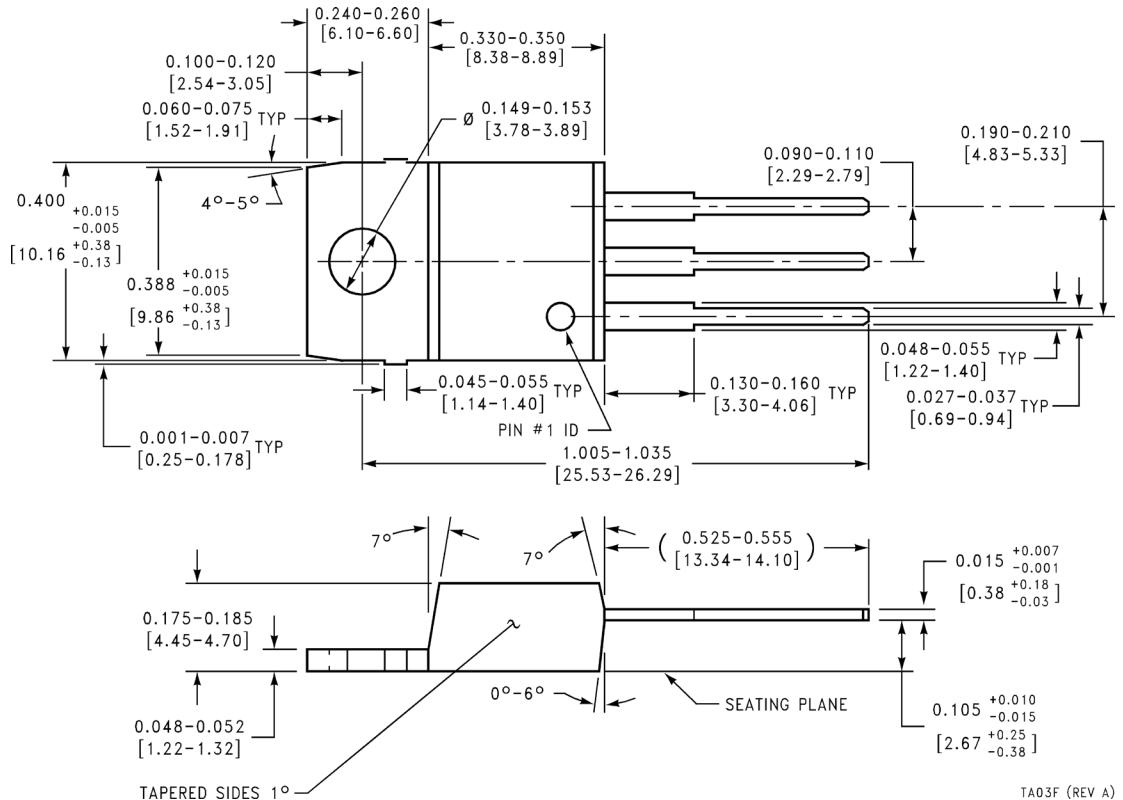


TO-46 Metal Can Package (H)
Order Number LM35H, LM35AH, LM35CH,
LM35CAH, or LM35DH
NS Package Number H03H



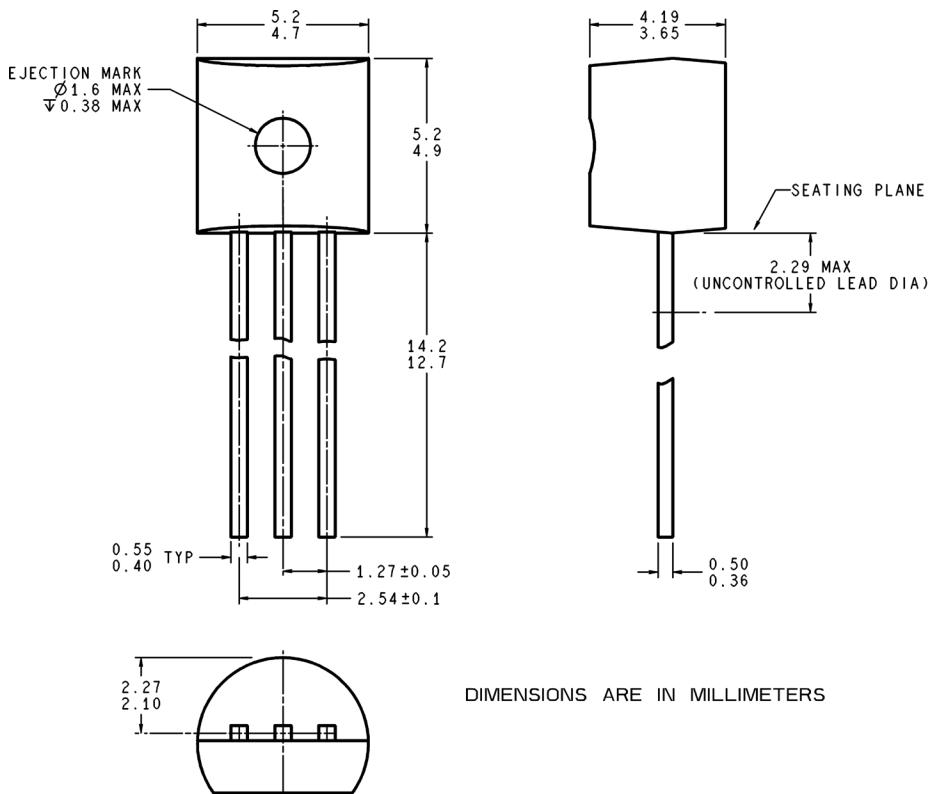
SO-8 Molded Small Outline Package (M)
Order Number LM35DM
NS Package Number M08A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



**Power Package TO-220 (T)
Order Number LM35DT
NS Package Number TA03F**

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



TO-92 Plastic Package (Z)
Order Number LM35CZ, LM35CAZ or LM35DZ
NS Package Number Z03A

Z03A (Rev. G)

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.



National Semiconductor Corporation
 Americas
 Tel: 1-800-272-9959
 Fax: 1-800-737-7018
 Email: support@nsc.com
www.national.com

National Semiconductor Europe
 Fax: +49 (0) 180-530 85 86
 Email: europe.support@nsc.com
 Deutsch Tel: +49 (0) 69 9508 6208
 English Tel: +44 (0) 870 24 0 2171
 Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
 Tel: 65-2544466
 Fax: 65-2504466
 Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
 Tel: 81-3-5639-7560
 Fax: 81-3-5639-7507

Temperature measurement

To use it for the correction of the sensor which a characteristic turns into by temperature.

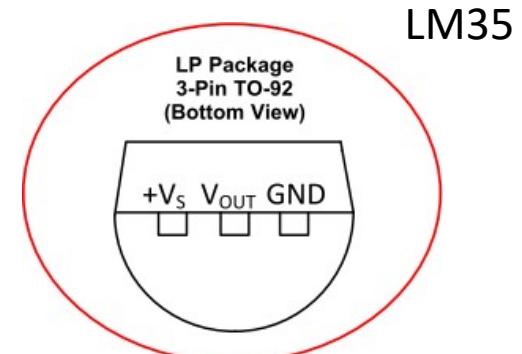
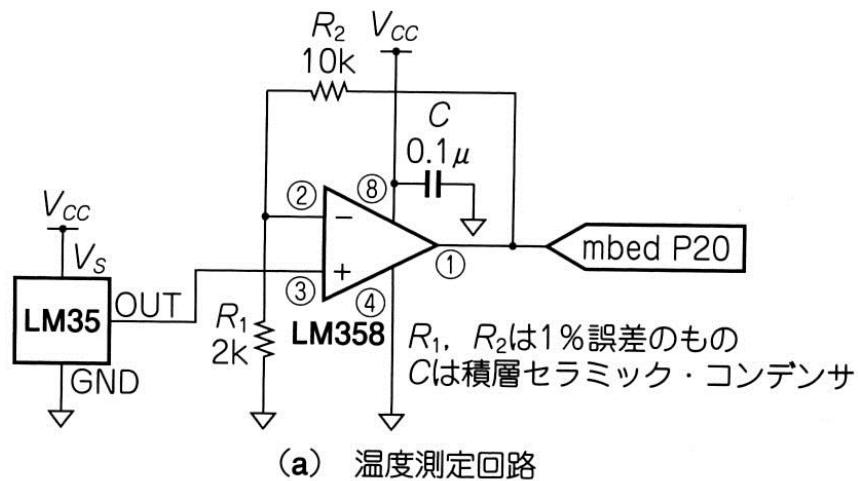
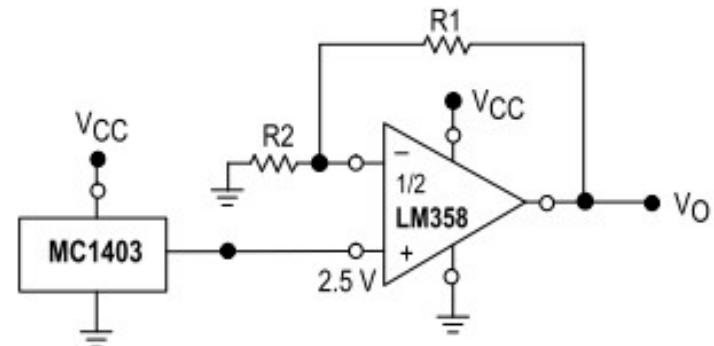
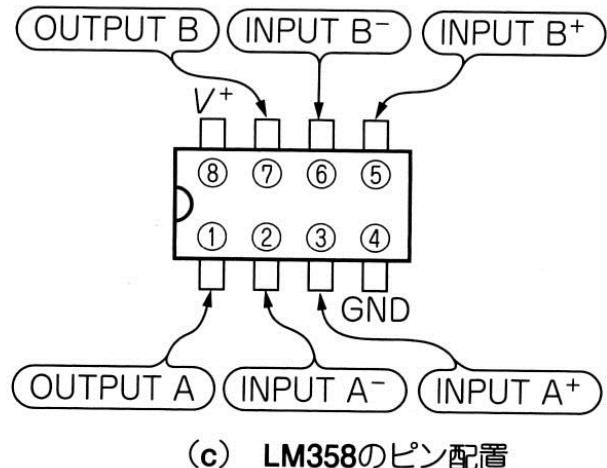
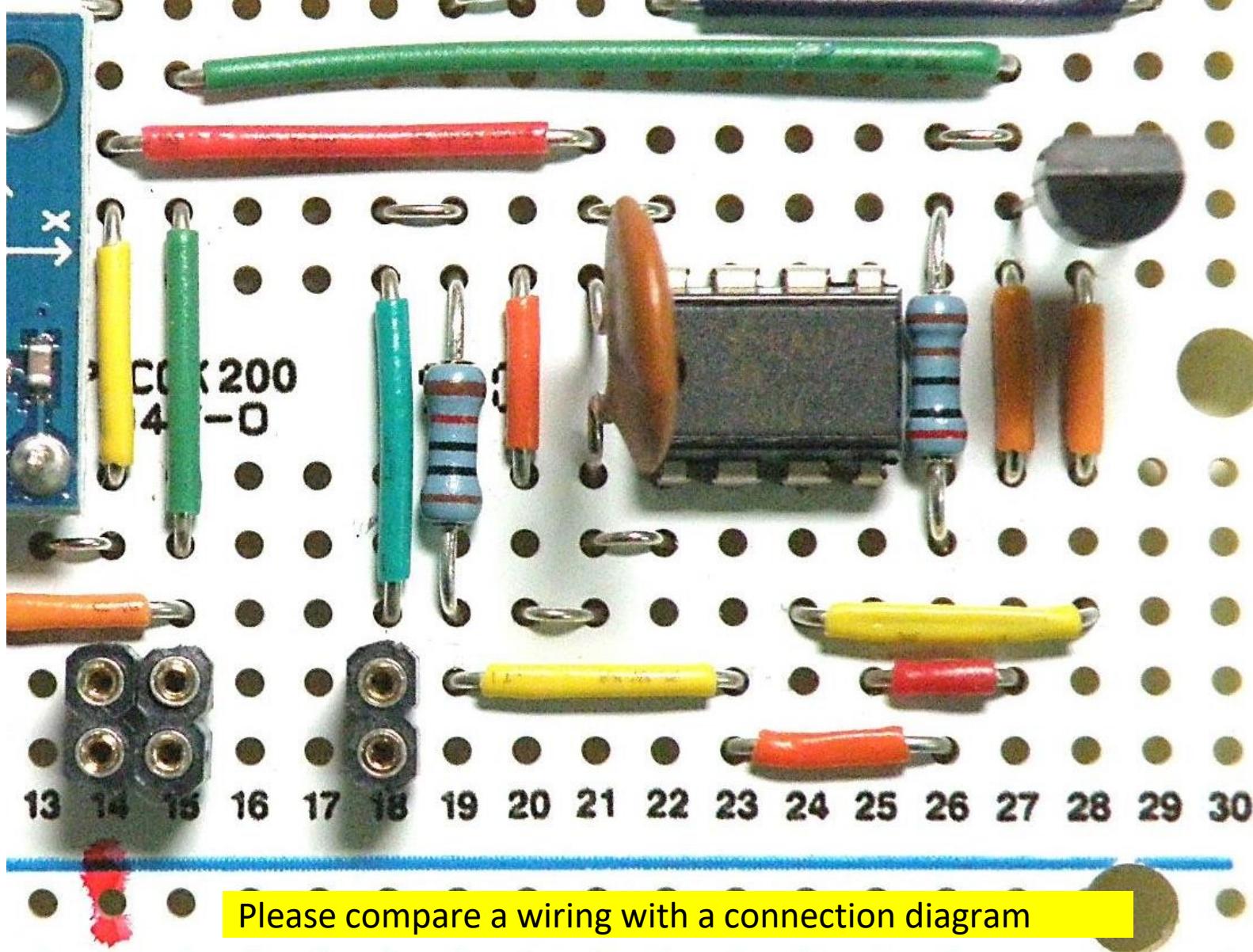


Figure 7. Voltage Reference



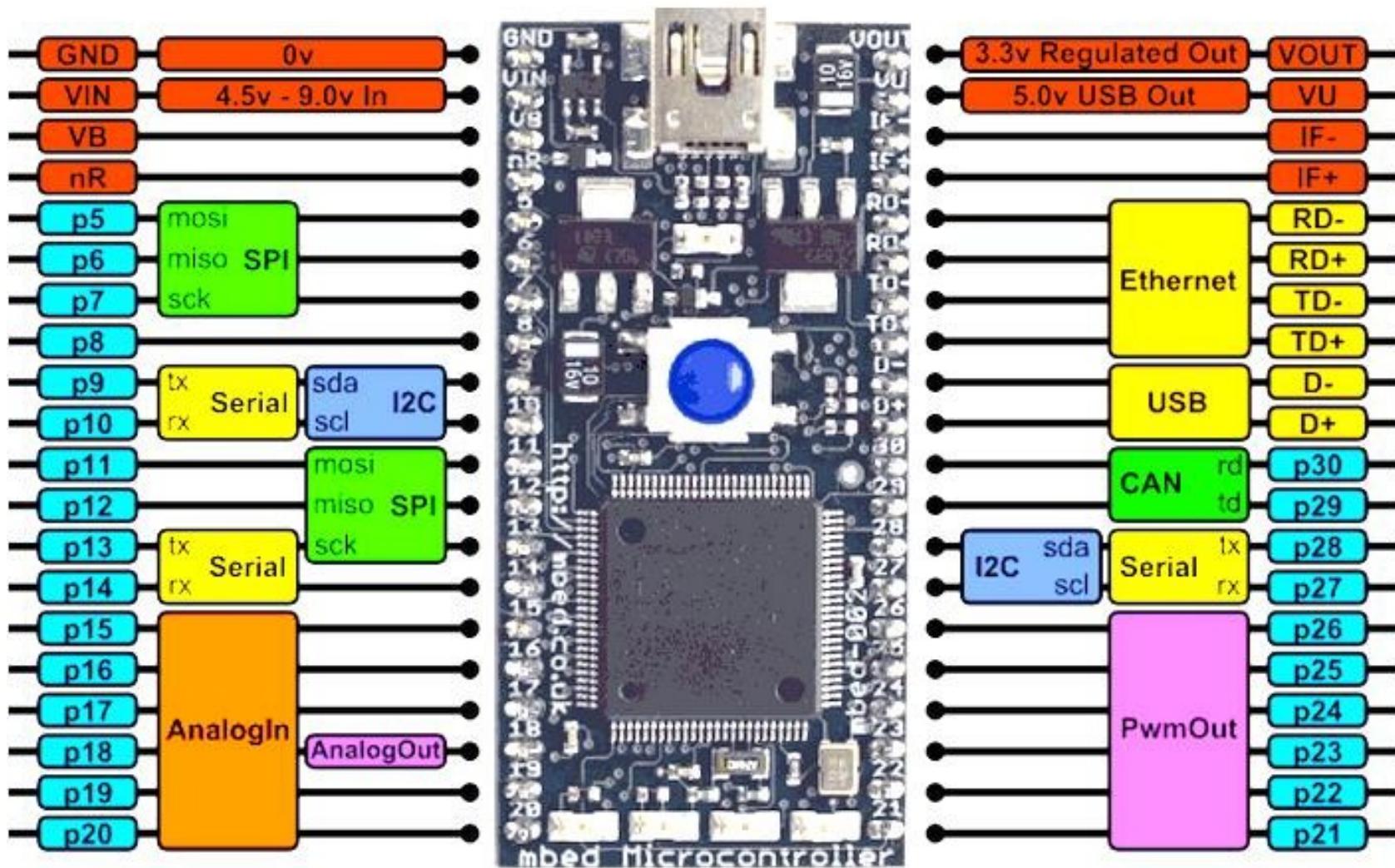
$$V_O = 2.5 V \left(1 + \frac{R_1}{R_2}\right)$$

13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30



Please compare a wiring with a connection diagram

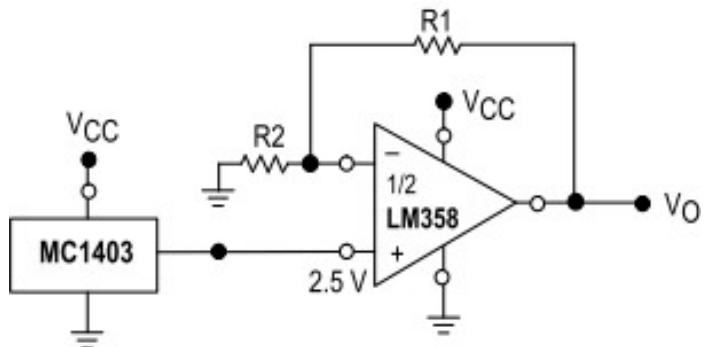
Connect the output to P20 of LPC1768.



if we take out the voltage, we can understand temperature.
 Ex. 0.25V → 25 degrees Celsius

$$\begin{aligned} R1 &= 10K, R2=2K \\ V_o &= V_{in} * (R1+R2)/R2 \\ &= V_{in} * 6 ; \end{aligned}$$

Figure 7. Voltage Reference



$$V_o = 2.5V \left(1 + \frac{R1}{R2}\right)$$

$$\begin{aligned} \text{Ex. at 25 degree} \\ V_{in} &= 0.25V \\ V_o &= V_{in} * 6 = 0.25 * 6 = 1.5V \end{aligned}$$

$$\begin{aligned} \text{at LPC1768} \\ 3.3V_{max} &\rightarrow 1.0_{max} \\ V_o &\rightarrow y \end{aligned}$$

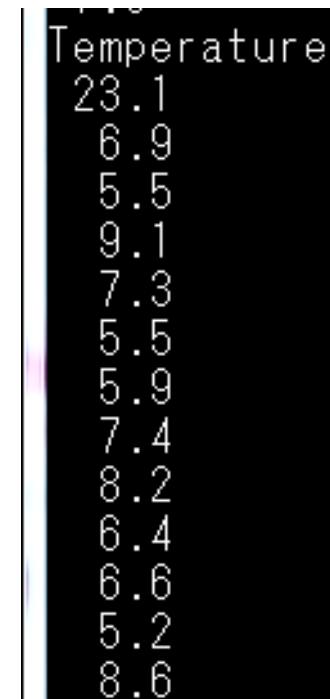
Calculate V_{in} from a value of y

$$\begin{aligned} 1.0 * V_o &= V_{in} * 6 = 3.3 * y \\ V_{in} &= 3.3 * y / 6 //\text{Ex. } 0.25V \\ V_{in} * &= 100 //\text{Ex. } 25 \text{ degree} \end{aligned}$$

```
AnalogIn ain(p20);
Serial pc(USBTX, USBRX);

void main()
{
    int n = 0;
    bool sw = false;
    pc.printf("Temperature\n");
    while(1) {
        double tmp = ain * 55 ; //3.3 * 100.0 / 6 = 55
        pc.printf("% 4.1f\n",tmp);
    }
}
```

Terminal emulator on the PC



A terminal window displaying a list of temperatures. The window has a dark background and light-colored text. The text is organized into two columns: a header column on the left and a data column on the right. The header column contains the word "Temperature" followed by a series of numbers. The data column contains a series of numbers, each preceded by a percentage sign and a space, indicating a value of 4.1 decimal places.

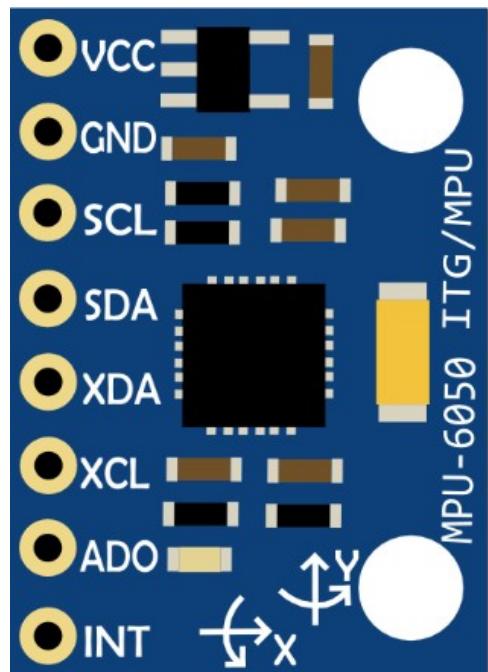
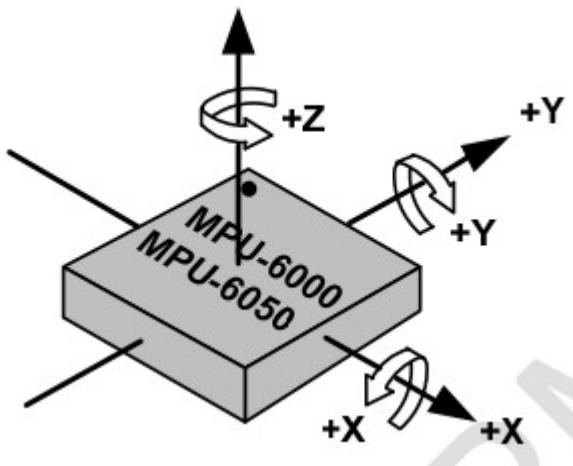
Temperature	
23.1	
6.9	
5.5	
9.1	
7.3	
5.5	
5.9	
7.4	
8.2	
6.4	
6.6	
5.2	
8.6	

How to display it by a simple graph?

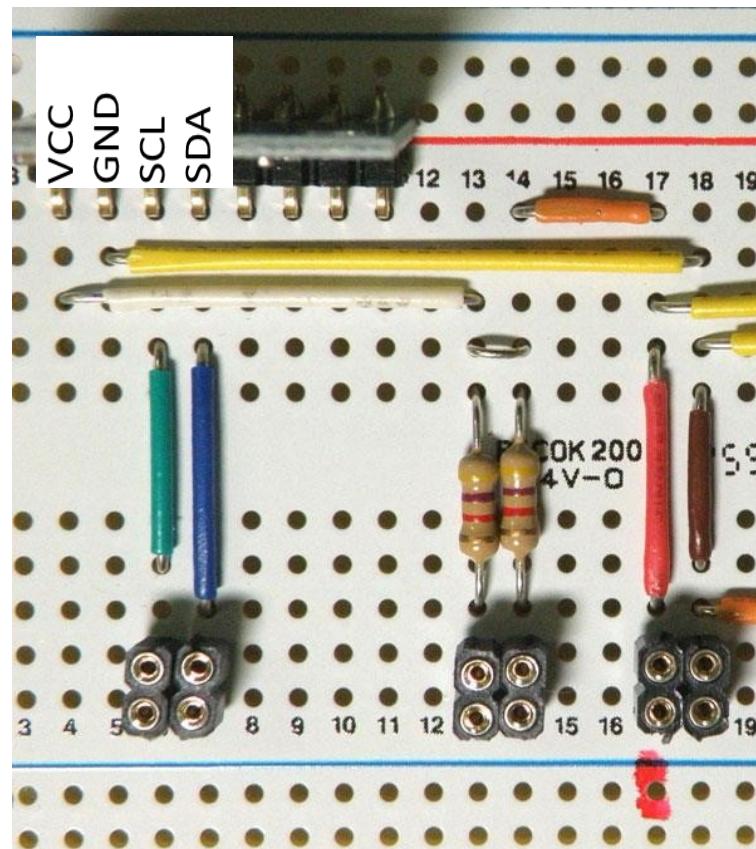
Terminal emulator on the PC

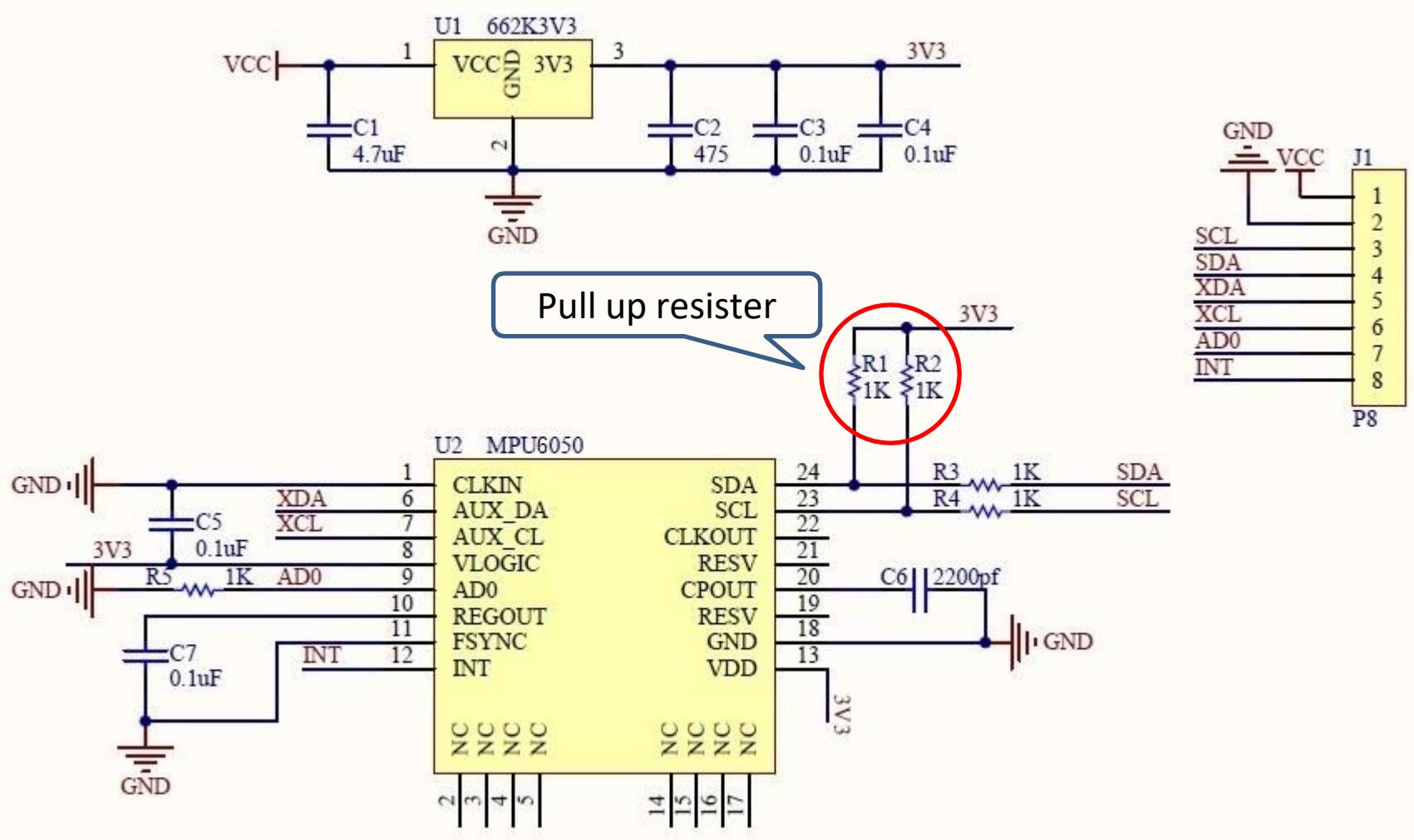
8.0	*
7.3	*
6.8	*
7.0	*
5.9	*
5.4	*
4.8	*
8.8	*
8.0	*
7.0	*
6.1	*
6.6	*

```
int main()
{
    char string[80];
    char txtBuf[18];
    int ctr = 0;
    strcpy( string, "Temperature");
    pc.printf("%s\r\n",string);
    wait_ms( 5 ); //5ms
    while(1) {
        double tmp = ain * 55;//3.3 * 100.0 / 6 = 55
        //For LCD
        sprintf(txtBuf,"Temp=% .1f",tmp);
        LcdString2(0,txtBuf);
        m = (int)((tmp - 30.0) * 5.0); //For PC
        txtBuf[0] = 0;
        for (; m > 0; m--)
            strcat(txtBuf, "=");
        LcdString2(1,txtBuf);
        sprintf(string, "% 3d: %4.1f ", ctr++, tmp);
        if ( ctr >= 100 )
            ctr = 0;
        m = (int)((tmp - 25.0) * 5.0);
        for (; m > 0; m--)
            strcat(string, "=");
        pc.printf("%s\r\n",string);
    }
}
```



Accelerometer + Gyroscope
MPU6050





Addr (Hex)	Addr (Dec.)	Register Name	
3A	58	INT_STATUS	
3B	59	ACCEL_XOUT_H	ACCEL_XOUT[15:8]
3C	60	ACCEL_XOUT_L	ACCEL_XOUT[7:0]
3D	61	ACCEL_YOUT_H	ACCEL_YOUT[15:8]
3E	62	ACCEL_YOUT_L	ACCEL_YOUT[7:0]
3F	63	ACCEL_ZOUT_H	ACCEL_ZOUT[15:8]
40	64	ACCEL_ZOUT_L	ACCEL_ZOUT[7:0]
41	65	TEMP_OUT_H	TEMP_OUT[15:8]
42	66	TEMP_OUT_L	TEMP_OUT[7:0]
43	67	GYRO_XOUT_H	GYRO_XOUT[15:8]
44	68	GYRO_XOUT_L	GYRO_XOUT[7:0]
45	69	GYRO_YOUT_H	GYRO_YOUT[15:8]
46	70	GYRO_YOUT_L	GYRO_YOUT[7:0]
47	71	GYRO_ZOUT_H	GYRO_ZOUT[15:8]
48	72	GYRO_ZOUT_L	GYRO_ZOUT[7:0]
75	117	WHO_AM_I	WHO_AM_I[6:1]



Accelerometer



Temperature



Gyroscope



ID

4.30 Register 107 – Power Management 1

PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Upon power up, the MPU-60X0 clock source defaults to the internal oscillator. However, it is highly recommended that the device be configured to use one of the gyroscopes (or an external clock source) as the clock reference for improved stability. The clock source can be selected according to the following table.

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

The reset value is 0x00 for all registers other than the registers below.

- Register 107: 0x40.
- Register 117: 0x68.

4.4 Register 27 – Gyroscope Configuration

GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]	-	-	-	-

FS_SEL selects the full scale range of the gyroscope outputs according to the following table.

FS_SEL	Full Scale Range
0	$\pm 250 \text{ } ^\circ/\text{s}$
1	$\pm 500 \text{ } ^\circ/\text{s}$
2	$\pm 1000 \text{ } ^\circ/\text{s}$
3	$\pm 2000 \text{ } ^\circ/\text{s}$

4.5 Register 28 – Accelerometer Configuration

ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]	-	-	-	-

AFS_SEL selects the full scale range of the accelerometer outputs according to the following table.

AFS_SEL	Full Scale Range
0	$\pm 2g$
1	$\pm 4g$
2	$\pm 8g$
3	$\pm 16g$

4.2 Register 25 – Sample Rate Divider SMPLRT_DIV

Type: Read/Write

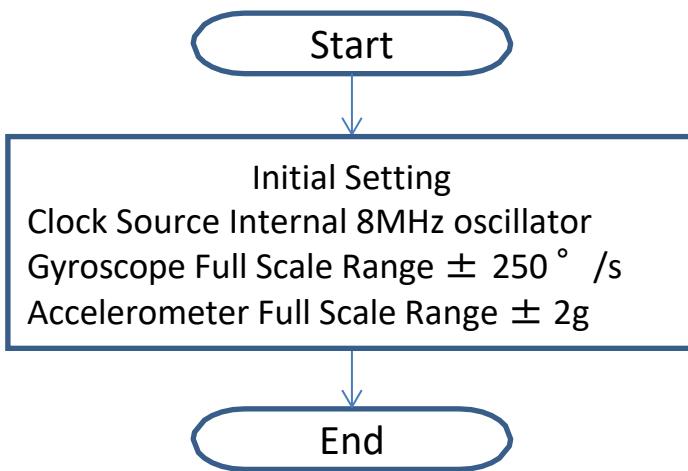
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25								SMPLRT_DIV[7:0]

The Sample Rate is generated by dividing the gyroscope output rate by *SMPLRT_DIV*:

$$\text{Sample Rate} = \text{Gyroscope Output Rate} / (1 + \text{SMPLRT_DIV})$$

where Gyroscope Output Rate = 8kHz when the DLPF is disabled (*DLPF_CFG* = 0 or 7), and 1kHz when the DLPF is enabled (see Register 26).

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.



```
#include "mbed.h"
#include "Gyro.h"

I2C i2cGyro(p9, p10);

static const int devAddr = 0xD0;

void InitGyro()
{
    char data[3] ;
    data[0] = 0x00;
    data[1] = 0x00;
    data[2] = 0x00;
    WriteBytes(0x6B, 1, &data[0]); //Internal 8MHz
    oscillator
    WriteBytes(0x1B, 1, &data[1]); //Gyroscope Full
    Scale Range  $\pm 250^\circ$  /s
    WriteBytes(0x1C, 1, &data[2]); //Accelerometer
    Full Scale Range  $\pm 2g$ 
}
```

4.34 Register 117 – Who Am I

WHO_AM_I

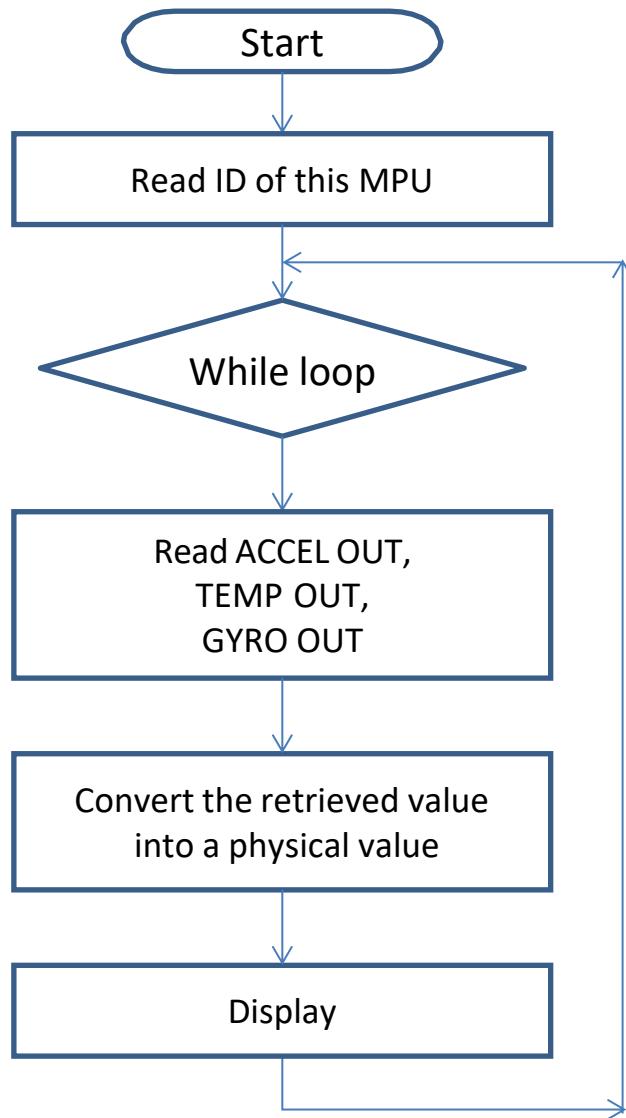
Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-				WHO_AM_I[6:1]			-

Description:

This register is used to verify **the identity of the device**. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I²C address. The least significant bit of the MPU-60X0's I²C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

The default value of the register is 0x68.



```

//Convert char to int
int convInt(char data[])
{
    short n = data[0];//Should be "short"
    n <= 8;
    n |= data[1];
    return n;
}

bool flag = ReadBytes(0x75, 1, data);
if(flag == true) {
    sprintf(txtBuf,"ID=%x",data[0]);
    LcdString(0,txtBuf);
} else {
    sprintf(txtBuf,"MPU6050 failed");
    LcdString(0,txtBuf);
}
pc.printf("%s\r\n",txtBuf);
  
```

4.18 Registers 59 to 64 – Accelerometer Measurements

ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L

Type: Read Only

4.19 Registers 65 and 66 – Temperature Measurement

TEMP OUT H and TEMP OUT L

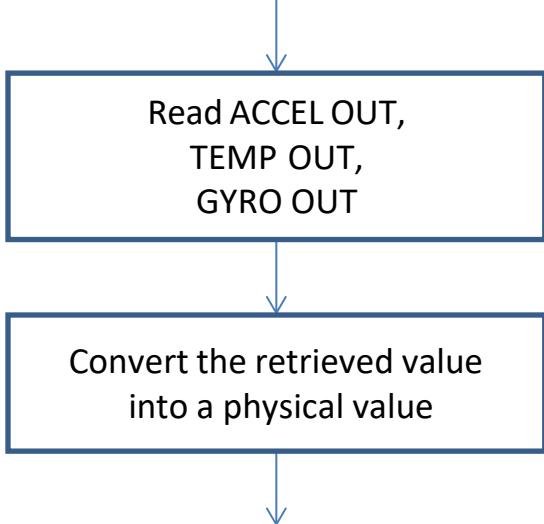
Type: Read Only

4.20 Registers 67 to 72 – Gyroscope Measurements

`GYRO_XOUT_H`, `GYRO_XOUT_L`, `GYRO_YOUT_H`, `GYRO_YOUT_L`, `GYRO_ZOUT_H`, and `GYRO_ZOUT_L`

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67					GYRO_XOUT[15:8]			
44	68					GYRO_XOUT[7:0]			
45	69					GYRO_YOUT[15:8]			
46	70					GYRO_YOUT[7:0]			
47	71					GYRO_ZOUT[15:8]			
48	72					GYRO_ZOUT[7:0]			



```
int ax, ay, az;
int gx, gy, gz;
int tp;
wait(2);
while(1) {
    //Read ACCEL OUT, TEMP OUT, GYRO OUT
    if ( ReadBytes(0x3B, 14, data) == true ) {

        //Accelerometer
        ax = convInt(&data[0]);
        ay = convInt(&data[2]);
        az = convInt(&data[4]);
        //Temperature
        tp = convInt(&data[6]);
        //Gyroscope
        gx = convInt(&data[8]);
        gy = convInt(&data[10]);
        gz = convInt(&data[12]);
        //For LCD
        sprintf(txtBuf,"x%6d y%6d",gx,gy);
        LcdString2(0,txtBuf);
```

the accelerometers' sensitivity per LSB

```
//  
double dax = (double)ax / 16384.0; //LSB Sensitivity 16384 LSB/g  
double day = (double)ay / 16384.0;  
double daz = (double)az / 16384.0;
```

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 2g$	16384 LSB/g
1	$\pm 4g$	8192 LSB/g
2	$\pm 8g$	4096 LSB/g
3	$\pm 16g$	2048 LSB/g

the gyroscopes' sensitivity per LSB

```
//  
double dgx = (double)gx / 131.0; //LSB Sensitivity 131 LSB/°/s  
double dgy = (double)gy / 131.0;  
double dgz = (double)gz / 131.0;
```

FS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 250 \text{ } ^\circ/\text{s}$	131 LSB/°/s
1	$\pm 500 \text{ } ^\circ/\text{s}$	65.5 LSB/°/s
2	$\pm 1000 \text{ } ^\circ/\text{s}$	32.8 LSB/°/s
3	$\pm 2000 \text{ } ^\circ/\text{s}$	16.4 LSB/°/s

4.19 Registers 65 and 66 – Temperature Measurement

TEMP_OUT_H and TEMP_OUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65								TEMP_OUT[15:8]
42	66								TEMP_OUT[7:0]

The temperature in degrees C for a given register value may be computed as:

$$\text{Temperature in degrees C} = (\text{TEMP_OUT Register Value as a signed quantity})/340 + 36.53$$

PARAMETER	CONDITIONS	TYP	Units
TEMPERATURE SENSOR			
Range		-40 to +85	°C
Sensitivity	Untrimmed	340	LSB/°C
Temperature Offset	35°C	-521	LSB
Linearity	Best fit straight line (-40°C to +85°C)	±1	°C

$$y - y_1 = m(x - x_1)$$

$$y_1 = 35, x_1 = -521, m = 1/340$$

$$y = (1/340) * (x + 521) + 35$$

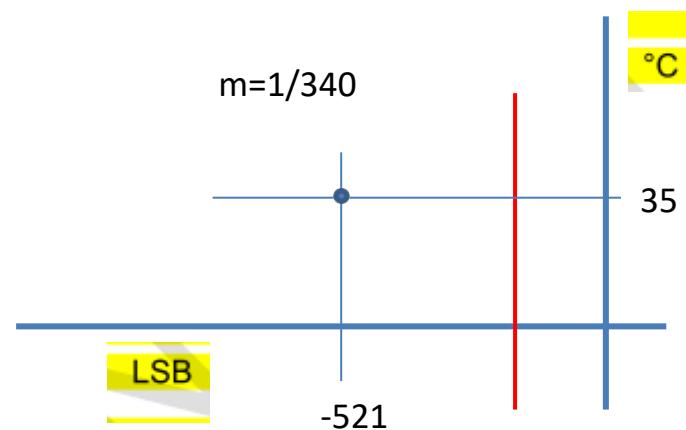
$$y = (1/340) * x + (521/340) + 35$$

$$y = (1/340) * x + 36.53$$

$$m = 1/340$$

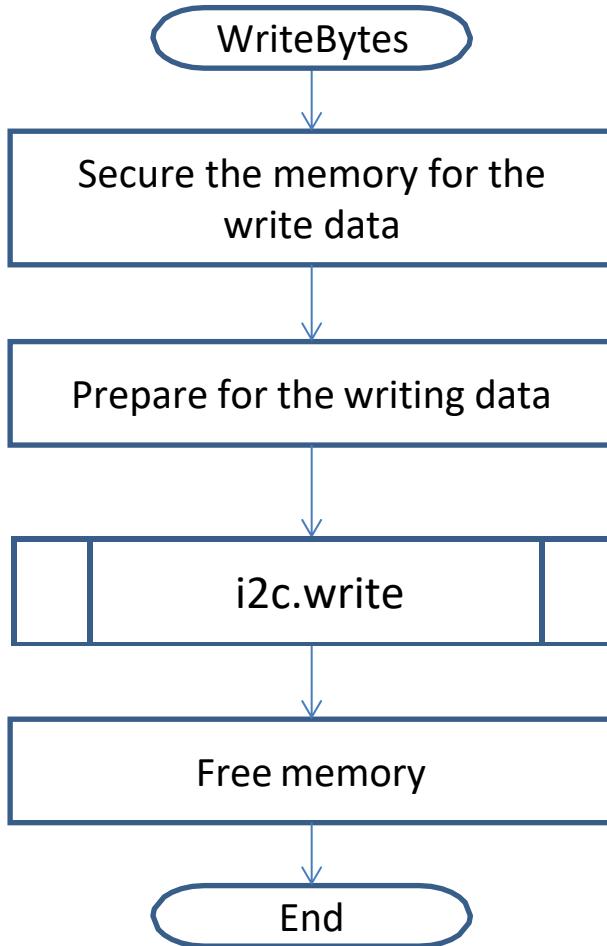
°C

35

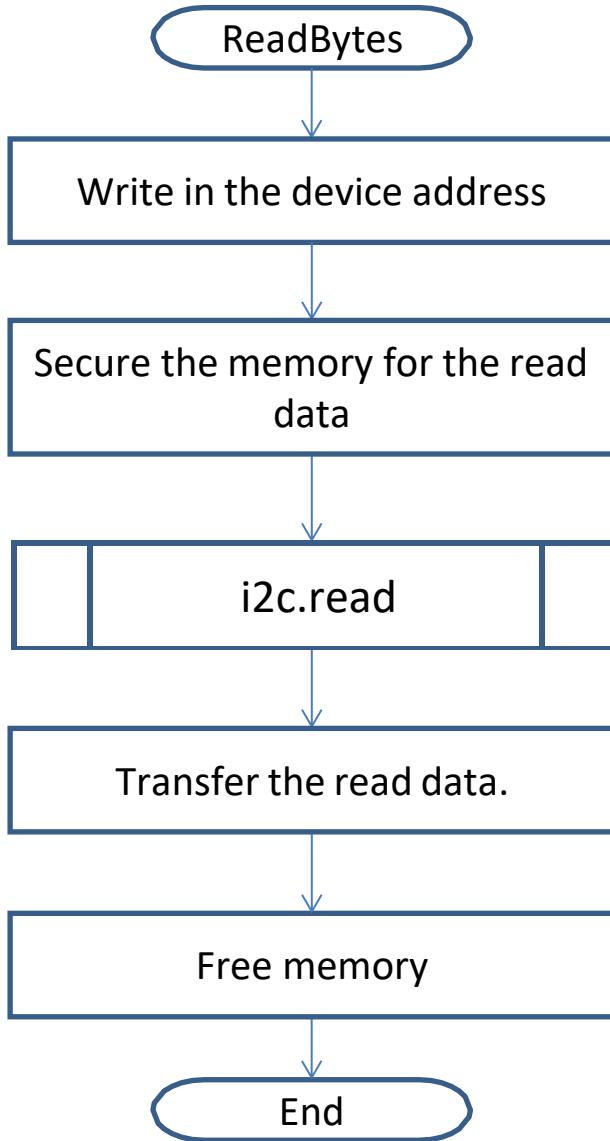


Convert the retrieved value
into a physical value

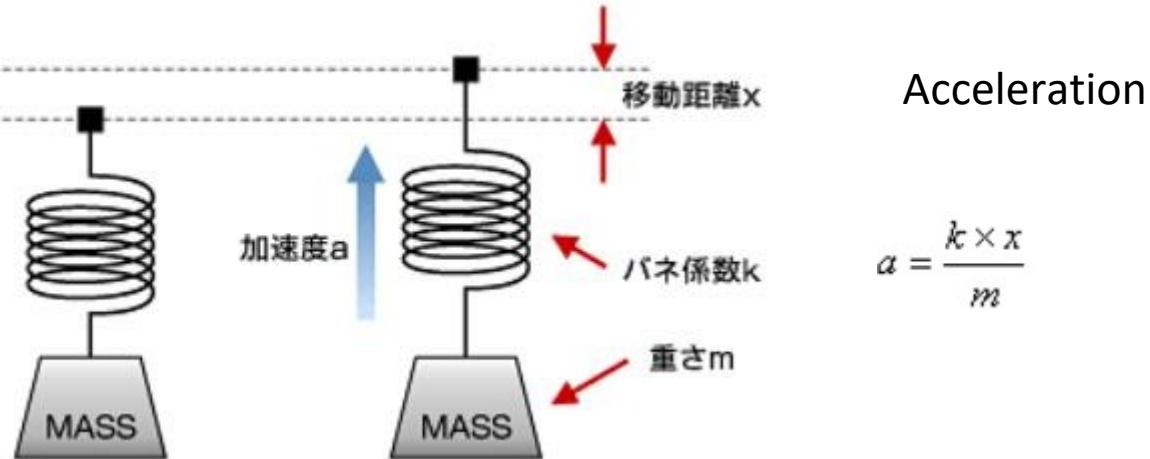
```
//Temperature in degrees C = (TEMP_OUT Register Value as a
signed quantity)/340 + 36.53
double temp = tp / 340.0 + 36.53 ;
//For LCD
sprintf(txtBuf,"z%6d t%6.2f",gz,temp);
LcdString2(1,txtBuf);
//Change to g value
double dax = (double)ax / accelsens;
double day = (double)ay / accelsens;
double daz = (double)az / accelsens;
pc.printf("ax=%6.2f ay=%6.2f az=%6.2f ",dax,day,daz);
//Change to degree value
double dgx = (double)gx / gyrosens;
double dgy = (double)gy / gyrosens;
double dgz = (double)gz / gyrosens;
pc.printf("gx=%6.1f gy=%6.1f gz=%6.1f ",dgx,dgy,dgz);
//Temperature
pc.printf("temp=%2f\r\n",temp);
```



```
bool WriteBytes(int regAddr, int byteLen, char *ptr)
{
    char *writeData = (char*)malloc(byteLen+1);
    char *s = writeData;
    *s++ = regAddr;
    for ( int n = 0 ; n < byteLen; n++) {
        *s++ = *ptr++;
    }
    if (i2cGyro.write(devAddr, writeData, byteLen+1, false) != 0 )
    {
        free( writeData);
        return false;
    } else {
        free( writeData);
        return true;
    }
}
```



```
bool ReadBytes(int regAddr, int byteLen, char *ptr)
{
    char cmd[1];
    cmd[0] = regAddr;
    if (i2cGyro.write(devAddr, cmd, 1, true) != 0 )
        return false;
    char *readData = (char*)malloc(byteLen);
    if (i2cGyro.read(devAddr, readData, byteLen) != 0 ) {
        free (readData);
        return false;
    }
    for(int i =0; i < byteLen; i++) {
        *ptr++ = readData[i];
    }
    free (readData);
    return true;
}
```



Acceleration

$$a = \frac{k \times x}{m}$$

C1とC2の静電容量から
加速度を検出

Capacitance type acceleration sensor

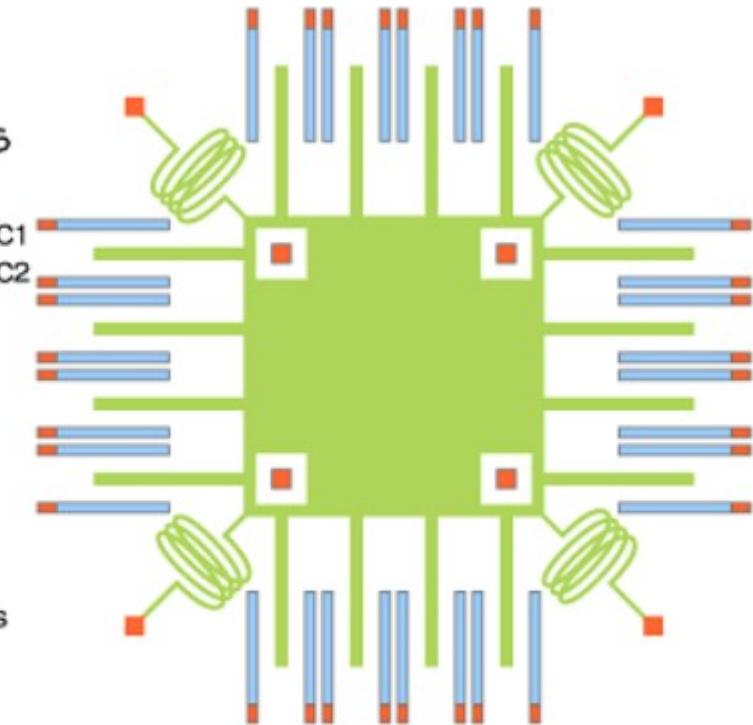
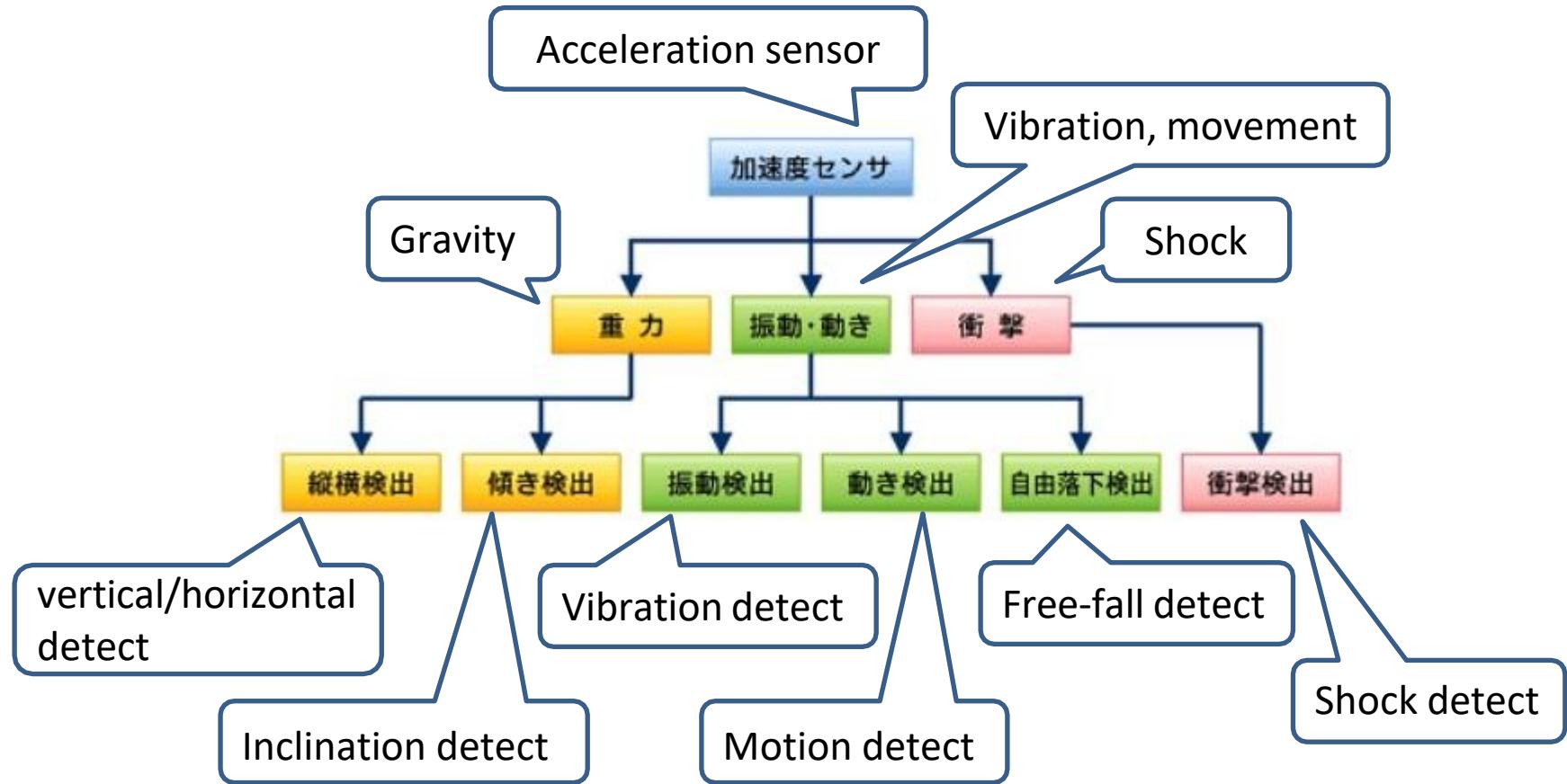


図2 静電容量型加速度センサの検出素子部模式図（参考：アナログ・デバイセズの加速度センサ）



Gyroscope $^{\circ}/s$ = deg/s = dps

± 2000 dps

Gaming



Gaming

± 500 dps

Pointing devices



Pointing devices

± 245 dps

Navigation



Navigation

± 100 dps ~ ± 500 dps

± 200 dps

Image
stabilization
(2-axis)



Image stabilization
150dps

± 65 dps

It is necessary to plan electric power saving practically using sleep mode and FIFO buffer.

4.30 Register 107 – Power Management 1

PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

4.31 Register 108 – Power Management 2

PWR_MGMT_2

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6C	108	LP_WAKE_CTRL[1:0]	STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG	

4.32 Register 114 and 115 – FIFO Count Registers

FIFO_COUNT_H and FIFO_COUNT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
72	114					FIFO_COUNT[15:8]			
73	115					FIFO_COUNT[7:0]			

4.33 Register 116 – FIFO Read Write

FIFO_R_W

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
74	116					FIFO_DATA[7:0]			

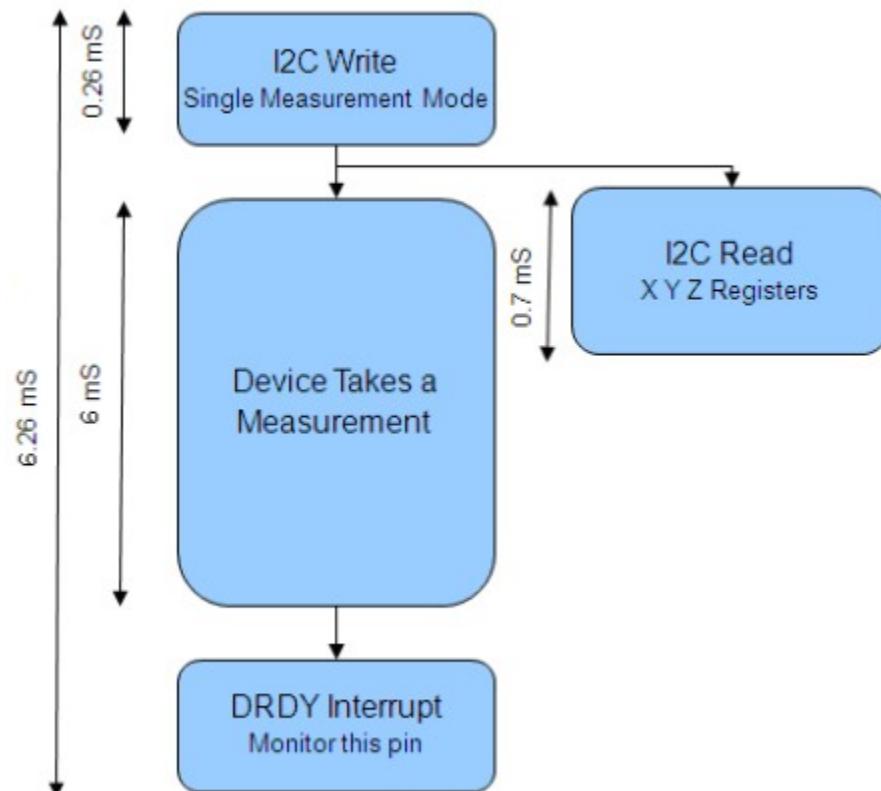
Normal Operating Current	Gyroscope + Accelerometer + DMP	3.9		mA
	Gyroscope + Accelerometer (DMP disabled)	3.8		mA
	Gyroscope + DMP (Accelerometer disabled)	3.7		mA
	Gyroscope only (DMP & Accelerometer disabled)	3.6		mA
	Accelerometer only (DMP & Gyroscope disabled)	500		µA
Accelerometer Low Power Mode Current	1 Hz update rate	10		µA
	5 Hz update rate	20		µA
	20 Hz update rate	70		µA
	40 Hz update rate	140		µA

```
//StandBy
data[0] = 0x28; //CYCLE=1,TEMP_DIS=1
data[1] = 0x3F; //1.25Hz, STB=1 ALL Standby
WriteBytes(0x6B, 2, &data[0]); //Power Management 1
```

```
//Wake up
data[0] = 0x00; //
data[1] = 0x00; //
WriteBytes(0x6B, 2, &data[0]); //Power Management 1
```

Output Rate (ODR)	Continuous Measurement Mode Single Measurement Mode	0.75		75	Hz
Measurement Period	From receiving command to data ready		6		ms
Turn-on Time	Ready for I2C commands Analog Circuit Ready for Measurements		200 50		μs ms

Typical Measurement Period in Single-Measurement Mode



* Monitoring of the DRDY Interrupt pin is only required if maximum output rate is desired.

Applied example of the gyroscope



ESCなし

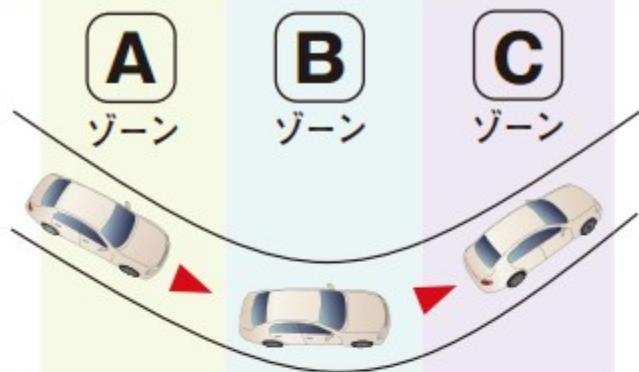


ESCあり

ESC: Electronic Stability Control)

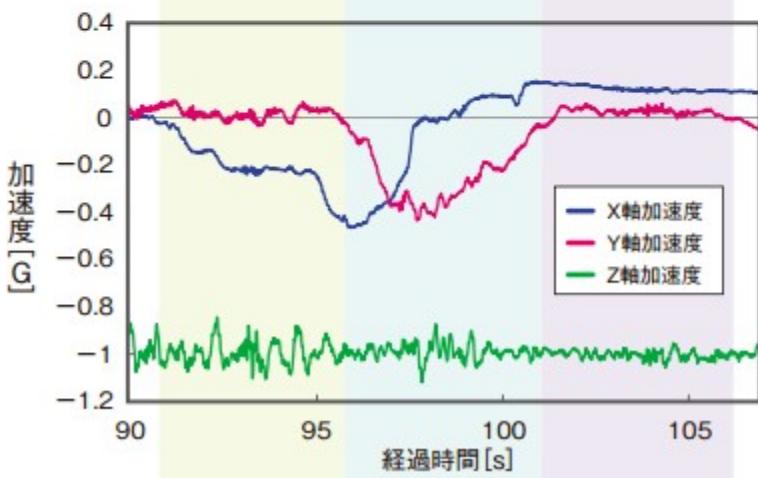
Rollover protector





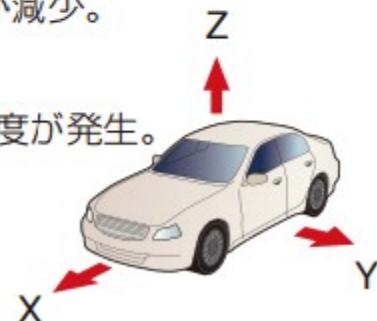
カーブ走行時は、車両にさまざまな力がかかります。VSAS-2GMで、カーブにさしかかるところから、曲がり終わって加速していく過程を追います。

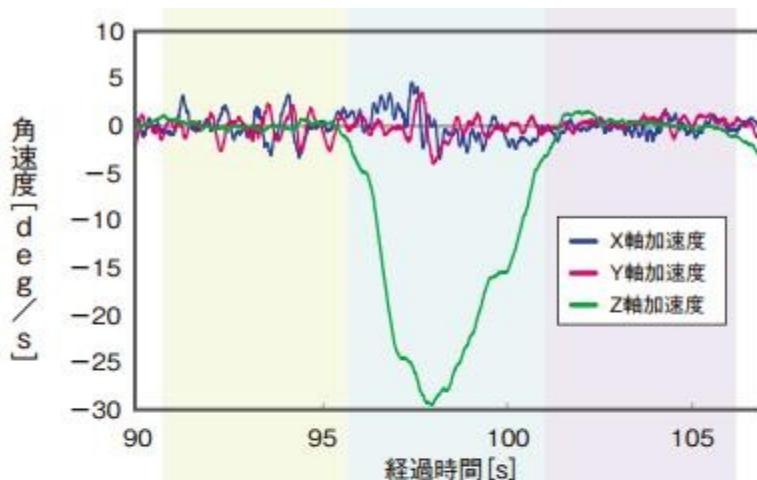
- Ⓐ ゾーン ブレーキをかけ、速度を落とす。
- Ⓑ ゾーン ハンドルを切り、大きく向きを変える。
- Ⓒ ゾーン 曲がる動作を完了しつつ、加速。



加速度

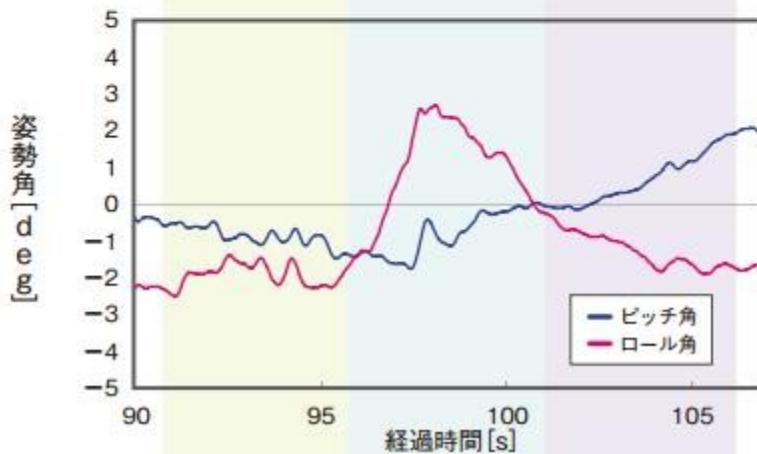
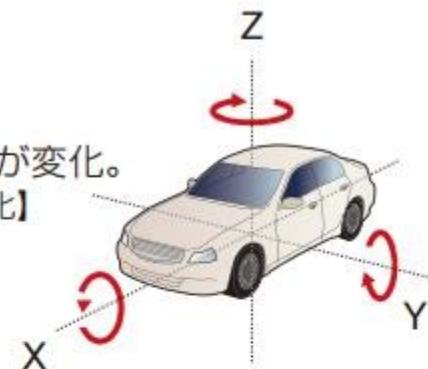
- Ⓐ ブレーキを踏み始め、スピードが減少。
【前後方向(X)加速度がマイナス】
- Ⓑ 曲がりはじめると横方向に加速度が発生。
【左右方向(Y)加速度が大きく変化】
- Ⓒ カーブが終了し、徐々に加速。
【前後方向(X)加速度がプラス】





角速度

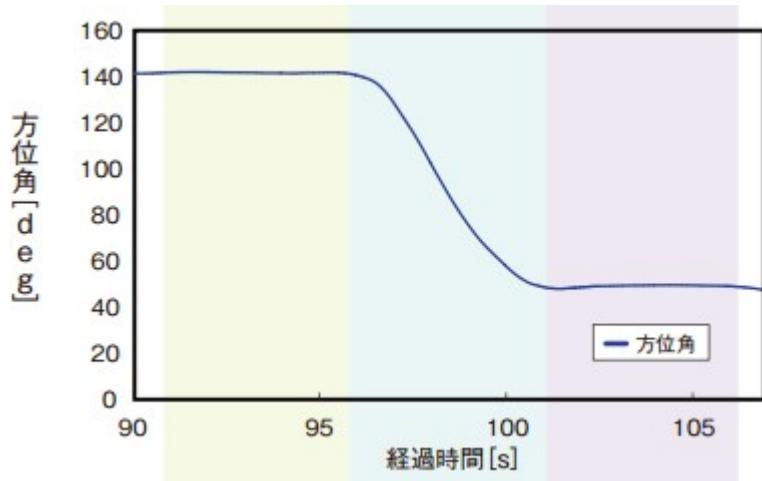
- A 直進走行。
- B ハンドルを切り、車体の向きが変化。
【左右方向(Z)角速度が大きく変化】
- C カーブが終了し、直進に移行。



姿勢角

- A ブレーキにより、前のめり状態。
【ピッチ角が変化】
- B カーブの遠心力により、車体が横に傾斜。
【ロール角が変化】
- C カーブ終了。加速を開始し、車体はわずかに頭をあげる。
【ピッチ角が変化】





方位角

- A 直進走行。
- B カーブで向きを変える。
- C カーブ終了。直進走行。

※方位角は真北を 0deg とする





InvenSense Inc.
1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A.
Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104
Website: www.invensense.com

Document Number: PS-MPU-6000A-00
Revision: 3.2
Release Date: 11/16/2011

MPU-6000 and MPU-6050
Product Specification
Revision 3.2

ADVANCE INFORMATION



CONTENTS

1 REVISION HISTORY	5
2 PURPOSE AND SCOPE	5
3 PRODUCT OVERVIEW	6
3.1 MPU-60X0 OVERVIEW.....	7
4 APPLICATIONS.....	9
5 FEATURES.....	10
5.1 GYROSCOPE FEATURES.....	10
5.2 ACCELEROMETER FEATURES.....	10
5.3 ADDITIONAL FEATURES.....	10
5.4 MOTION PROCESSING.....	11
5.5 CLOCKING	11
6 ELECTRICAL CHARACTERISTICS	12
6.1 GYROSCOPE SPECIFICATIONS	12
6.2 ACCELEROMETER SPECIFICATIONS	13
6.3 ELECTRICAL AND OTHER COMMON SPECIFICATIONS	14
6.4 ELECTRICAL SPECIFICATIONS, CONTINUED	15
6.5 ELECTRICAL SPECIFICATIONS, CONTINUED	16
6.6 ELECTRICAL SPECIFICATIONS, CONTINUED	17
6.7 I ² C TIMING CHARACTERIZATION	18
6.8 SPI TIMING CHARACTERIZATION (MPU-6000 ONLY)	19
6.9 ABSOLUTE MAXIMUM RATINGS	20
7 APPLICATIONS INFORMATION	21
7.1 PIN OUT AND SIGNAL DESCRIPTION	21
7.2 TYPICAL OPERATING CIRCUIT	22
7.3 BILL OF MATERIALS FOR EXTERNAL COMPONENTS	22
7.4 RECOMMENDED POWER-ON PROCEDURE	23
7.5 BLOCK DIAGRAM	24
7.6 OVERVIEW.....	24
7.7 THREE-AXIS MEMS GYROSCOPE WITH 16-BIT ADCS AND SIGNAL CONDITIONING	25
7.8 THREE-AXIS MEMS ACCELEROMETER WITH 16-BIT ADCS AND SIGNAL CONDITIONING	25
7.9 DIGITAL MOTION PROCESSOR	25
7.10 PRIMARY I ² C AND SPI SERIAL COMMUNICATIONS INTERFACES	25
7.11 AUXILIARY I ² C SERIAL INTERFACE	26

7.12	SELF-TEST.....	27
7.13	MPU-60X0 SOLUTION FOR 9-AXIS SENSOR FUSION USING I ² C INTERFACE.....	28
7.14	MPU-6000 USING SPI INTERFACE.....	29
7.15	INTERNAL CLOCK GENERATION.....	30
7.16	SENSOR DATA REGISTERS.....	30
7.17	FIFO	30
7.18	INTERRUPTS	31
7.19	DIGITAL-OUTPUT TEMPERATURE SENSOR	31
7.20	BIAS AND LDO	31
7.21	CHARGE PUMP.....	31
8	PROGRAMMABLE INTERRUPTS.....	32
8.1	FREE FALL, MOTION, AND ZERO MOTION SIGNAL PATHS	33
8.2	FREE FALL INTERRUPT.....	34
8.3	MOTION INTERRUPT.....	34
8.4	ZERO MOTION INTERRUPT	35
9	DIGITAL INTERFACE	36
9.1	I ² C AND SPI (MPU-6000 ONLY) SERIAL INTERFACES.....	36
9.2	I ² C INTERFACE.....	36
9.3	I ² C COMMUNICATIONS PROTOCOL	36
9.4	I ² C TERMS	39
9.5	SPI INTERFACE (MPU-6000 ONLY)	40
10	SERIAL INTERFACE CONSIDERATIONS (MPU-6050).....	41
10.1	MPU-6050 SUPPORTED INTERFACES	41
10.2	LOGIC LEVELS	41
10.3	LOGIC LEVELS DIAGRAM FOR AUX_VDDIO = 0	42
10.4	LOGIC LEVELS DIAGRAM FOR AUX_VDDIO = 1	43
11	ASSEMBLY	44
11.1	ORIENTATION OF AXES	44
11.2	PACKAGE DIMENSIONS	45
11.3	PCB DESIGN GUIDELINES	46
11.4	ASSEMBLY PRECAUTIONS	47
11.5	STORAGE SPECIFICATIONS.....	51
11.6	PACKAGE MARKING SPECIFICATION.....	51
11.7	TAPE & REEL SPECIFICATION.....	52



11.8	LABEL.....	ERROR! BOOKMARK NOT DEFINED.
11.9	PACKAGING.....	ERROR! BOOKMARK NOT DEFINED.
12	RELIABILITY	53
12.1	QUALIFICATION TEST POLICY.....	55
12.2	QUALIFICATION TEST PLAN.....	55
13	ENVIRONMENTAL COMPLIANCE.....	56

ADVANCE INFORMATION



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00

Revision: 3.2

Release Date: 11/16/2011

1 Revision History

Revision Date	Revision	Description
11/24/2010	1.0	Initial Release
05/19/2011	2.0	For Rev C parts. Clarified wording in sections (3.2, 5.1, 5.2, 6.1-6.4, 6.6, 6.9, 7, 7.1-7.6, 7.11, 7.12, 7.14, 8, 8.2-8.4, 10.3, 10.4, 11, 12.2)
07/28/2011	2.1	Edited supply current numbers for different modes (section 6.4)
08/05/2011	2.2	Unit of measure for accelerometer sensitivity changed from LSB/mg to LSB/g
10/12/2011	2.3	Updated accelerometer self test specifications in Table 6.2. Updated package dimensions (section 11.2). Updated PCB design guidelines (section 11.3)
10/18/2011	3.0	For Rev D parts. Updated accelerometer specifications in Table 6.2. Updated accelerometer specification note (sections 8.2, 8.3, & 8.4). Updated qualification test plan (section 12.2).
10/24/2011	3.1	Edits for clarity Changed operating voltage range to 2.375V-3.46V Added accelerometer Intelligence Function increment value of 1mg/LSB (Section 6.2) Updated absolute maximum rating for acceleration (any axis, unpowered) from 0.3ms to 0.2ms (Section 6.9) Modified absolute maximum rating for Latch-up to Level A and ±100mA (Section 6.9, 12.2)
11/16/2011	3.2	Updated self-test response specifications for Revision D parts dated with date code 1147 (YYWW) or later. Edits for clarity Added Gyro self-test (sections 5.1, 6.1, 7.6, 7.12) Added Min/Max limits to Accel self-test response (section 6.2) Updated Accelerometer low power mode operating currents (Section 6.3) Added gyro self test to block diagram (section 7.5) Updated packaging labels and descriptions (sections 11.8 & 11.9)



2 Purpose and Scope

This product specification provides advanced information regarding the electrical specification and design related information for the MPU-6000™ and MPU-6050™ Motion Processing Unit™, collectively called the MPU-60X0™ or MPU™.

Electrical characteristics are based upon design analysis and simulation results only. Specifications are subject to change without notice. Final specifications will be updated based upon characterization of production silicon. For references to register map and descriptions of individual registers, please refer to the MPU-6000/MPU-6050 Register Map and Register Descriptions document.

The self-test response specifications provided in this document pertain to Revision D parts with date codes of 1147 (YYWW) or later. Please see Section 11.6 for package marking description details.

ADVANCE INFORMATION



3 Product Overview

3.1 MPU-60X0 Overview

The MPU-60X0 Motion Processing Unit is the world's first motion processing solution with integrated 9-Axis sensor fusion using its field-proven and proprietary MotionFusion™ engine for handset and tablet applications, game controllers, motion pointer remote controls, and other consumer devices. The MPU-60X0 has an embedded 3-axis MEMS gyroscope, a 3-axis MEMS accelerometer, and a Digital Motion Processor™ (DMP™) hardware accelerator engine with an auxiliary I²C port that interfaces to 3rd party digitalsensors such as magnetometers. When connected to a 3-axis magnetometer, the MPU-60X0 delivers a complete 9-axis MotionFusion output to its primary I²C or SPI port (SPI is available on MPU-6000 only). The MPU-60X0 combines acceleration and rotational motion plus heading information into a single data stream for the application. This MotionProcessing™ technology integration provides a smaller footprint and has inherent cost advantages compared to discrete gyroscope plus accelerometer solutions. The MPU-60X0 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I²C port. The MPU-60X0 is a 2nd generation motion processor and is footprint compatible with the MPU-30X0 family.

The MPU-60X0 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps) and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g.

An on-chip 1024 Byte FIFO buffer helps lower system power consumption by allowing the system processor to read the sensor data in bursts and then enter a low-power mode as the MPU collects more data. With all the necessary on-chip processing and sensor components required to support many motion-based use cases, the MPU-60X0 uniquely supports a variety of advanced motion-based applications entirely on-chip. The MPU-60X0 thus enables low-power MotionProcessing in portable applications with reduced processing requirements for the system processor. By providing an integrated MotionFusion output, the DMP in the MPU-60X0 offloads the intensive MotionProcessing computation requirements from the system processor, minimizing the need for frequent polling of the motion sensor output.

Communication with all registers of the device is performed using either I²C at 400kHz or SPI at 1MHz (MPU-6000 only). For applications requiring faster communications, the sensor and interrupt registers maybe read using SPI at 20MHz (MPU-6000 only). Additional features include an embedded temperature sensor and an on-chip oscillator with ±1% variation over the operating temperature range.

By leveraging its patented and volume-proven Nasiri-Fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the MPU-60X0 package size down to a revolutionary footprint of 4x4x0.9mm (QFN), while providing the highest performance, lowest noise, and the lowest cost semiconductor packaging required for handheld consumer electronic devices. The part features a robust 10,000g shock tolerance, and has programmable low-pass filters for the gyroscopes, accelerometers, and the on-chip temperature sensor.

For power supply flexibility, the MPU-60X0 operates from VDD power supply voltage range of 2.375V-3.46V. Additionally, the MPU-6050 provides a VLOGIC reference pin (in addition to its analog supply pin: VDD), which sets the logic levels of its I²C interface. The VLOGIC voltage may be 1.8V±5% or VDD.

The MPU-6000 and MPU-6050 are identical, except that the MPU-6050 supports the I²C serial interface only, and has a separate VLOGIC reference pin. The MPU-6000 supports both I²C and SPI interfaces and has a single supply pin, VDD, which is both the device's logic reference supply and the analog supply for the part. The table below outlines these differences:



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.2
Release Date: 11/16/2011

Primary Differences between MPU-6000 and MPU-6050

Part / Item	MPU-6000	MPU-6050
VDD	2.375V-3.46V	V-3.46V
VLOGIC	n/a	1.71V to VDD
Serial Interfaces Supported	I ² C, SPI	I ² C
Pin 8	/CS	VLOGIC
Pin 9	AD0/SDO	AD0
Pin 23	SCL/SCLK	SCL
Pin 24	SDA/SDI	SDA

ADVANCE INFORMATION



4 Applications

- *BlurFree™* technology (for Video/Still Image Stabilization)
- *AirSign™* technology (for Security/Authentication)
- *TouchAnywhere™* technology (for “no touch” UI Application Control/Navigation)
- *MotionCommand™* technology (for Gesture Short-cuts)
- Motion-enabled game and application framework
- InstantGesture™ iG™ gesture recognition
- Location based services, points of interest, and dead reckoning
- Handset and portable gaming
- Motion-based game controllers
- 3D remote controls for Internet connected DTVs and set top boxes, 3D mice
- Wearable sensors for health, fitness and sports
- Toys

ADVANCE INFORMATION

5 Features

5.1 Gyroscope Features

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^{\circ}/sec$
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5 μ A
- Factory calibrated sensitivity scale factor
- User self-test

5.2 Accelerometer Features

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$ and $\pm 16g$
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500 μ A
- Low power accelerometer mode current: 10 μ A at 1.25Hz, 20 μ A at 5Hz, 60 μ A at 20Hz, 110 μ A at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- Free-fall interrupt
- High-G interrupt
- Zero Motion/Motion interrupt
- User self-test

5.3 Additional Features

The MPU-60X0 includes the following additional features:

- 9-Axis MotionFusion by the on-chip Digital Motion Processor (DMP)
- Auxiliary master I²C bus for reading data from external sensors (e.g., magnetometer)
- 3.9mA operating current when all 6 motion sensing axes and the DMP are enabled
- VDD supply voltage range of 2.375V-3.46V
- Flexible VLOGIC reference voltage supports multiple I²C interface voltages (MPU-6050 only)
- Smallest and thinnest QFN package for portable devices: 4x4x0.9mm
- Minimal cross-axis sensitivity between the accelerometer and gyroscope axes
- 1024 byte FIFO buffer reduces power consumption by allowing host processor to read the data in bursts and then go into a low-power mode as the MPU collects more data
- Digital-output temperature sensor
- User-programmable digital filters for gyroscope, accelerometer, and temp sensor
- 10,000 g shock tolerant
- 400kHz Fast Mode I²C for communicating with all registers



- 1MHz SPI serial interface for communicating with all registers (MPU-6000 only)
- 20MHz SPI serial interface for reading sensor and interrupt registers (MPU-6000 only)
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

5.4 MotionProcessing

- Internal Digital Motion Processing™ (DMP™) engine supports 3D MotionProcessing and gesture recognition algorithms
- The MPU-60X0 collects gyroscope and accelerometer data while synchronizing data sampling at a user defined rate. The total dataset obtained by the MPU-60X0 includes 3-Axis gyroscope data, 3-Axis accelerometer data, and temperature data. The MPU's calculated output to the system processor can also include heading data from a digital 3-axis third party magnetometer.
- The FIFO buffers the complete data set, reducing timing requirements on the system processor by allowing the processor burst read the FIFO data. After burst reading the FIFO data, the system processor can save power by entering a low-power sleep mode while the MPU collects more data.
- Programmable interrupt supports features such as gesture recognition, panning, zooming, scrolling, zero-motion detection, tap detection, and shake detection
- Digitally-programmable low-pass filters
- Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.

5.5 Clocking

- On-chip timing generator $\pm 1\%$ frequency variation over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz



6 Electrical Characteristics

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0 FS_SEL=1 FS_SEL=2 FS_SEL=3		± 250 ± 500 ± 1000 ± 2000		%/s %/s %/s %/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0 FS_SEL=1 FS_SEL=2 FS_SEL=3		131 65.5 32.8 16.4		LSB/(°/s) LSB/(°/s) LSB/(°/s) LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			± 2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			± 2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		± 20		%/s	
ZRO Variation Over Temperature	-40°C to +85°C		± 20		%/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		%/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		%/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		%/s	
Linear Acceleration Sensitivity	Static		0.1		%/s/g	
SELF-TEST RESPONSE						
X-Axis		10		105	%/s	
Y-Axis ¹		-105		-10	%/s	
Z-Axis		10		105	%/s	
GYROSCOPE NOISE PERFORMANCE						
Total RMS Noise	FS_SEL=0 DLPFCFG=2 (100Hz)		0.05		%/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to 10Hz		0.033		%/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		%/s/√Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE	Programmable Range	5		256	Hz	
OUTPUT DATA RATE	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME	DLPFCFG=0 to $\pm 1\%$ /s of Final		30		ms	

1. The Y-Axis self-test has an opposite polarities from the X-, and Z-Axes. However, the magnitude of the self-test limit is the same for all 3 axes.



6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		± 2 ± 4 ± 8 ± 16		g g g g	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		16,384 8,192 4,096 2,048		LSB/g LSB/g LSB/g LSB/g	
Initial Calibration Tolerance			± 3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		± 0.02		%/ ^o C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			± 2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance ¹	X and Y axes Z axis		± 50 ± 80		mg mg	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C Z axis, 0°C to +70°C		± 35 ± 60		mg	
SELF TEST RESPONSE		300		950	mg	
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		μ g/ \sqrt Hz	
LOW PASS FILTER RESPONSE	Programmable Range	5		260	Hz	
OUTPUT DATA RATE	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			1		mg/LSB	

1. Typical zero-g initial calibration tolerance value after MSL3 preconditioning



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.2
Release Date: 11/16/2011

6.3 Electrical and Other Common Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	Notes
TEMPERATURE SENSOR						
Range			-40 to +85		°C	
Sensitivity						
Temperature Offset	35'					
Linearity	Best fit straight line (-40°C to +85°C)					
VDD POWER SUPPLY		2.375		3.46	V	
Operating Voltages						
Normal Operating Current	Gyroscope + Accelerometer + DMP		3.9		mA	
	Gyroscope + Accelerometer (DMP disabled)		3.8		mA	
	Gyroscope + DMP (Accelerometer disabled)		3.7		mA	
	Gyroscope only (DMP & Accelerometer disabled)		3.6		mA	
	Accelerometer only (DMP & Gyroscope disabled)	500			μA	
Accelerometer Low Power Mode Current	1 Hz update rate		10		μA	
	5 Hz update rate		20		μA	
	20 Hz update rate		70		μA	
	40 Hz update rate		140		μA	
Full-Chip Idle Mode Supply Current			5		μA	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			100	ms	
VLOGIC REFERENCE VOLTAGE						
Voltage Range	MPU-6050 only			VDD	V	
Power Supply Ramp Rate	VLOGIC must be \leq VDD at all times	1.71		3	ms	
Normal Operating Current	Monotonic ramp. Ramp rate is 10% to 90% of the final value		100		μA	
START-UP TIME FOR REGISTER READ/WRITE			20	100	ms	
TEMPERATURE RANGE						
Specified Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range	-40		+85	°C	



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.2
Release Date: 11/16/2011

6.4 Electrical Specifications, Continued

V_{DD} = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or V_{DD}, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
SERIAL INTERFACE						
SPI Operating Frequency, All Registers Read/Write	MPU-6000 only, Low Speed Characterization MPU-6000 only, High Speed Characterization MPU-6000 only		100 ±10% 1 ±10% 20 ±10%		kHz MHz MHz	
SPI Operating Frequency, Sensor and Interrupt Registers Read Only	All registers, Fast-mode			400	kHz	
I ² C Operating Frequency	All registers, Standard-mode			100	kHz	
I²C ADDRESS	AD0 = 0 AD0 = 1		1101000 1101001			
DIGITAL INPUTS (SDI/SDA, AD0, SCLK/SCL, FSYNC, /CS, CLKIN)						
V _{IH} , High Level Input Voltage	MPU-6000 MPU-6050		0.7*VDD 0.7*VLOGIC		V V	
V _{IL} , Low Level Input Voltage	MPU-6000 MPU-6050			0.3*VDD 0.3*VLOGIC	V V	
C _i , Input Capacitance			< 5		pF	
DIGITAL OUTPUT (SDO, INT)						
V _{OH} , High Level Output Voltage	R _{LOAD} =1MΩ; MPU-6000		0.9*VDD		V	
V _{OL1} , LOW-Level Output Voltage	R _{LOAD} =1MΩ; MPU-6050		0.9*VLOGIC		V	
V _{OL,INT1} , INT Low-Level Output Voltage	R _{LOAD} =1MΩ; MPU-6000			0.1*VDD	V	
Output Leakage Current	R _{LOAD} =1MΩ; MPU-6050			0.1*VLOGIC	V	
t _{INT} , INT Pulse Width	OPEN=1, 0.3mA sink Current OPEN=1 LATCH_INT_EN=0			0.1	V	
				100	nA	
				50	μs	
DIGITAL OUTPUT (CLKOUT)						
V _{OH} , High Level Output Voltage	R _{LOAD} =1MΩ		0.9*VDD		V	
V _{OL1} , LOW-Level Output Voltage	R _{LOAD} =1MΩ			0.1*VDD	V	



6.5 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD, TA = 25°C

Parameters	Conditions	Typical	Units	Notes
Primary I²C I/O (SCL, SDA)				
V _{IL} , LOW-Level Input Voltage	MPU-6000	-0.5 to 0.3*VDD	V	
V _{IH} , HIGH-Level Input Voltage	MPU-6000	0.7*VDD to VDD + 0.5V	V	
V _{hys} , Hysteresis	MPU-6000	0.1*VDD	V	
V _{IL} , LOW Level Input Voltage	MPU-6050	-0.5V to 0.3*VLOGIC	V	
V _{IH} , HIGH-Level Input Voltage	MPU-6050	0.7*VLOGIC to VLOGIC + 0.5V	V	
V _{hys} , Hysteresis	MPU-6050	0.1*VLOGIC	V	
V _{OL1} , LOW-Level Output Voltage	3mA sink current	0 to 0.4	V	
I _{OL} , LOW-Level Output Current	V _{OL} = 0.4V	3	mA	
	V _{OL} = 0.6V	5	mA	
Output Leakage Current		100	nA	
t _{of} , Output Fall Time from V _{IHmax} to V _{ILmax}	C _b bus capacitance in pF	20+0.1C _b to 250	ns	
C _i , Capacitance for Each I/O pin		< 10	pF	
Auxiliary I²C I/O (AUX_CL, AUX_DA)	MPU-6050: AUX_VDDIO=0			
V _{IL} , LOW-Level Input Voltage		-0.5 to 0.8*VLOGIC	V	
V _{IH} , HIGH-Level Input Voltage		0.7*VLOGIC to VLOGIC + 0.5V	V	
V _{hys} , Hysteresis		0.1*VLOGIC	V	
V _{OL1} , LOW-Level Output Voltage	VLOGIC > 2V; 1mA sink current	0 to 0.4	V	
V _{OL3} , LOW-Level Output Voltage	VLOGIC < 2V; 1mA sink current	0 to 0.2*VLOGIC	V	
I _{OL} , LOW-Level Output Current	V _{OL} = 0.4V	1	mA	
	V _{OL} = 0.6V	1	mA	
Output Leakage Current		100	nA	
t _{of} , Output Fall Time from V _{IHmax} to V _{ILmax}	C _b bus capacitance in pF	20+0.1C _b to 250	ns	
C _i , Capacitance for Each I/O pin		< 10	pF	
Auxiliary I²C I/O (AUX_CL, AUX_DA)	MPU-6050: AUX_VDDIO=1; MPU-6000			
V _{IL} , LOW-Level Input Voltage		-0.5 to 0.3*VDD	V	
V _{IH} , HIGH-Level Input Voltage		0.7*VDD to VDD+0.5V	V	
V _{hys} , Hysteresis		0.1*VDD	V	
V _{OL1} , LOW-Level Output Voltage	1mA sink current	0 to 0.4	V	
I _{OL} , LOW-Level Output Current	V _{OL} = 0.4V	1	mA	
	V _{OL} = 0.6V	1	mA	
Output Leakage Current		100	nA	
t _{of} , Output Fall Time from V _{IHmax} to V _{ILmax}	C _b bus cap. in pF	20+0.1C _b to 250	ns	
C _i , Capacitance for Each I/O pin		< 10	pF	



6.6 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD, TA = 25°C

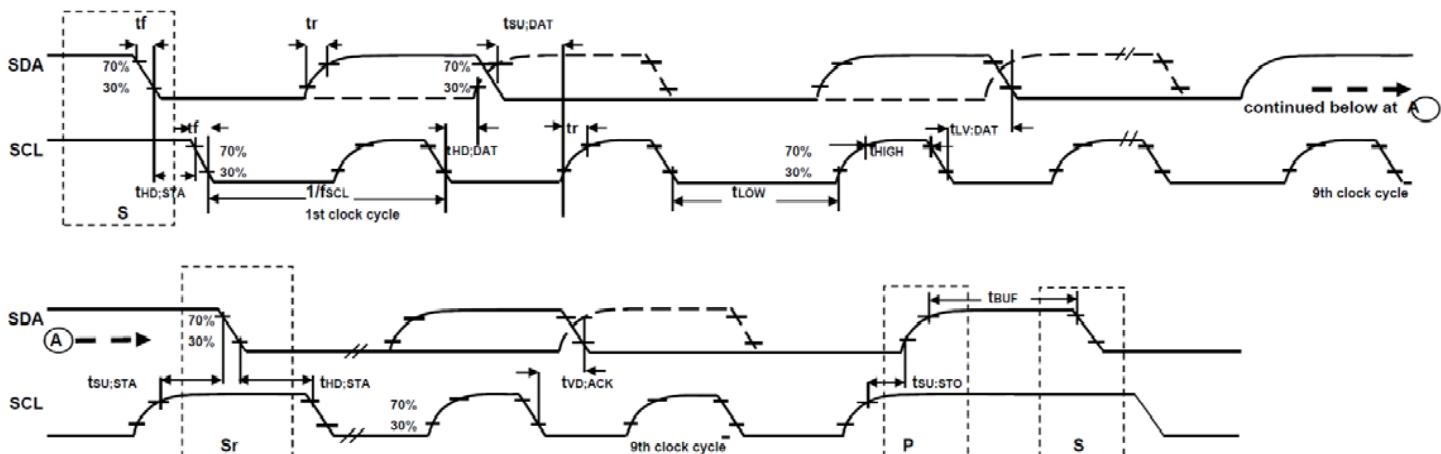
Parameters	Conditions	Min	Typical	Max	Units	Notes
INTERNAL CLOCK SOURCE	CLK_SEL=0,1,2,3					
Gyroscope Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Accelerometer Sample Rate			1		kHz	
Reference Clock Output	CLKOUTEN = 1		1.024		MHz	
Clock Frequency Initial Tolerance	CLK_SEL=0, 25°C	-5		+5	%	
Frequency Variation over Temperature	CLK_SEL=1,2,3; 25°C	-1		+1	%	
PLL Settling Time	CLK_SEL=0		-15 to +10 \pm 1		%	
	CLK_SEL=1,2,3		1	10	ms	
EXTERNAL 32.768kHz CLOCK	CLK_SEL=4					
External Clock Frequency			32.768		kHz	
External Clock Allowable Jitter	Cycle-to-cycle rms		1 to 2		μ s	
Gyroscope Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8.192		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1.024		kHz	
Accelerometer Sample Rate			1.024		kHz	
Reference Clock Output	CLKOUTEN = 1		1.0486		MHz	
PLL Settling Time			1	10	ms	
EXTERNAL 19.2MHz CLOCK	CLK_SEL=5					
External Clock Frequency	Full programmable range		19.2		MHz	
Gyroscope Sample Rate	DLPFCFG=0 SAMPLERATEDIV = 0		8		Hz	
Gyroscope Sample Rate, Fast Mode	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Gyroscope Sample Rate, Slow Mode			1		kHz	
Accelerometer Sample Rate			1		kHz	
Reference Clock Output	CLKOUTEN = 1		1.024		MHz	
PLL Settling Time			1	10	ms	

6.7 I²C Timing Characterization

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	I²C FAST-MODE					
f _{SCL} , SCL Clock Frequency		0.6		400	kHz	
t _{HOLD,STA} , (Repeated) START Condition Hold Time		1.3			μs	
t _{LOW} , SCL Low Period		0.6			μs	
t _{HIGH} , SCL High Period		0.6			μs	
t _{SU,STA} , Repeated START Condition Setup Time		0.6			μs	
t _{HOLD,DAT} , SDA Data Hold Time		0			μs	
t _{SU,DAT} , SDA Data Setup Time		100			ns	
t _r , SDA and SCL Rise Time	C _b bus cap. from 10 to 400pF	20+0.1C _b			ns	
t _f , SDA and SCL Fall Time	C _b bus cap. from 10 to 400pF	20+0.1C _b			ns	
t _{SU,STO} , STOP Condition Setup Time		0.6			μs	
t _{BUF} , Bus Free Time Between STOP and START Condition		1.3			μs	
C _b , Capacitive Load for each Bus Line		< 400			pF	
t _{VD,DAT} , Data Valid Time		0.9			μs	
t _{VD,ACK} , Data Valid Acknowledge Time		0.9			μs	

Note: Timing Characteristics apply to both Primary and Auxiliary I²C Bus

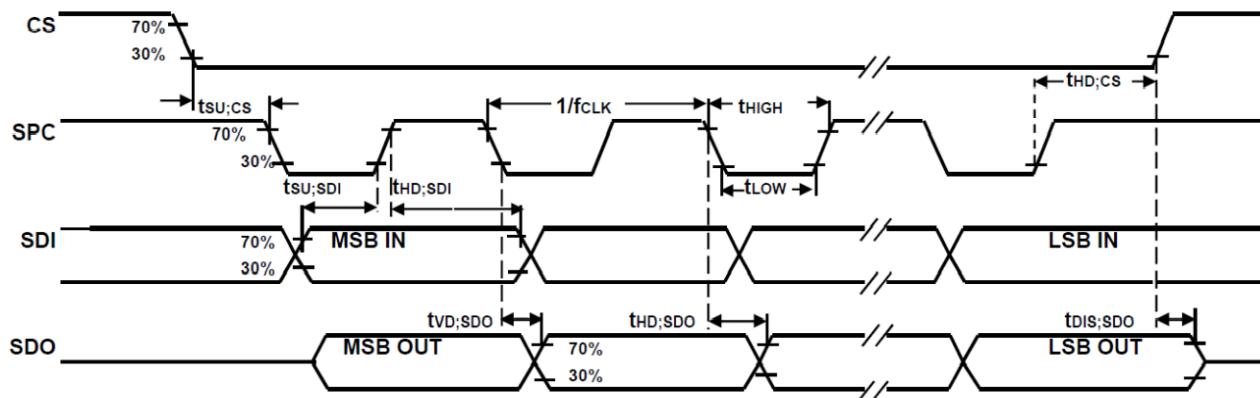


I²C Bus Timing Diagram

6.8 SPI Timing Characterization (MPU-6000 only)

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V \pm 5% or VDD,T_A = -40°C to +85°C, unless otherwise noted.

Parameters	Conditions	Min	Typical	Max	Units	Notes
SPI TIMING						
f _{SCLK} , SCLK Clock Frequency				1	MHz	
t _{LOW} , SCLK Low Period		400			ns	
t _{HIGH} , SCLK High Period		400			ns	
t _{SU;CS} , CS Setup Time		8			ns	
t _{HD;CS} , CS Hold Time		500			ns	
t _{SU;SDI} , SDI Setup Time		11			ns	
t _{HD;SDI} , SDI Hold Time		7			ns	
t _{VD;SDO} , SDO Valid Time	C _{load} = 20pF		100		ns	
t _{HD;SDO} , SDO Hold Time	C _{load} = 20pF	4			ns	
t _{DIS;SDO} , SDO Output Disable Time				10	ns	



SPI Bus Timing Diagram



6.9 Absolute Maximum Ratings

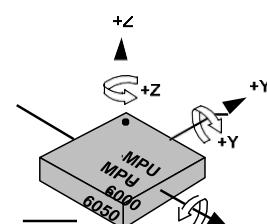
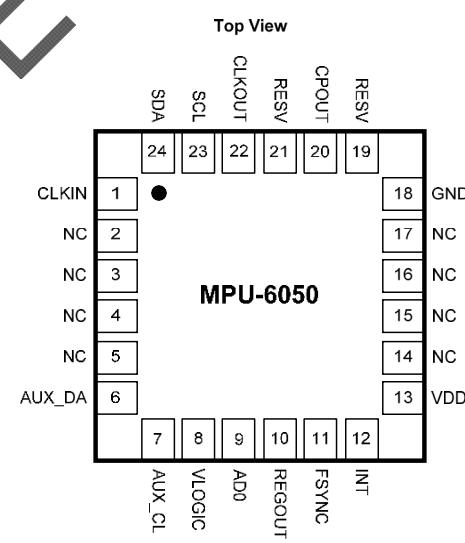
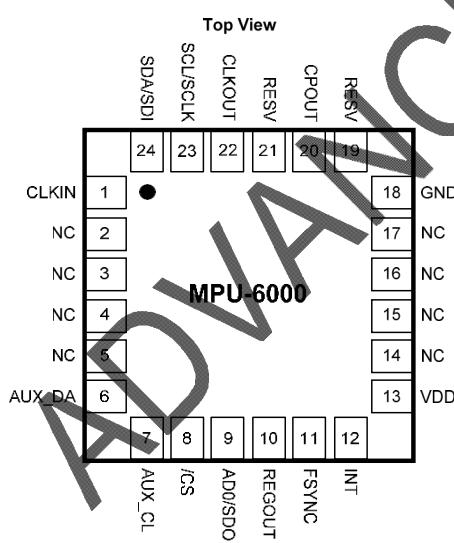
Stress above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

Parameter	Rating
Supply Voltage, VDD	-0.5V to +6V
VLOGIC Input Voltage Level (MPU-6050)	-0.5V to VDD + 0.5V
REGOUT	-0.5V to 2V
Input Voltage Level (CLKIN, AUX_DA, AD0, FSYNC, INT, SCL, SDA)	-0.5V to VDD + 0.5V
CPOUT (2.5V ≤ VDD ≤ 3.6V)	-0.5V to 30V
Acceleration (Any Axis, unpowered)	10,000g for 0.2ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	2kV (HBM); 200V (MM)
Latch-up	JEDEC Class II (2), 125°C Level A, ±100mA

7 Applications Information

7.1 Pin Out and Signal Description

Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I ² C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I ² C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I ² C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I ² C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	CLKOUT	System clock output
23	Y		SCL / SCLK	I ² C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I ² C serial clock (SCL)
24	Y		SDA / SDI	I ² C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I ² C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.

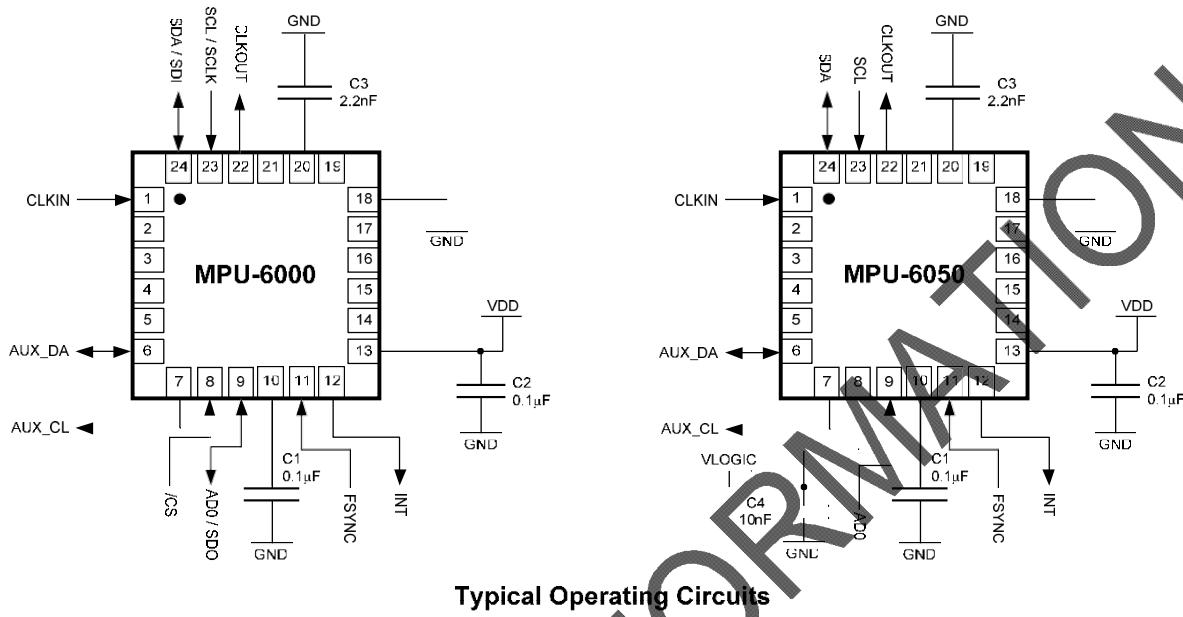


QFN Package
24-pin, 4mm x 4mm x 0.9mm

QFN Package
24-pin, 4mm x 4mm x 0.9mm

Orientation of Axes of Sensitivity and
Polarity of Rotation

7.2 Typical Operating Circuit

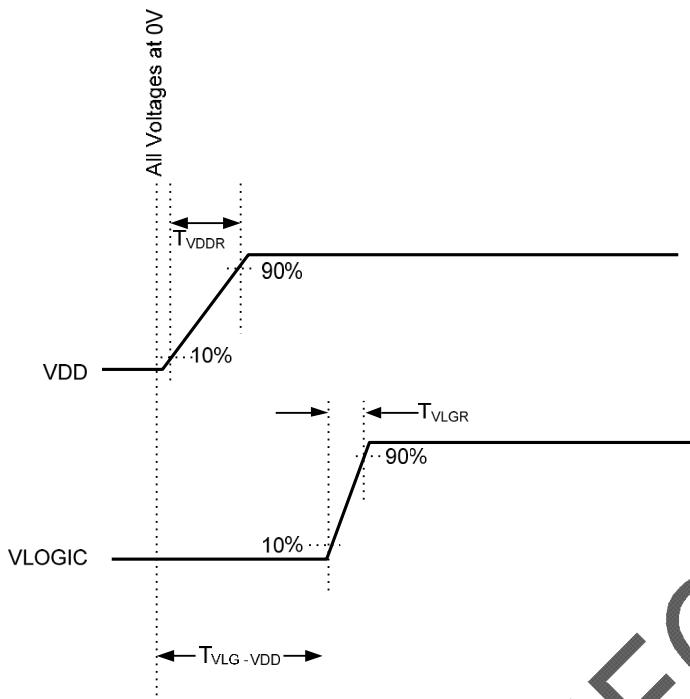


7.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Regulator Filter Capacitor (Pin 10)	C1	Ceramic, X7R, 0.1µF ±10%, 2V	1
VDD Bypass Capacitor (Pin 13)	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Charge Pump Capacitor (Pin 20)	C3	Ceramic, X7R, 2.2nF ±10%, 50V	1
VLOGIC Bypass Capacitor (Pin 8)	C4*	Ceramic, X7R, 10nF ±10%, 4V	1

* MPU-6050 Only.

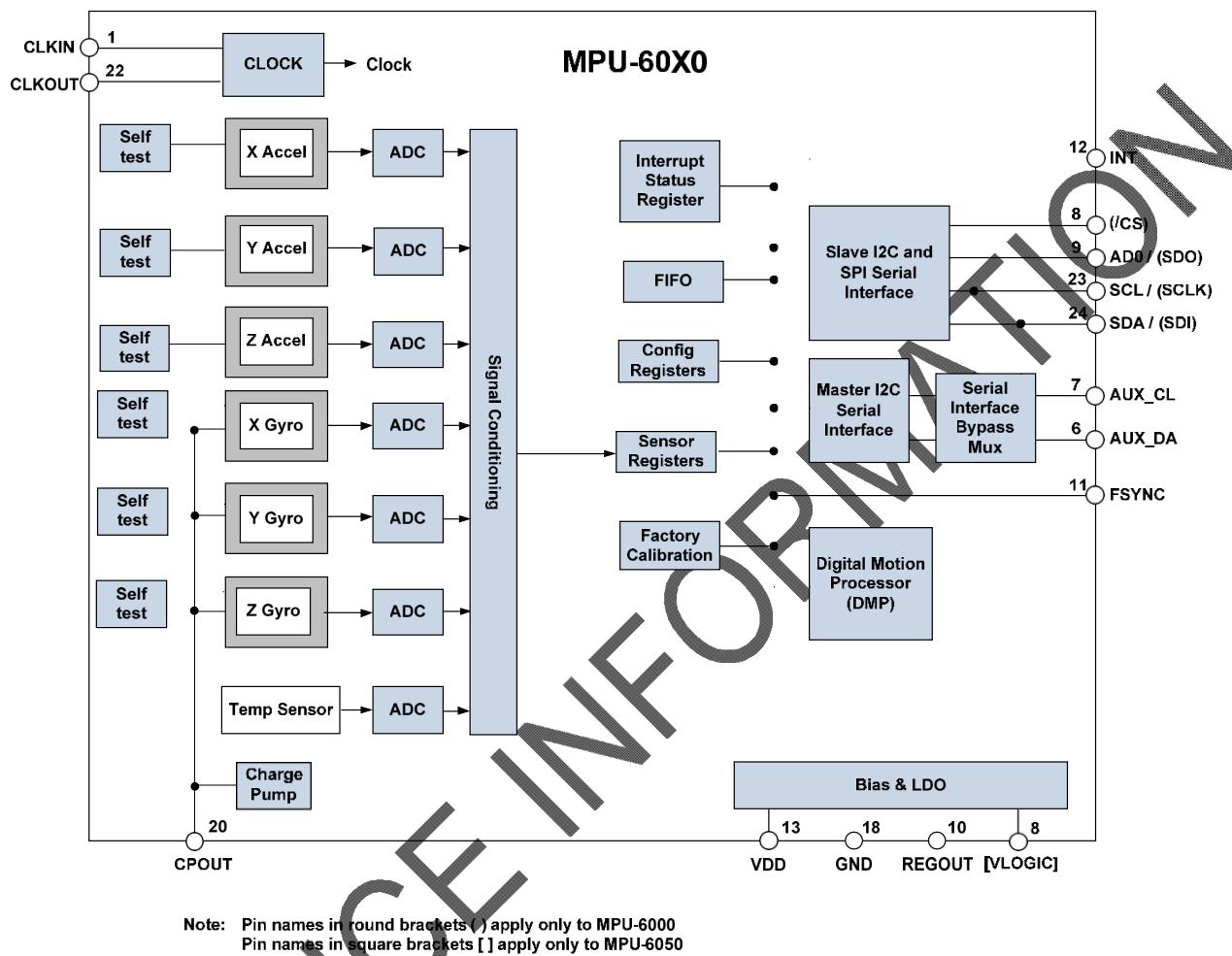
7.4 Recommended Power-on Procedure



Power-Up Sequencing

1. VLOGIC amplitude must always be \leq VDD amplitude
2. T_{VDDR} is VDD rise time: Time for VDD to rise from 10% to 90% of its final value
3. $T_{VDDR} \leq 100\text{ms}$
4. T_{VLGR} is VLOGIC rise time: Time for VLOGIC to rise from 10% to 90% of its final value
5. $T_{VLGR} \leq 3\text{ms}$
6. $T_{VLG-VDD}$ is the delay from the start of VDD ramp to the start of VLOGIC rise
7. $T_{VLG-VDD} \geq 0$
8. VDD and VLOGIC must be monotonic ramps

7.5 Block Diagram



7.6 Overview

The MPU-60X0 is comprised of the following key blocks and functions:

- Three-axis MEMS rate gyroscope sensor with 16-bit ADCs and signal conditioning
- Three-axis MEMS accelerometer sensor with 16-bit ADCs and signal conditioning
- Digital Motion Processor (DMP) engine
- Primary I²C and SPI (MPU-6000 only) serial communications interfaces
- Auxiliary I²C serial interface for 3rd party magnetometer & other sensors
- Clocking
- Sensor Data Registers
- FIFO
- Interrupts
- Digital-Output Temperature Sensor
- Gyroscope & Accelerometer Self-test
- Bias and LDO
- Charge Pump

7.7 Three-Axis MEMS Gyroscope with 16-bit ADCs and Signal Conditioning

The MPU-60X0 consists of three independent vibratory MEMS rate gyroscopes, which detect rotation about the X-, Y-, and Z- Axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis. The full-scale range of the gyro sensors may be digitally programmed to ± 250 , ± 500 , ± 1000 , or ± 2000 degrees per second (dps). The ADC sample rate is programmable from 8,000 samples per second, down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

7.8 Three-Axis MEMS Accelerometer with 16-bit ADCs and Signal Conditioning

The MPU-60X0's 3-Axis accelerometer uses separate proof masses for each axis. Acceleration along a particular axis induces displacement on the corresponding proof mass, and capacitive sensors detect the displacement differentially. The MPU-60X0's architecture reduces the accelerometers' susceptibility to fabrication variations as well as to thermal drift. When the device is placed on a flat surface, it will measure 0g on the X- and Y-axes and +1g on the Z-axis. The accelerometers' scale factor is calibrated at the factory and is nominally independent of supply voltage. Each sensor has a dedicated sigma-delta ADC for providing digital outputs. The full scale range of the digital output can be adjusted to $\pm 2g$, $\pm 4g$, $\pm 8g$, or $\pm 16g$.

7.9 Digital Motion Processor

The embedded Digital Motion Processor (DMP) is located within the MPU-60X0 and offloads computation of motion processing algorithms from the host processor. The DMP acquires data from accelerometers, gyroscopes, and additional 3rd party sensors such as magnetometers, and processes the data. The resulting data can be read from the DMP's registers, or can be buffered in a FIFO. The DMP has access to one of the MPU's external pins, which can be used for generating interrupts.

The purpose of the DMP is to offload both timing requirements and processing power from the hostprocessor. Typically, motion processing algorithms should be run at a high rate, often around 200Hz, in orderto provide accurate results with low latency. This is required even if the application updates at a much lower rate; for example, a low power user interface may update as slowly as 5Hz, but the motion processing shouldstill run at 200Hz. The DMP can be used as a tool in order to minimize power, simplify timing, simplify the software architecture, and save valuable MIPS on the host processor for use in the application.

7.10 Primary I²C and SPI Serial Communications Interfaces

The MPU-60X0 communicates to a system processor using either a SPI (MPU-6000 only) or an I²C serial interface. The MPU-60X0 always acts as a slave when communicating to the system processor. The LSB of the I²C slave address is set by pin 9 (AD0).

The logic levels for communications between the MPU-60X0 and its master are as follows:

- MPU-6000: The logic level for communications with the master is set by the voltage on VDD
- MPU-6050: The logic level for communications with the master is set by the voltage on VLOGIC

For further information regarding the logic levels of the MPU-6050, please refer to Section 10.

7.11 Auxiliary I²C Serial Interface

The MPU-60X0 has an auxiliary I²C bus for communicating to an off-chip 3-Axis digital output magnetometer or other sensors. This bus has two operating modes:

- I²C Master Mode: The MPU-60X0 acts as a master to any external sensors connected to the auxiliary I²C bus
- Pass-Through Mode: The MPU-60X0 directly connects the primary and auxiliary I²C buses together, allowing the system processor to directly communicate with any external sensors.

Auxiliary I²C Bus Modes of Operation:

- I²C Master Mode: Allows the MPU-60X0 to directly access the data registers of external digital sensors, such as a magnetometer. In this mode, the MPU-60X0 directly obtains data from auxiliary sensors, allowing the on-chip DMP to generate sensor fusion data without intervention from the system applications processor.

For example, In I²C Master mode, the MPU-60X0 can be configured to perform burst reads, returning the following data from a magnetometer:

- X magnetometer data (2 bytes)
- Y magnetometer data (2 bytes)
- Z magnetometer data (2 bytes)

The I²C Master can be configured to read up to 24 bytes from up to 4 auxiliary sensors. A fifth sensor can be configured to work single byte read/write mode.

- Pass-Through Mode: Allows an external system processor to act as master and directly communicate to the external sensors connected to the auxiliary I²C bus pins (AUX_DA and AUX_CL). In this mode, the auxiliary I²C bus control logic (3rd party sensor interface block) of the MPU-60X0 is disabled, and the auxiliary I²C pins AUX_DA and AUX_CL (Pins 6 and 7) are connected to the main I²C bus (Pins 23 and 24) through analog switches.

Pass-Through Mode is useful for configuring the external sensors, or for keeping the MPU-60X0 in a low-power mode when only the external sensors are used.

In Pass-Through Mode the system processor can still access MPU-60X0 data through the I²C interface.

Auxiliary I²C Bus IO Logic Levels

- MPU-6000: The logic level of the auxiliary I²C bus is VDD
- MPU-6050: The logic level of the auxiliary I²C bus can be programmed to be either VDD or VLOGIC

For further information regarding the MPU-6050's logic levels, please refer to Section 10.2.



7.12 Self-Test

Self-test allows for the testing of the mechanical and electrical portions of the sensors. The self-test for each measurement axis can be activated by controlling the bits of the Gyro and Accel control registers.

When self-test is activated, the electronics cause the sensors to be actuated and produce an output signal. The output signal is used to observe the self-test response.

The self-test response is defined as follows:

Self-test response = Sensor output with self-test enabled – Sensor output without self-test enabled

The self-test response for each accelerometer axis is defined in the accelerometer specification table (Section 6.2). Similarly, for the gyro axes, self test allows the proof masses to be moved equivalent to a pre-defined Coriolis force, resulting in a change in sensor output. The self-test response for each gyroscope axis is defined in the gyroscope specification table (Section 6.1).

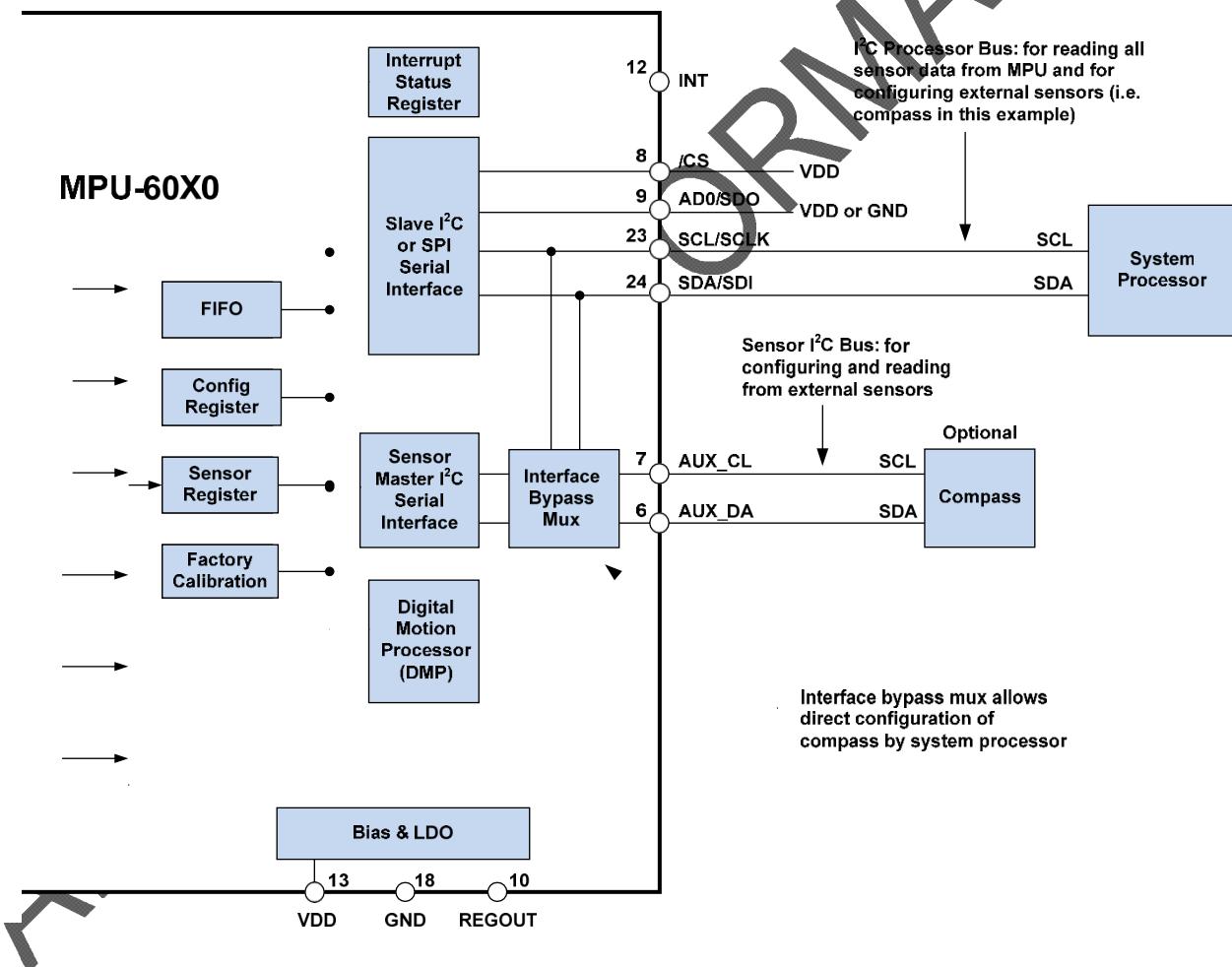
For further information regarding the Gyro & Accel control registers, please refer to the MPU-6050 Register Map and Register Descriptions document.

7.13 MPU-60X0 Solution for 9-axis Sensor Fusion Using I²C Interface

In the figure below, the system processor is an I²C master to the MPU-60X0. In addition, the MPU-60X0 is an I²C master to the optional external compass sensor. The MPU-60X0 has limited capabilities as an I²C Master, and depends on the system processor to manage the initial configuration of any auxiliary sensors. The MPU-60X0 has an interface bypass multiplexer, which connects the system processor I²C bus pins 23 and 24 (SDA and SCL) directly to the auxiliary sensor I²C bus pins 6 and 7 (AUX_DA and AUX_CL).

Once the auxiliary sensors have been configured by the system processor, the interface bypass multiplexer should be disabled so that the MPU-60X0 auxiliary I²C master can take control of the sensor I²C bus and gather data from the auxiliary sensors.

For further information regarding I²C master control, please refer to Section 10.



7.14 MPU-6000 Using SPI Interface

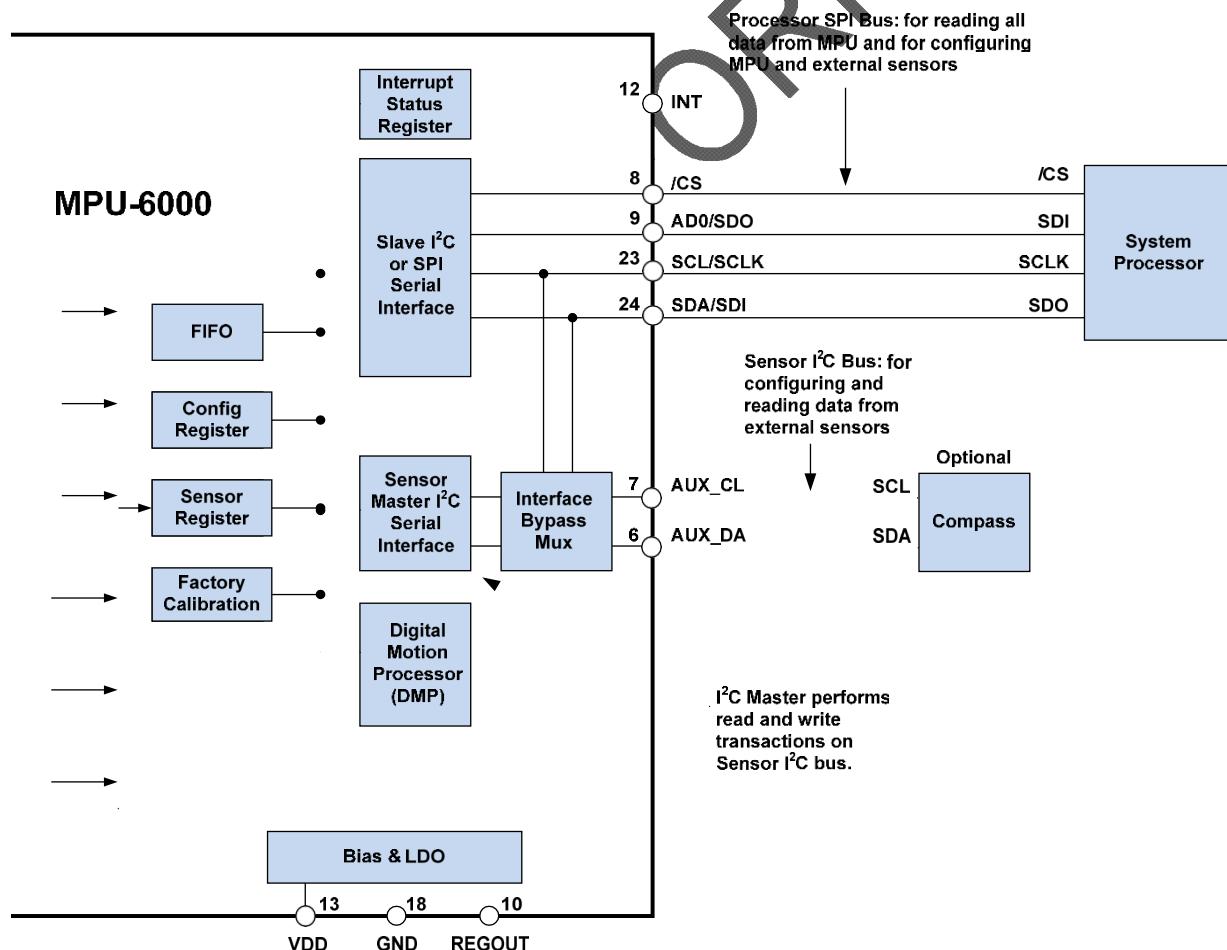
In the figure below, the system processor is an SPI master to the MPU-6000. Pins 8, 9, 23, and 24 are used to support the /CS, SDO, SCLK, and SDI signals for SPI communications. Because these SPI pins are shared with the I²C slave pins (9, 23 and 24), the system processor cannot access the auxiliary I²C bus through the interface bypass multiplexer, which connects the processor I²C interface pins to the sensor I²C interface pins.

Since the MPU-6000 has limited capabilities as an I²C Master, and depends on the system processor to manage the initial configuration of any auxiliary sensors, another method must be used for programming the sensors on the auxiliary sensor I²C bus pins 6 and 7 (AUX_DA and AUX_CL).

When using SPI communications between the MPU-6000 and the system processor, configuration of devices on the auxiliary I²C sensor bus can be achieved by using I²C Slaves 0-4 to perform read and write transactions on any device and register on the auxiliary I²C bus. The I²C Slave 4 interface can be used to perform only single byte read and write transactions.

Once the external sensors have been configured, the MPU-6000 can perform single or multi-byte reads using the sensor I²C bus. The read results from the Slave 0-3 controllers can be written to the FIFO buffer as well as to the external sensor registers.

For further information regarding the control of the MPU-60X0's auxiliary I²C interface, please refer to the MPU-60X0 Register Map and Register Descriptions document.



7.15 Internal Clock Generation

The MPU-60X0 has a flexible clocking scheme, allowing a variety of internal or external clock sources to be used for the internal synchronous circuitry. This synchronous circuitry includes the signal conditioning and ADCs, the DMP, and various control circuits and registers. An on-chip PLL provides flexibility in the allowable inputs for generating this clock.

Allowable internal sources for generating the internal clock are:

- An internal relaxation oscillator
- Any of the X, Y, or Z gyros (MEMS oscillators with a variation of $\pm 1\%$ over temperature)

Allowable external clocking sources are:

- 32.768kHz square wave
- 19.2MHz square wave

Selection of the source for generating the internal synchronous clock depends on the availability of external sources and the requirements for power consumption and clock accuracy. These requirements will most likely vary by mode of operation. For example, in one mode, where the biggest concern is power consumption, the user may wish to operate the Digital Motion Processor of the MPU-60X0 to process accelerometer data, while keeping the gyros off. In this case, the internal relaxation oscillator is a good clock choice. However, in another mode, where the gyros are active, selecting the gyros as the clock source provides for a more accurate clock source.

Clock accuracy is important, since timing errors directly affect the distance and angle calculations performed by the Digital Motion Processor (and by extension, by any processor).

There are also start-up conditions to consider. When the MPU-60X0 first starts up, the device uses its internal clock until programmed to operate from another source. This allows the user, for example, to wait for the MEMS oscillators to stabilize before they are selected as the clock source.

7.16 Sensor Data Registers

The sensor data registers contain the latest gyro, accelerometer, auxiliary sensor, and temperature measurement data. They are read-only registers, and are accessed via the serial interface. Data from these registers may be read anytime. However, the interrupt function may be used to determine when new data is available.

For a table of interrupt sources please refer to Section 8.

7.17 FIFO

The MPU-60X0 contains a 1024-byte FIFO register that is accessible via the Serial Interface. The FIFO configuration register determines which data is written into the FIFO. Possible choices include gyro data, accelerometer data, temperature readings, auxiliary sensor readings, and FSYNC input. A FIFO counter keeps track of how many bytes of valid data are contained in the FIFO. The FIFO register supports burst reads. The interrupt function may be used to determine when new data is available.

For further information regarding the FIFO, please refer to the MPU-60X0 Register Map and Register Descriptions document.



7.18 Interrupts

Interrupt functionality is configured via the Interrupt Configuration register. Items that are configurable include the INT pin configuration, the interrupt latching and clearing method, and triggers for the interrupt. Items that can trigger an interrupt are (1) Clock generator locked to new reference oscillator (used when switching clock sources); (2) new data is available to be read (from the FIFO and Data registers); (3) accelerometer event interrupts; and (4) the MPU-60X0 did not receive an acknowledge from an auxiliary sensor on the secondary I²C bus. The interrupt status can be read from the Interrupt Status register.

For further information regarding interrupts, please refer to the MPU-60X0 Register Map and Register Descriptions document.

For information regarding the MPU-60X0's accelerometer event interrupts, please refer to Section 8.

7.19 Digital-Output Temperature Sensor

An on-chip temperature sensor and ADC are used to measure the MPU-60X0 die temperature. The readings from the ADC can be read from the FIFO or the Sensor Data registers.

7.20 Bias and LDO

The bias and LDO section generates the internal supply and the reference voltages and currents required by the MPU-60X0. Its two inputs are an unregulated VDD of 2.375 to 3.46V and a VLOGIC logic reference supply voltage of 1.71V to VDD (MPU-6050 only). The LDO output is bypassed by a capacitor at REGOUT. For further details on the capacitor, please refer to the Bill of Materials for External Components (Section 7.3).

7.21 Charge Pump

An on-board charge pump generates the high voltage required for the MEMS oscillators. Its output is bypassed by a capacitor at CPOUT. For further details on the capacitor, please refer to the Bill of Materials for External Components (Section 7.3).



8 Programmable Interrupts

The MPU-60X0 has a programmable interrupt system which can generate an interrupt signal on the INT pin. Status flags indicate the source of an interrupt. Interrupt sources may be enabled and disabled individually.

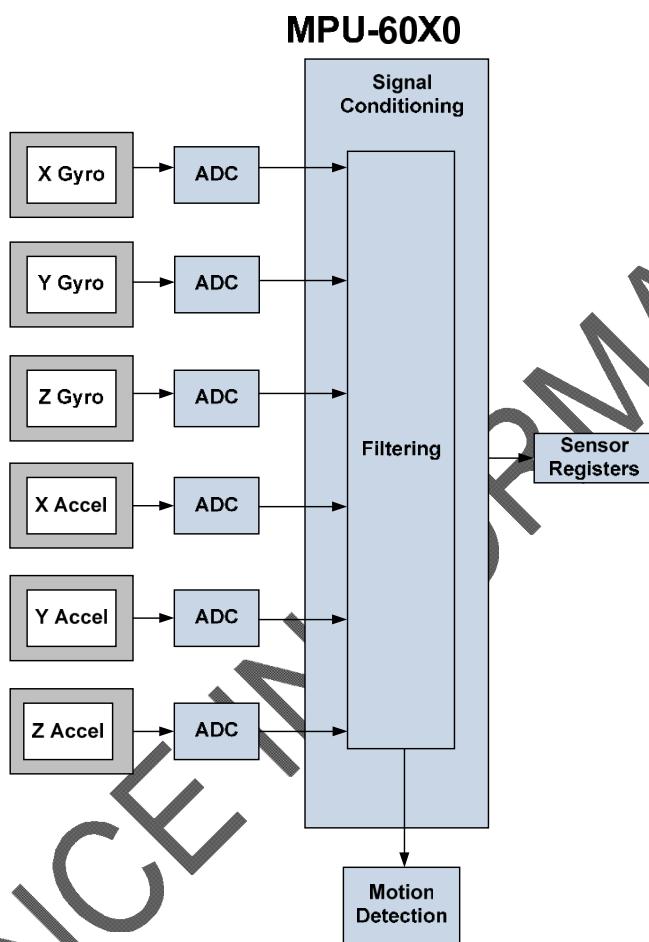
Table of Interrupt Sources

Interrupt Name	Module
Free Fall Detection	Free Fall
Motion Detection	Motion
Zero Motion Detection	Zero Motion
FIFO Overflow	FIFO
Data Ready	Sensor Registers
I ² C Master errors: Lost Arbitration, NACKs	I ² C Master
I ² C Slave 4	I ² C Master

For information regarding the interrupt enable/disable registers and flag registers, please refer to the MPU-6000/MPU-6050 Register Map and Register Descriptions document. Some interrupt sources are explained below.

8.1 Free Fall, Motion, and Zero Motion Signal Paths

The diagram below shows the signal path for the gyroscope and accelerometer sensors. Note that each digital low pass filter (DLPF) is configured identically, as is each sample rate divider and digital high pass filter (DHPF).

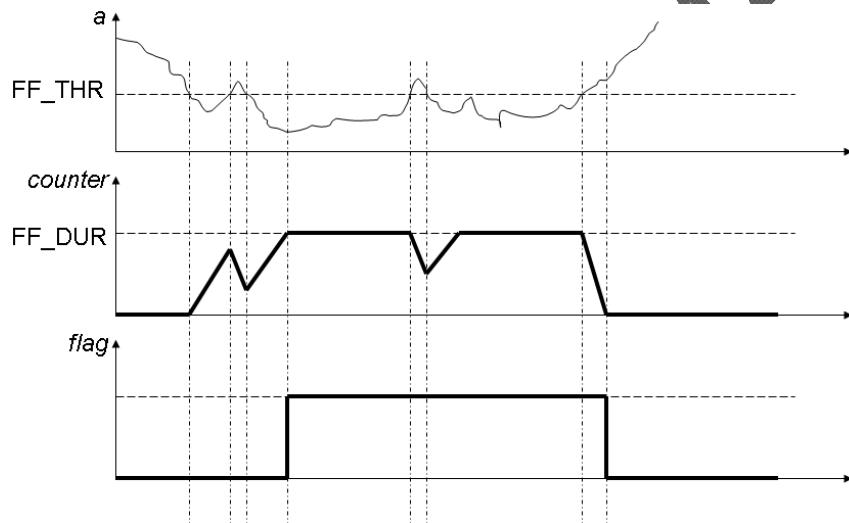


8.2 Free Fall Interrupt

Free fall is detected by checking if the accelerometer measurements from all 3 axes have an absolute value below a user-programmable threshold (acceleration threshold). For each sample where this condition is true (a qualifying sample), a counter is incremented. For each sample where this condition is false (a non- qualifying sample), the counter is decremented. Once the counter reaches a user-programmable threshold (the counter threshold), the Free Fall interrupt is triggered and a flag is set. The flag is cleared once the counter has decremented to zero. The counter does not increment above the counter threshold or decrement below zero.

The user is given several configuration parameters to fine tune Free Fall detection. Both, the acceleration threshold and counter threshold are user configurable. The FF_THR register allows the user to set a threshold in 1 mg increments. The FF_DUR register allows the user to set duration in 1 ms increments.

The decrement rate for non-qualifying samples is also configurable. The MOT_DETECT_CTRL register allows the user to specify whether a non-qualifying sample makes the counter reset to zero, or decrement in steps of 1, 2, or 4.



The figure above shows a simplified example with just one axis. An example acceleration input signal (simplified to only show one axis), qualifying sample counter, and Free Fall flag are shown.

8.3 Motion Interrupt

The MPU-60X0 provides Motion detection capability with similar functionality to Free Fall detection. Accelerometer measurements are passed through a configurable digital high pass filter (DHPF) in order to eliminate bias due to gravity. A qualifying motion sample is one where the high passed sample from any axis has an absolute value exceeding a user-programmable threshold. A counter increments for each qualifying sample, and decrements for each non-qualifying sample. Once the counter reaches a user-programmable counter threshold, a motion interrupt is triggered. The axis and polarity which caused the interrupt to be triggered is flagged in the MOT_DETECT_STATUS register.

Like Free Fall detection, Motion detection has a configurable acceleration threshold MOT_THR specified in 1 mg increments. The counter threshold MOT_DUR is specified in 1 ms increments. The decrement rate has the same options as Free Fall detection, and is specified in the MOT_DETECT_CTRL register.



8.4 Zero Motion Interrupt

The Zero Motion detection capability uses the digital high pass filter (DHPF) and a similar threshold scheme to that of Free Fall detection. Each axis of the high passed accelerometer measurement must have an absolute value less than a threshold specified in the ZRMOT_THR register, which can be increased in 1 mg increments. Each time a motion sample meets this condition, a counter increments. When this counter reaches a threshold specified in ZRMOT_DUR, an interrupt is generated.

Unlike Free Fall or Motion detection, Zero Motion detection triggers an interrupt both when Zero Motion is first detected and when Zero Motion is no longer detected. While Free Fall and Motion are indicated with a flag which clears after being read, reading the state of the Zero Motion detected from the MOT_DETECT_STATUS register does not clear its status.

ADVANCE INFORMATION

9 Digital Interface

9.1 I²C and SPI (MPU-6000 only) Serial Interfaces

The internal registers and memory of the MPU-6000/MPU-6050 can be accessed using either I²C at 400 kHz or SPI at 1MHz (MPU-6000 only). SPI operates in four-wire mode.

Serial Interface

Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
8	Y		/CS	SPI chip select (0=SPI enable)
8		Y	VLOGIC	Digital I/O supply voltage. VLOGIC must be \leq VDD at all times.
9	Y		AD0 / SDO	I ² C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I ² C Slave Address LSB
23	Y		SCL / SCLK	I ² C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I ² C serial clock
24	Y		SDA / SDI	I ² C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I ² C serial data

Note:

To prevent switching into I²C mode when using SPI (MPU-6000), the I²C interface should be disabled by setting the *I2C_IF_DIS* configuration bit. Setting this bit should be performed immediately after waiting for the time specified by the “Start-Up Time for Register Read/Write” in Section 6.3.

For further information regarding the *I2C_IF_DIS* bit, please refer to the MPU-60X0 Register Map and Register Descriptions document.

9.2 I²C Interface

I²C is a two-wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I²C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master.

The MPU-60X0 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400 kHz.

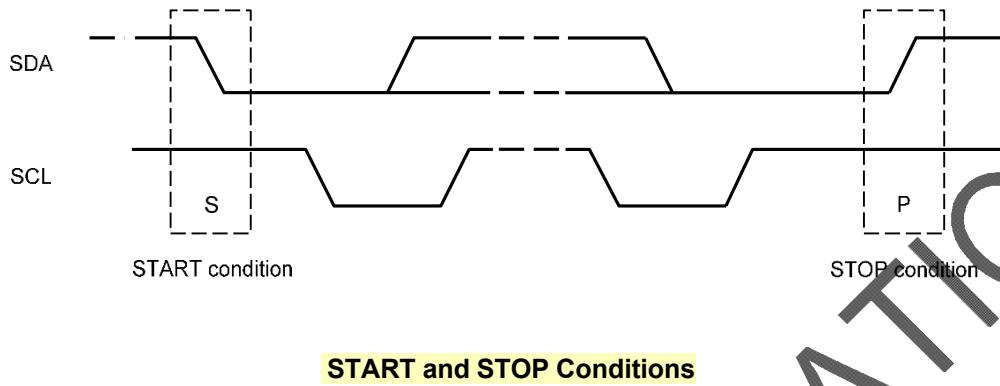
The slave address of the MPU-60X0 is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin AD0. This allows two MPU-60X0s to be connected to the same I²C bus. When used in this configuration, the address of the one of the devices should be [pin AD0 is logic low] and the address of the other should be b1101001 (pin AD0 is logic high).

9.3 I²C Communications Protocol

START (S) and STOP (P) Conditions

Communication on the I²C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below).

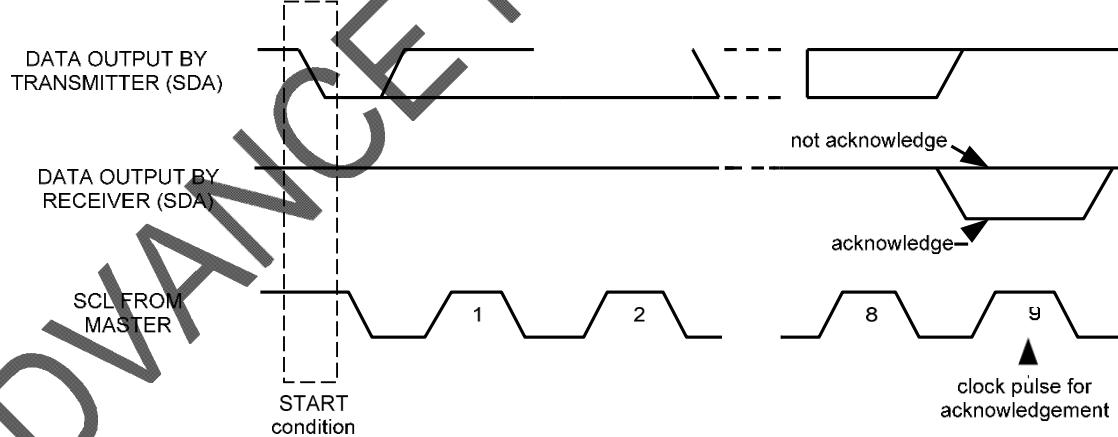
Additionally, the bus remains busy if a repeated START (Sr) is generated instead of a STOP condition.



Data Format / Acknowledge

I²C data bytes are defined to be 8 bits. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse.

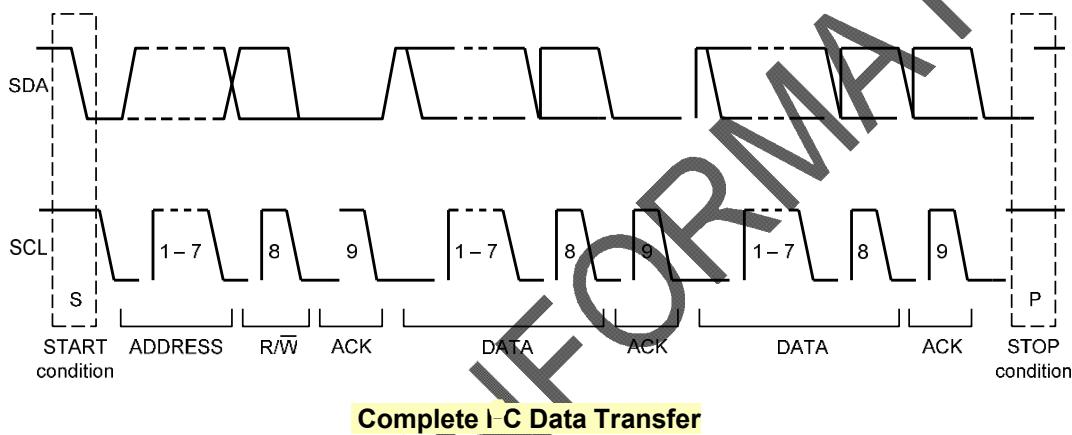
If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (refer to the following figure).



Acknowledge on the I²C Bus

Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8-bit data byte, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.



To write the internal MPU-60X0 registers, the master transmits the start condition (S), followed by the I²C address and the write bit (0). At the 9th clock cycle (when the clock is high), the MPU-60X0 acknowledges the transfer. After the MPU-60X0 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the MPU-60X0 automatically loads the data to the appropriate register. The following figures show single and two-byte write sequences.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

To read the internal MPU-60X0 registers, the master sends a start condition, followed by the I²C address and a write bit, and then the register address of the register it is going to be read. Upon receiving the ACK signal from the MPU-60X0, the master transmits a start signal followed by the slave address and read bit. As a result, the MPU-60X0 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R		ACK		NACK	P
Slave			ACK		ACK			ACK	DATA	DATA		

9.4 I²C Signals

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I ² C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9 th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9 th clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

9.5 SPI Interface (MPU-6000 only)

SPI is a 4-wire synchronous serial interface that uses two control lines and two data lines. The MPU-6000 always operates as a Slave device during standard Master-Slave SPI operation.

With respect to the Master, the Serial Clock output (SCLK), the Serial Data Output (SDO) and the Serial Data Input (SDI) are shared among the Slave devices. Each SPI slave device requires its own Chip Select (/CS) line from the master.

/CS goes low (active) at the start of transmission and goes back high (inactive) at the end. Only one /CS line is active at a time, ensuring that only one slave is selected at any given time. The /CS lines of the non-selected slave devices are held high, causing their SDO lines to remain in a high-impedance (high-z) state so that they do not interfere with any active devices.

SPI Operational Features

1. Data is delivered MSB first and LSB last
2. Data is latched on the rising edge of SCLK
3. Data should be transitioned on the falling edge of SCLK
4. The maximum frequency of SCLK is 1MHz
5. SPI read and write operations are completed in 16 or more clock cycles (two or more bytes). The first byte contains the SPI Address, and the following byte(s) contain(s) the SPI data. The first bit of the first byte contains the Read/Write bit and indicates the Read (1) or Write (0) operation. The following 7 bits contain the Register Address. In cases of multiple-byte Read/Writes, data is two or more bytes:

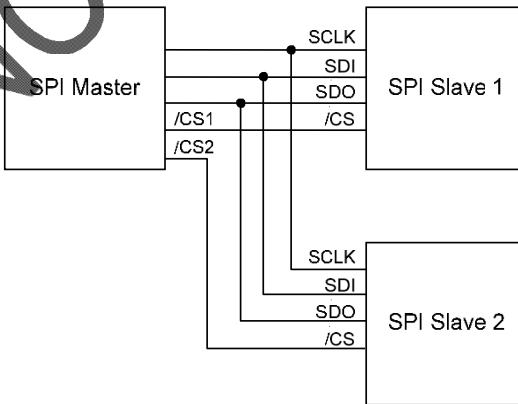
SPI Address format

MSB							LSB
R/W	A6	A5	A4	A3	A2	A1	A0

SPI Data format

MSB							LSB
D7	D6	D5	D4	D3	D2	D1	D0

6. Supports Single or Burst Read/Writes.



Typical SPI Master / Slave Configuration

10 Serial Interface Considerations (MPU-6050)

10.1 MPU-6050 Supported Interfaces

The MPU-6050 supports I²C communications on both its primary (microprocessor) serial interface and its auxiliary interface.

10.2 Logic Levels

The MPU-6050's I/O logic levels are set to be either VDD or VLOGIC, as shown in the table below.

I/O Logic Levels vs. *AUX_VDDIO*

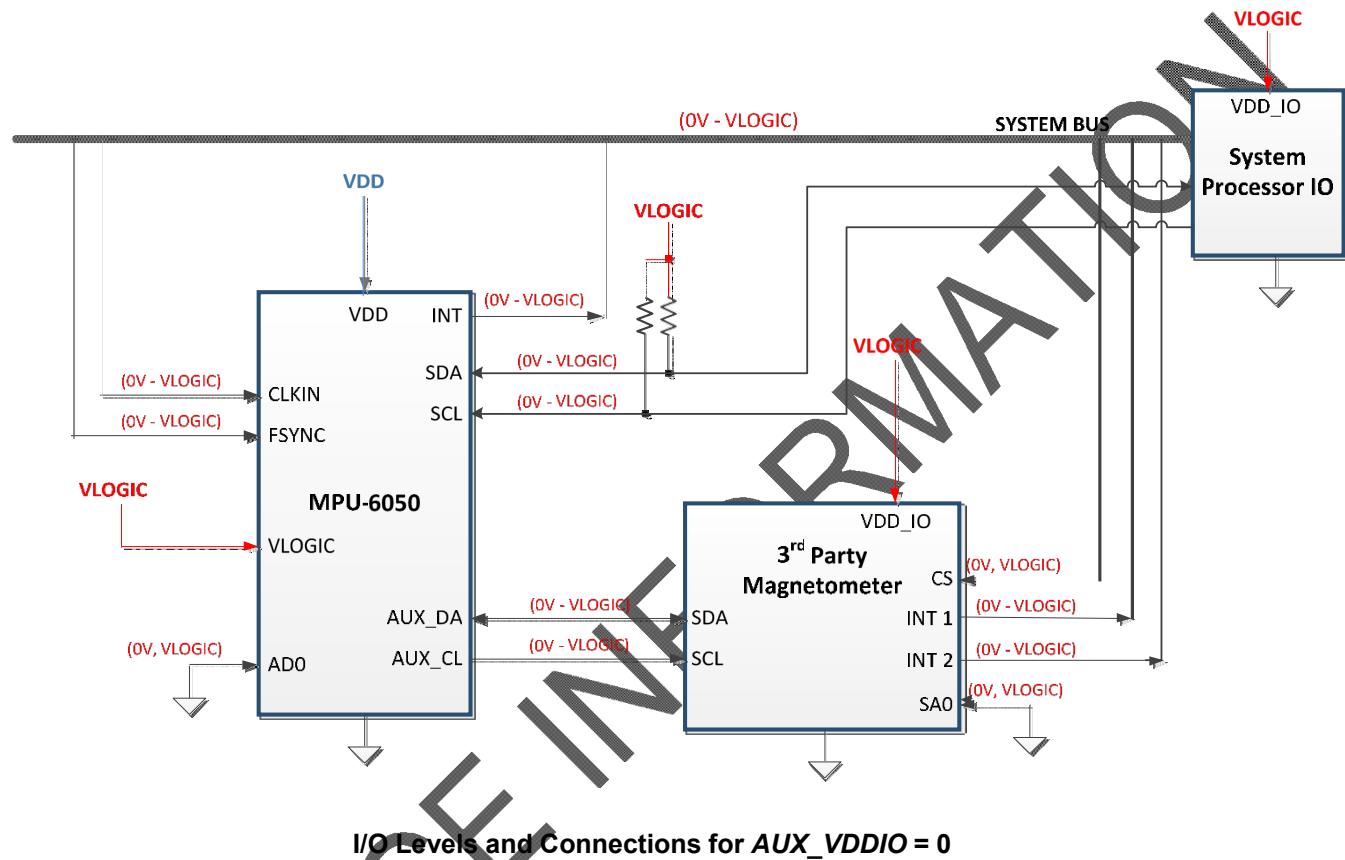
<i>AUX_VDDIO</i>	MICROPROCESSOR LOGIC LEVELS (Pins: SDA, SCL, AD0, CLKIN, INT)	AUXILIARY LOGIC LEVELS (Pins: AUX_DA, AUX_CL)
0	VLOGIC	VLOGIC
1	VLOGIC	VDD

Note: The power-on-reset value for *AUX_VDDIO* is 0.

VLOGIC may be set to be equal to VDD or to another voltage. However, VLOGIC must be \leq VDD at all times. When *AUX_VDDIO* is set to 0 (its power-on-reset value), VLOGIC is the power supply voltage for both the microprocessor system bus and the auxiliary I²C bus, as shown in the figure of Section 10.3. When *AUX_VDDIO* is set to 1, VLOGIC is the power supply voltage for the microprocessor system bus and VDD is the supply for the auxiliary I²C bus, as shown in the figure of Section 10.4.

10.3 Logic Levels Diagram for AUX_VDDIO = 0

The figure below depicts a sample circuit with a third party magnetometer attached to the auxiliary I²C bus. It shows logic levels and voltage connections for AUX_VDDIO = 0. Note: Actual configuration will depend on the auxiliary sensors used.

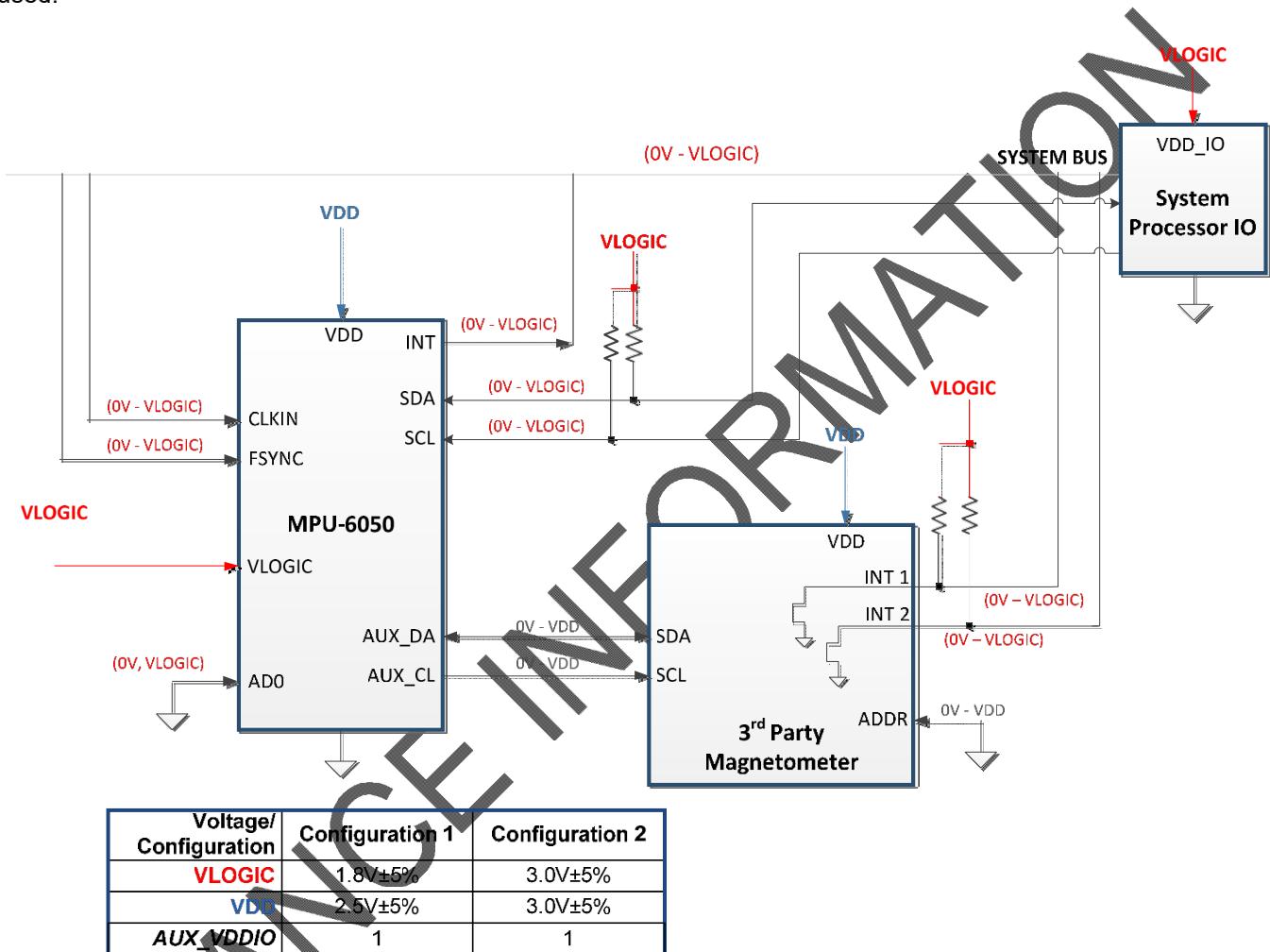


Notes:

1. AUX_VDDIO determines the IO voltage levels of AUX_DA and AUX_CL (0 = set output levels relative to VLOGIC)
2. CLKOUT is referenced to VDD.
3. All other MPU-6050 logic IOs are referenced to VLOGIC.

10.4 Logic Levels Diagram for AUX_VDDIO = 1

The figure below depicts a sample circuit with a 3rd party magnetometer attached to the auxiliary I²C bus. It shows logic levels and voltage connections for AUX_VDDIO = 1. This configuration is useful when the auxiliary sensor has only one supply for logic and power. Note: Actual configuration will depend on the auxiliary sensors used.



I/O Levels and Connections for Two Example Power Configurations (AUX_VDDIO = 1)

Notes:

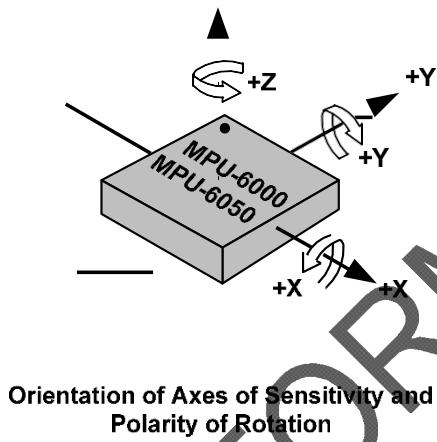
1. AUX_VDDIO determines the IO voltage levels of AUX_DA and AUX_CL. AUX_VDDIO = 1 sets output levels relative to VDD.
2. 3rd-party auxiliary device logic levels are referenced to VDD. Setting INT1 and INT2 to open drain configuration provides voltage compatibility when VDD ≠ VLOGIC. When VDD = VLOGIC, INT1 and INT2 may be set to push-pull outputs, and external pull-up resistors are not needed.
3. CLKOUT is referenced to VDD.
4. All other MPU-6050 logic IOs are referenced to VLOGIC.

11 Assembly

This section provides general guidelines for assembling InvenSense Micro Electro-Mechanical Systems (MEMS) gyros packaged in Quad Flat No leads package (QFN) surface mount integrated circuits.

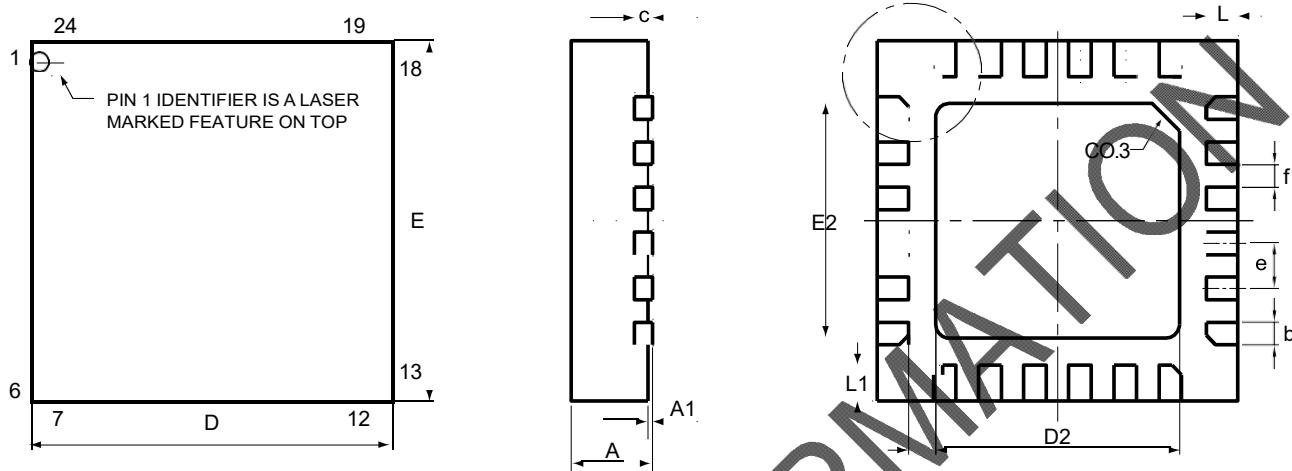
11.1 Orientation of Axes

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation. Note the pin 1 identifier (•) in the figure.

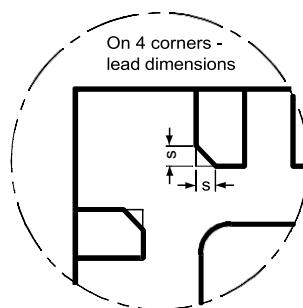


11.2 Package Dimensions

24 Lead QFN (4x4x0.9) mm NiPdAu Lead-frame finish

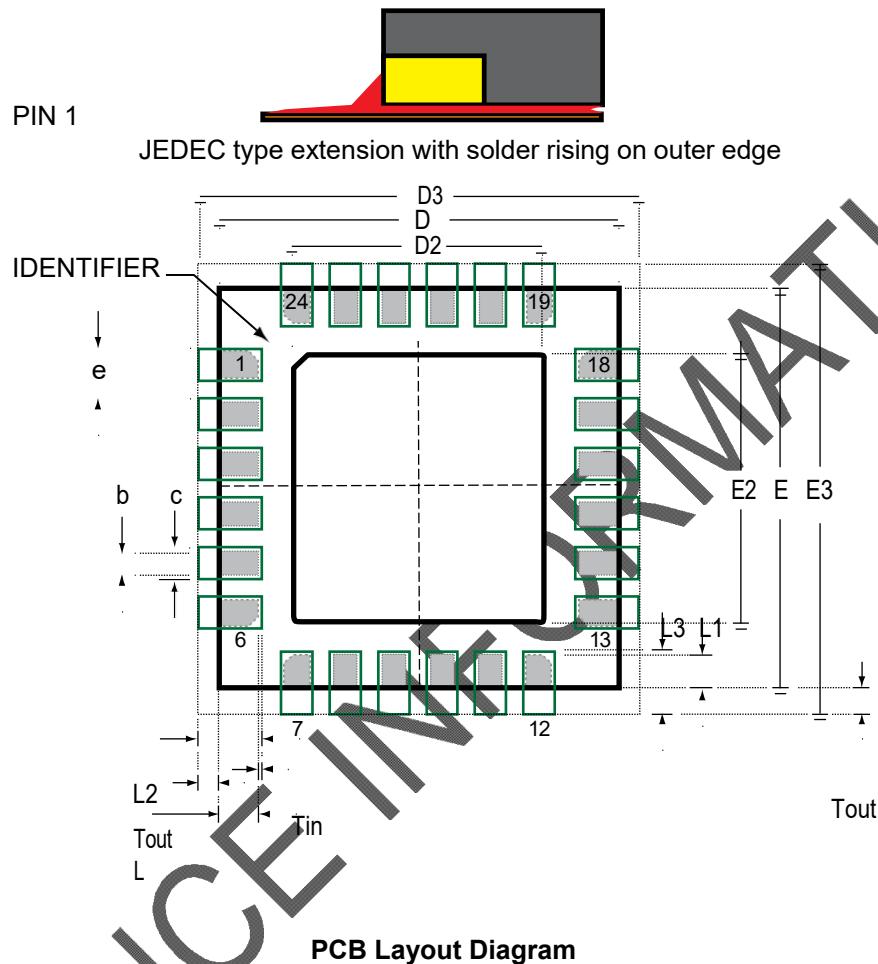


SYMBOLS	DIMENSIONS IN MILLIMETERS		
	MIN	NOM	MAX
A	0.85	0.90	0.95
A1	0.00	0.02	0.05
b	0.18	0.25	0.30
c	---	0.20 REF	---
D	3.90	4.00	4.10
D2	2.65	2.70	2.75
E	3.90	4.00	4.10
E2	2.55	2.60	2.65
e	---	0.50	---
f (e-b)	---	0.25	---
K	0.25	0.30	0.35
L	0.30	0.35	0.40
L1	0.35	0.40	0.45
s	0.05	---	0.15



11.3 PCB Design Guidelines

The Pad Diagram using a JEDEC type extension with solder rising on the outer edge is shown below. The Pad Dimensions Table shows pad sizing (mean dimensions) recommended for the MPU-60X0 product.



SYMBOLS	DIMENSIONS IN MILLIMETERS		NOM
	Nominal Package I/O Pad Dimensions		
e	Pad Pitch	0.50	
b	Pad Width	0.25	
L	Pad Length	0.35	
L1	Pad Length	0.40	
D	Package Width	4.00	
E	Package Length	4.00	
D2	Exposed Pad Width	2.70	
E2	Exposed Pad Length	2.60	
I/O Land Design Dimensions (Guidelines)			
D3	I/O Pad Extent Width	4.80	
E3	I/O Pad Extent Length	4.80	
c	Land Width	0.35	
Tout	Outward Extension	0.40	
Tin	Inward Extension	0.05	
L2	Land Length	0.80	
L3	Land Length	0.85	

PCB Dimensions Table (for PCB Lay-out Diagram)

11.4 Assembly Precautions

11.4.1 Gyroscope Surface Mount Guidelines

InvenSense MEMS Gyros sense rate of rotation. In addition, gyroscopes sense mechanical stress coming from the printed circuit board (PCB). This PCB stress can be minimized by adhering to certain design rules:

When using MEMS gyroscope components in plastic packages, PCB mounting and assembly can cause package stress. This package stress in turn can affect the output offset and its value over a wide range of temperatures. This stress is caused by the mismatch between the Coefficient of Linear Thermal Expansion (CTE) of the package material and the PCB. Care must be taken to avoid package stress due to mounting.

Traces connected to pads should be as symmetric as possible. Maximizing symmetry and balance for pad connection will help component self alignment and will lead to better control of solder paste reduction after reflow.

Any material used in the surface mount assembly process of the MEMS gyroscope should be free of restricted RoHS elements or compounds. Pb-free solders should be used for assembly.

11.4.2 Exposed Die Pad Precautions

The MPU-60X0 has very low active and standby current consumption. The exposed die pad is not required for heat sinking, and should not be soldered to the PCB. Failure to adhere to this rule can induce performance changes due to package thermo-mechanical stress. There is no electrical connection between the pad and the CMOS.

11.4.3 Trace Routing

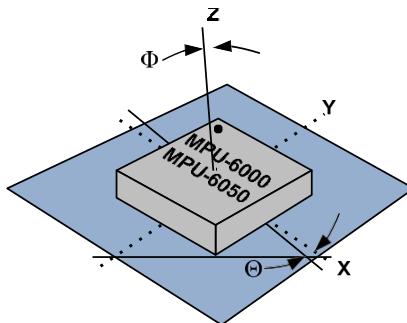
Routing traces or vias under the gyro package such that they run under the exposed die pad is prohibited. Routed active signals may harmonically couple with the gyro MEMS devices, compromising gyro response. These devices are designed with the drive frequencies as follows: X = $33\pm3\text{Khz}$, Y = $30\pm3\text{Khz}$, and Z = $27\pm3\text{Khz}$. To avoid harmonic coupling don't route active signals in non-shielded signal planes directly below, or above the gyro package. Note: For best performance, design a ground plane under the e-pad to reduce PCB signal noise from the board on which the gyro device is mounted. If the gyro device is stacked under an adjacent PCB board, design a ground plane directly above the gyro device to shield active signals from the adjacent PCB board.

11.4.4 Component Placement

Do not place large insertion components such as keyboard or similar buttons, connectors, or shielding boxes at a distance of less than 6 mm from the MEMS gyro. Maintain generally accepted industry design practices for component placement near the MPU-60X0 to prevent noise coupling and thermo-mechanical stress.

11.4.5 PCB Mounting and Cross-Axis Sensitivity

Orientation errors of the gyroscope and accelerometer mounted to the printed circuit board can cause cross-axis sensitivity in which one gyro or accel responds to rotation or acceleration about another axis, respectively. For example, the X-axis gyroscope may respond to rotation about the Y or Z axes. The orientation mounting errors are illustrated in the figure below.



Package Gyro & Accel Axes (-----) Relative to PCB Axes (——) with Orientation Errors (Θ and Φ)

The table below shows the cross-axis sensitivity as a percentage of the gyroscope or accelerometer's sensitivity for a given orientation error, respectively.

Cross-Axis Sensitivity vs. Orientation Error

Orientation Error (Θ or Φ)	Cross-Axis Sensitivity ($\sin\theta$ or $\sin\Phi$)
0°	0%
0.5°	0.87%
1°	1.75%

The specifications for cross-axis sensitivity in Section 6.1 and Section 6.2 include the effect of the die orientation error with respect to the package.

11.4.6 MEMS Handling Instructions

MEMS (Micro Electro-Mechanical Systems) are a time-proven, robust technology used in hundreds of millions of consumer, automotive and industrial products. MEMS devices consist of microscopic moving mechanical structures. They differ from conventional IC products, even though they can be found in similar packages. Therefore, MEMS devices require different handling precautions than conventional ICs prior to mounting onto printed circuit boards (PCBs).

The MPU-60X0 has been qualified to a shock tolerance of 10,000g. InvenSense packages its gyroscopes as it deems proper for protection against normal handling and shipping. It recommends the following handling precautions to prevent potential damage.

- Do not drop individually packaged gyroscopes, or trays of gyroscopes onto hard surfaces. Components placed in trays could be subject to g-forces in excess of 10,000g if dropped.
- Printed circuit boards that incorporate mounted gyroscopes should not be separated by manually snapping apart. This could also create g-forces in excess of 10,000g.

11.4.7 ESD Considerations

Establish and use ESD-safe handling precautions when unpacking and handling ESD-sensitive devices.

- Store ESD sensitive devices in ESD safe containers until ready for use. The Tape-and-Reel moisture-sealed bag is an ESD approved barrier. The best practice is to keep the units in the original moisture sealed bags until ready for assembly.

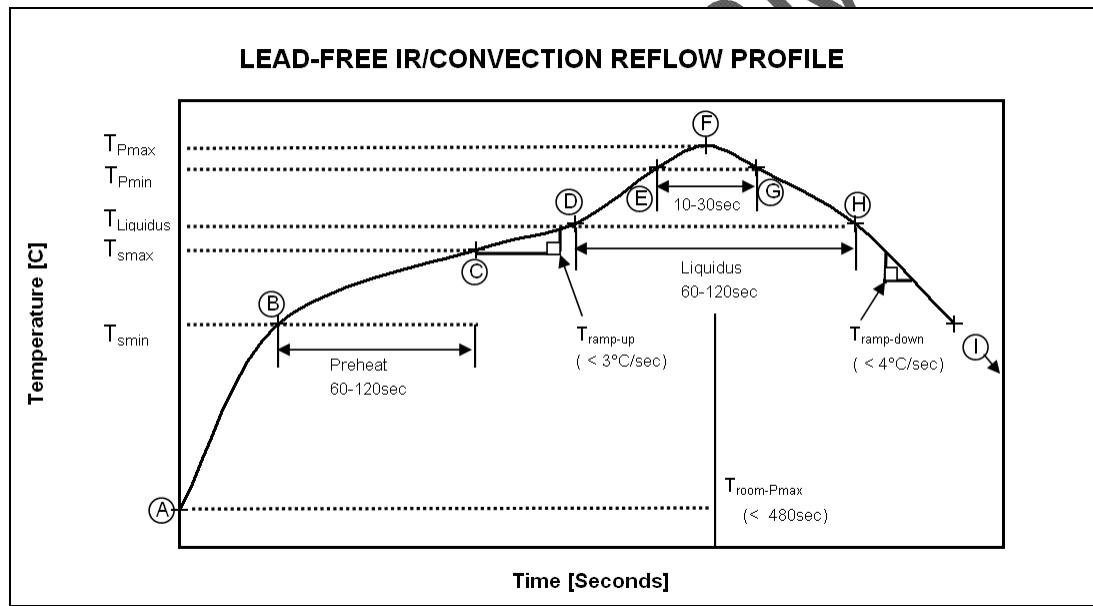
Restrict all device handling to ESD protected work areas that measure less than 200V static charge. Ensure that all workstations and personnel are properly grounded to prevent ESD.

11.4.8 Reflow Specification

Qualification Reflow: The MPU-60X0 was qualified in accordance with IPC/JEDEC J-STD-020D.01. This standard classifies proper packaging, storage and handling in order to avoid subsequent thermal and mechanical damage during the solder reflow attachment phase of assembly. The classification specifies a sequence consisting of a bake cycle, a moisture soak cycle in a temperature humidity oven, followed by three solder reflow cycles and functional testing for qualification. All temperatures refer to the topside of the QFN package, as measured on the package body surface. The peak solder reflow classification temperature requirement is (260 +5/-0°C) for lead-free soldering of components measuring less than 16 mm in thickness.

Production Reflow: Check the recommendations of your solder manufacturer. For optimum results, production solder reflow processes should reduce exposure to high temperatures, and use lower ramp-up and ramp-down rates than those used in the component qualification profile shown for reference below.

Production reflow should never exceed the maximum constraints listed in the table and shown in the figure below that were used for the qualification profile, as these represent the maximum tolerable ratings for the device.



Approved IR/Convection Solder Reflow Curve



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.2
Release Date: 11/16/2011

Temperature Set Points for IR / Convection Reflow Corresponding to Figure Above

Step	Setting	CONSTRAINTS		
		Temp (°C)	Time (sec)	Rate (°C/sec)
A	T _{room}	25		
B	T _{Smin}	150		
C	T _{Smax}	200	60 < t _{BC} < 120	
D	T _{Liquidus}	217		r _(TLiquidus-TPmax) < 3
E	T _{Pmin} [255°C, 260°C]	255		r _(TLiquidus-TPmax) < 3
F	T _{Pmax} [260°C, 265°C]	260	t _{AF} < 480	r _(TLiquidus-TPmax) < 3
G	T _{Pmin} [255°C, 260°C]	255	10 < t _{EG} < 30	r _(TPmax-TLiquidus) < 4
H	T _{Liquidus}	217	60 < t _{DH} < 120	
I	T _{room}	25		

Notes:

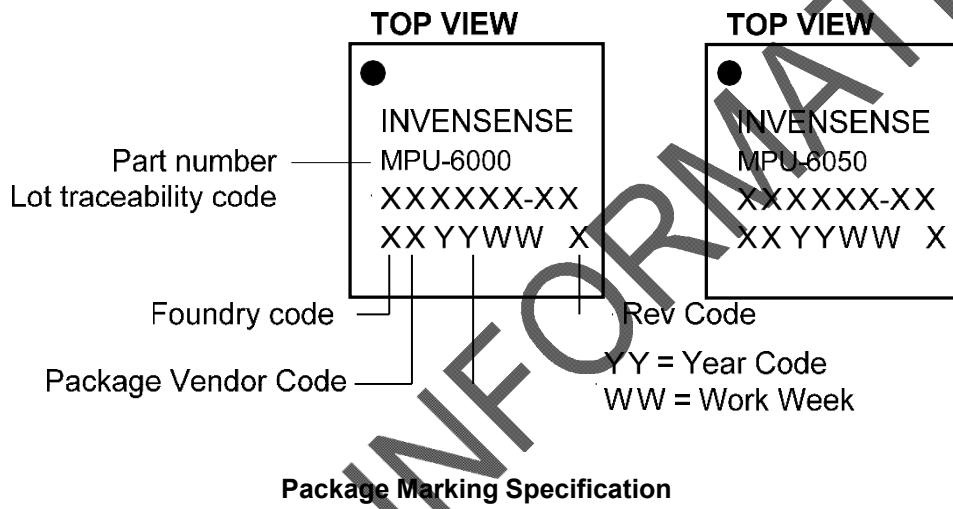
- For users T_{Pmax} must not exceed the Classification temperature (260°C).
- For suppliers T_{Pmax} must equal or exceed the classification temperature.

11.5 Storage Specifications

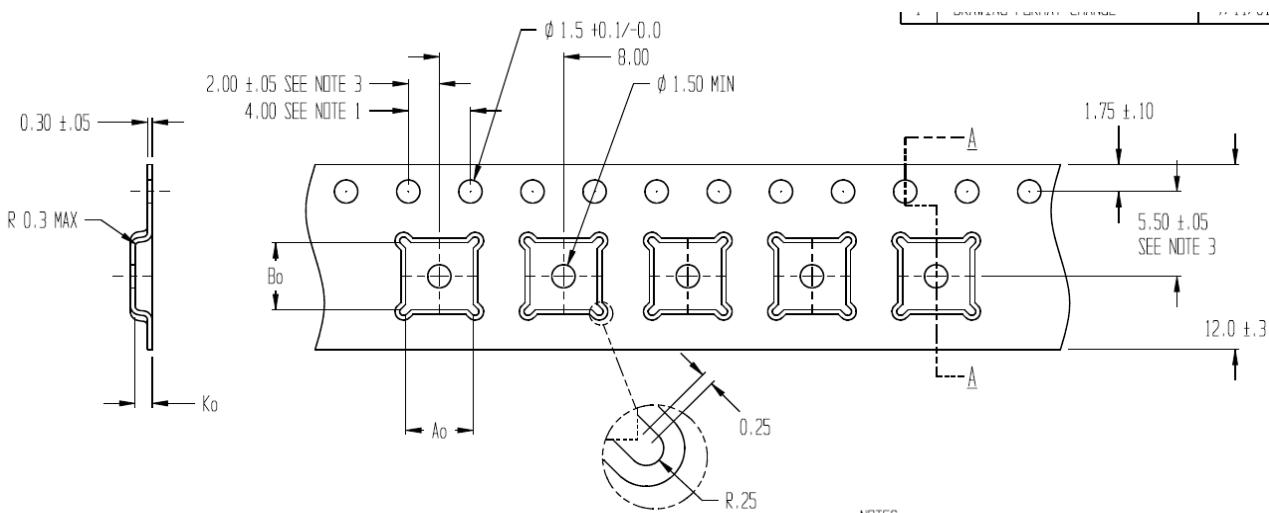
The storage specification of the MPU-60X0 conforms to IPC/JEDEC J-STD-020D.01 Moisture Sensitivity Level (MSL) 3.

Calculated shelf-life in moisture-sealed bag	12 months -- Storage conditions: <40°C and <90% RH
After opening moisture-sealed bag	168 hours -- Storage conditions: ambient ≤30°C at 60%RH

11.6 Package Marking Specification



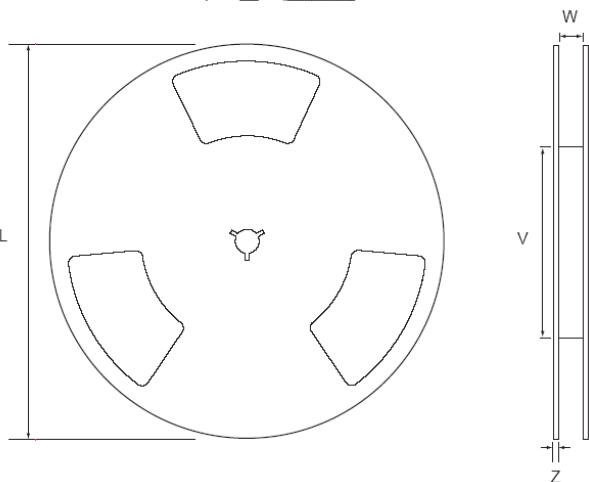
11.7 Tape & Reel Specification



NOTES:

1. 10 SPROCKET HOLE PITCH CUMULATIVE TOLERANCE ± 0.2
2. CAMBER IN COMPLIANCE WITH EIA 481
3. POCKET POSITION RELATIVE TO SPROCKET HOLE MEASURED AS TRUE POSITION OF POCKET, NOT POCKET HOLE

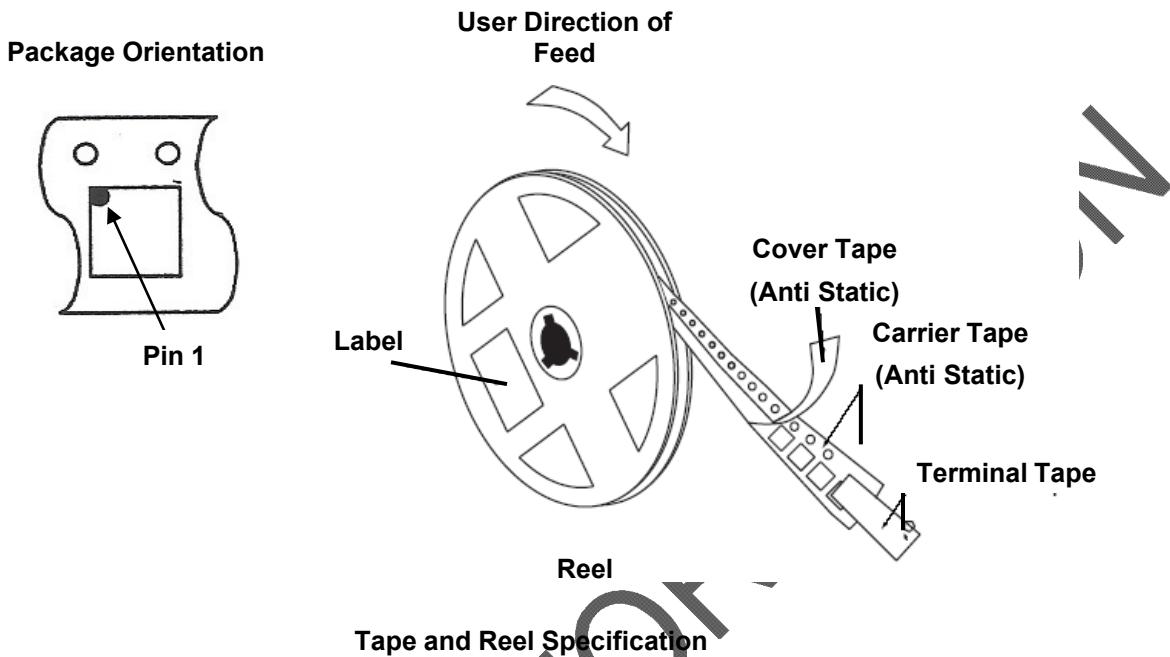
Tape Dimensions



Reel Outline Drawing

Reel Dimensions and Package Size

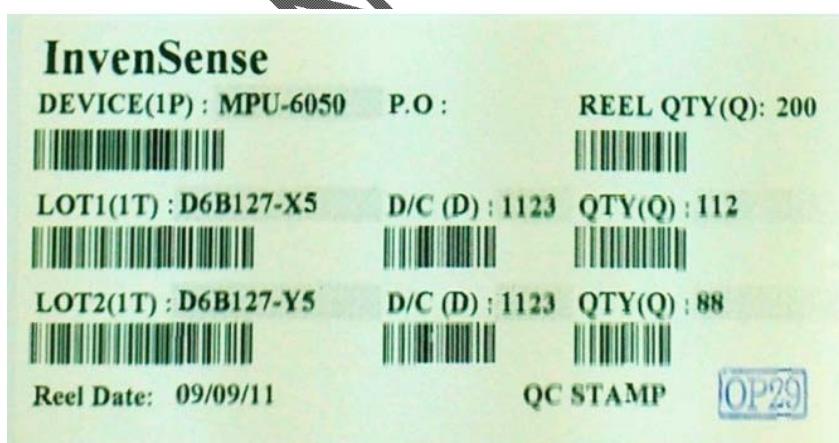
PACKAGE SIZE	REEL (mm)			
	L	V	W	Z
4x4	330	100	13.2	2.2



Reel Specifications

Quantity Per Reel	5,000
Reels per Box	1
Boxes Per Carton (max)	3
Pcs/Carton (max)	15,000

11.8 Label



Barcode Label



Location of Label on Reel

11.9 Packaging



REEL – with Barcode & Caution Labels



Vacuum Sealed Moisture Barrier Bag with ESD, MSL Caution & Barcode Labels



MSL Label



Caution Label



ESD Label



Barcode Labeled Pizza Box



Pizza Box



Pizza Boxes in Foam-Lined Shipper Box



12 Reliability

12.1 Qualification Test Policy

InvenSense's products complete a Qualification Test Plan before being released to production. The Qualification Test Plan for the MPU-60X0 followed the JEDEC 47G.01 Standard, "Stress-Test-Driven Qualification of Integrated Circuits," with the individual tests described below.

12.2 Qualification Test Plan

Accelerated Life Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
High Temperature Operating Life (HTOL/LFR)	JEDEC JESD22-A108C, Dynamic, 3.63V biased, $T_j > 125^\circ\text{C}$ [read-points 168, 500, 1000 hours]	3	77	(0/1)
Highly Accelerated Stress Test, Unbiased ⁽¹⁾ (HAST)	JEDEC JESD22-A118 Condition A, 130°C, 85%RH, 33.3 psia., unbiased, [read-point 96 hours]	3	77	(0/1)
High Temperature Storage Life (HTS)	JEDEC JESD22-A103C, Cond. A, 125°C, Non-Biased Bake [read-points 168, 500, 1000 hours]	3	77	(0/1)

Device Component Level Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
ESD-HBM	JEDEC JESD22-A114E (2kV)	1	3	(0/1)
ESD-MM	JEDEC JESD22-A115-A, (200V)	1	3	(0/1)
Latch Up	JEDEC JESD78B Class II (2), 125°C; Level A $\pm 100\text{mA}$	1	6	(0/1)
Mechanical Shock	JEDEC JESD22-B104C, Mil-Std-883H, method 2002.5, Cond. E, 10,000g's, 0.2ms, $\pm X, Y, Z$ – 6 directions, 5 times/direction	3	5	(0/1)
Vibration	JEDEC JESD22-B103B, Variable Frequency (random), Cond. B, 5-500Hz, X, Y, Z – 4 times/direction	3	5	(0/1)
Temperature Cycling (TC) ⁽¹⁾	JEDEC JESD22-A104D Condition N, [-40°C to +85°C], Soak Mode 2 [5'], 100 cycles	3	77	(0/1)

Board Level Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
Board Mechanical Shock	JEDEC JESD22-B104C, Mil-Std-883H, method 2002.5, Cond. E, 10000g's, 0.2ms, $\pm X, Y, Z$ – 6 directions, 5 times/direction	1	5	(0/1)

(1) Tests are preceded by MSL3 Preconditioning in accordance with JEDEC JESD22-A113F



13 Environmental Compliance

The MPU-60X0 is RoHS and Green compliant and is in full environmental compliance as evidenced in report HS-MPU-6000A, Materials Declaration Data Sheet.

Environmental Declaration Disclaimer:

InvenSense believes this environmental information to be correct but cannot guarantee accuracy or completeness. Conformity documents for the above component constituents are on file. InvenSense subcontracts manufacturing and the information contained herein is based on data received from vendors and suppliers, which has not been validated by InvenSense.

ADVANCE INFORMATION



This information furnished by InvenSense is believed to be accurate and reliable. However, no responsibility is assumed by InvenSense for its use, or for any infringements of patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. InvenSense reserves the right to make changes to this product, including its circuits and software, in order to improve its design and/or performance, without prior notice. InvenSense makes no warranties, neither expressed nor implied, regarding the information and specifications contained in this document. InvenSense assumes no responsibility for any claims or damages arising from information contained in this document, or from the use of products and services detailed therein. This includes, but is not limited to, claims or damages based on the infringement of patents, copyrights, mask work and/or other intellectual property rights.

Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by implication or otherwise under any patent or patent rights of InvenSense. This publication supersedes and replaces all information previously supplied. Trademarks that are registered trademarks are the property of their respective companies. InvenSense sensors should not be used or sold in the development, storage, production or utilization of any conventional or mass-destructive weapons or for any other weapons or life threatening applications, as well as in any other life critical applications such as medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, disaster prevention and crime prevention equipment.

InvenSense®, MotionCommand®, TouchAnywhere®, and AirSign® are registered trademarks of InvenSense, Inc. MPU™, MPU- 6000™, MPU-6050™, MPU-60X0™, Motion Processing Unit™, MotionFusion™, MotionProcessing™, MotionApps™, Digital Motion Processor™, Digital Motion Processing™, DMP™, BlurFree™, and InstantGesture™ are trademarks of InvenSense, Inc.

