Understanding wCK module and C programming with RoboBuilder

RoboBuilder Co., Ltd.

- **TITLE –**

1. Overview

1.1 Introduction

This tutorial is for intermediate-advanced RoboBuilder user, who is already accustomed with MotionBuilder, ActionBuilder GUI based programming, and for controlling the individual wCK module of robot directly by C language programming.

First of all, user needs to understand wCK module (hereinafter, “wCK”) properly. wCK is not a just a part of Robot, but also, it has various useful functions itself. More robot project can be done as user uses these wCK featured functions.

If, user understood wCK fully, let’s start the C programming with RoboBuilder.
By realizing all wCK functions with C programming, user learns C programming as well.
User learns from link up with MotionBuilder (*.rbm) file to humanoid robot maze escape
and IR communication programming in this book.

Lastly, basic C programming grammar and simple methods are introduced in this book in
order to help to C programming beginners.

1.2 Structure

Chapter 2.1. Change wCK ID

It describes how to change wCK ID parameter configuration.

Chapter 2.2. Change various wCK parameters

It describes how to configure ID, Baud Rate, Over Load, Speed, etc parameters.

Chapter 2.3. Set PID Gain and Check wCK response time

It describes PID control theory and how to adjust PID value in accordance with PID gains.

Chapter 2.4. wCK free motion programming

It describes how to do the wCK direct programming without controller.

Chapter 3.1. Firmware and C programming

It describes general firmware's definition and C programming structure for firmware.

Chapter 3.2. RBC hardware and I/O MAP

It describes RBC hardware structure and I/O MAP

Chapter 3.3. C Programming with Motion file

It describes provided C programming project file structure.

Chapter 3.4. Control RBC LED (Project 3-4 RBC_LED)

It describes how to control RBC LED.

Chapter 3.5. Control wCK Position (Project 3-5 wCK Position)

It describes how to control wCK position for Robot motion.

Chapter 3.6. Control wCK LED (Project 3-6 wCK_LED)

It describes how to control wCK LED and Command packet communication.

Chapter 3.7. Configuration of wCK parameter (Project 3-7 wCK_Parameter)

It describes how to configure wCK parameter with C programming.

Chapter 3.8. Make C program with Motion file (Project 3-8 Motion_Program)

It describes how to include motion file into C program.

Chapter 3.9. Using IR controller and Understanding C program (Project 3-9 IR_RemoteCon)

It describes IR remote controller principles and how to change the configuration.

Chapter 3.10. Humanoid Robot Maze Escape Programming (Project 3-10 Maze)

It describes how to program maze escape programming for humanoid robot by using RoboBuilder distance sensor.

Chapter 4. C Programing Summary

It describes basic C programming grammar for beginner.

1.3 Requirement

- RoboBuilder Kit (5710K or 5720T model)
- CodeVision-AVR C compiler

CodeVision C compiler can be purchased from www.yklogic.co.kr or www.hpinfotech.ro.
Free released version for students is not enough to support the examples in this book.

2. Understanding wCK

2.1 Change wCK ID

In RoboBuilder kit, wCK is set own ID from 0 to 15.

If necessary, user can change ID No. from 0 to 30 as user want to change.

※ Requirements

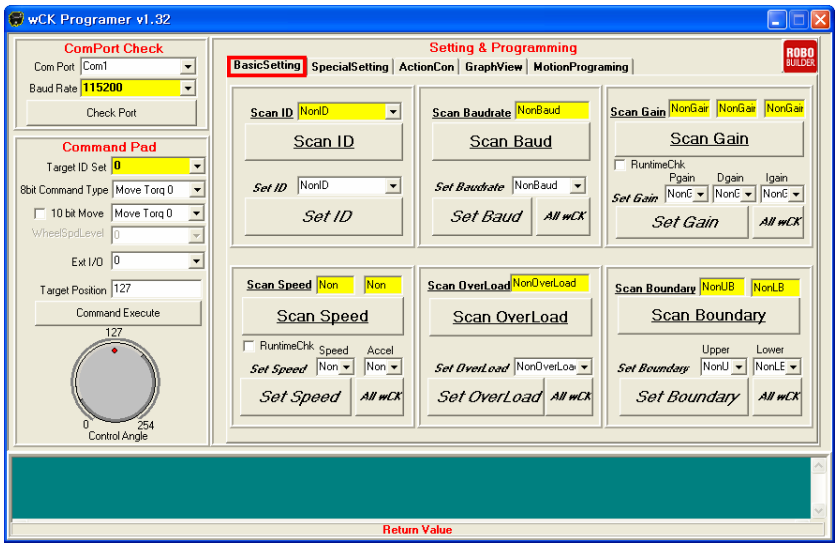
- RBC (Control Box) : 1 EA
- wCK : 1 EA
- wCK Cable : 1 EA
- RS-232 Serial Cable (PC cable) : 1 EA
- Window XP based PC and wCK Programmer software

Let's change wCK ID 2 to ID 0. Connect wCK, RBC, PC cable and Power adapter as the below.

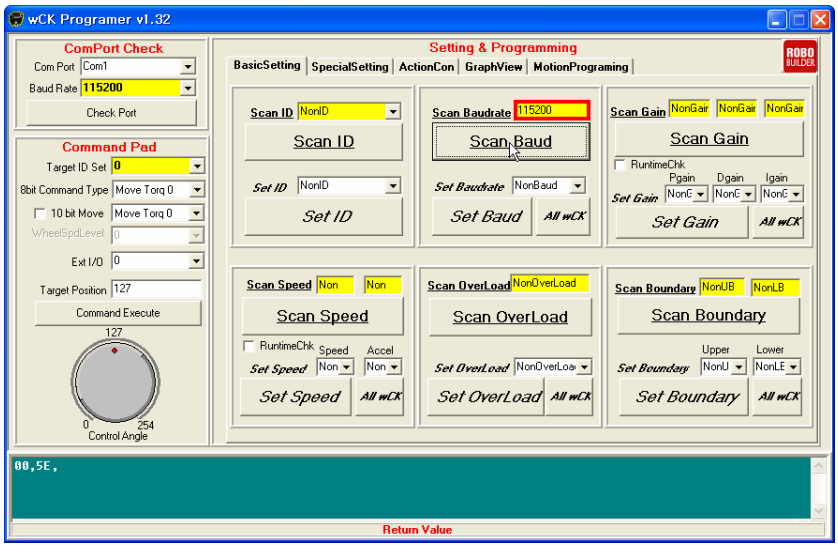
※ User can connect any of connector in RBC box. But only one wCK should be connected.

Press PF2 button then, power on RBC box. Then RBC Box goes into PC control mode. PF1 LED (Blue), PF2 LED (Orange) is ON together. (In wCK Programmer ver 1.34 or higher, this procedure is not needed.)

Click “Basic Setting” tap in wCK Programmer.

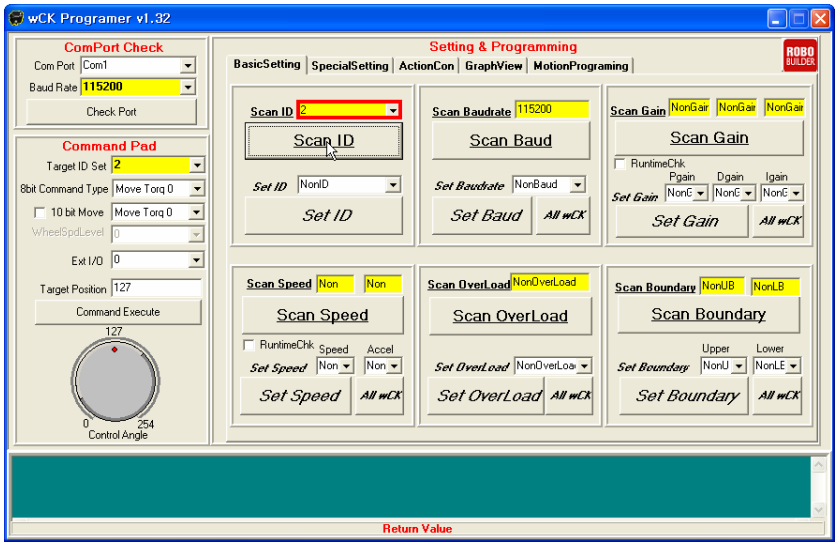


After connected with PC COM port, click “Scan Baud”, then wCK Baud Rate is shown as the below.

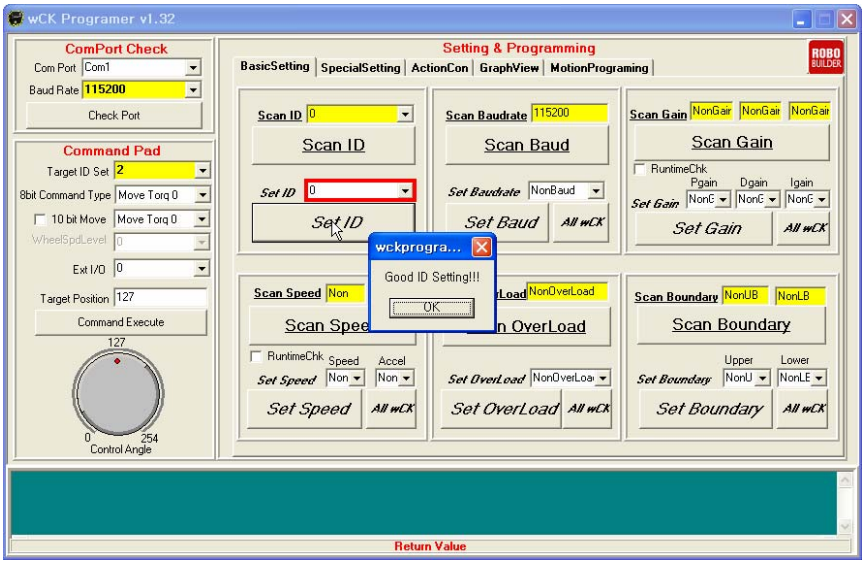


※ If “Try Again” message is shown, please check the wCK cable and PC cable connection.

If Scan ID button is clicked, it shows the present wCK ID as shown in the below.



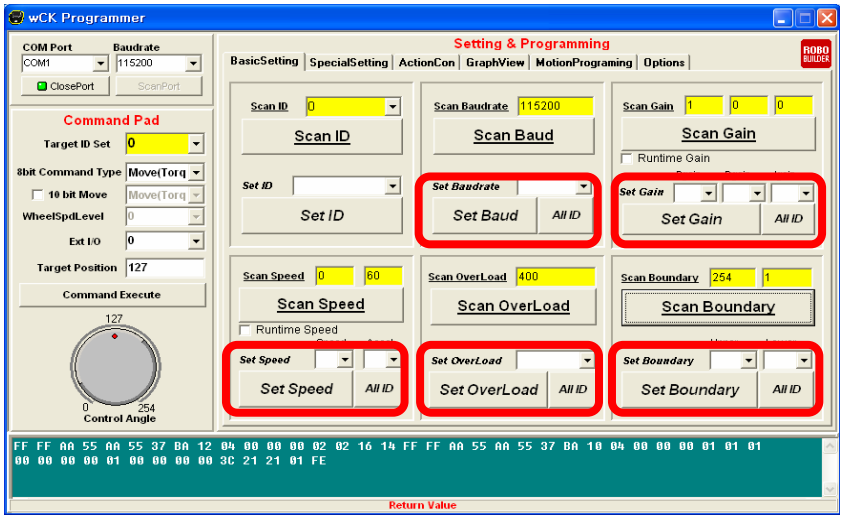
Select desired wCK ID No. in “Set ID” in drop box, then click “Set ID” button. It shows “Good ID Setting!!!” message, then new wCK ID is shown in Scan ID drop box.



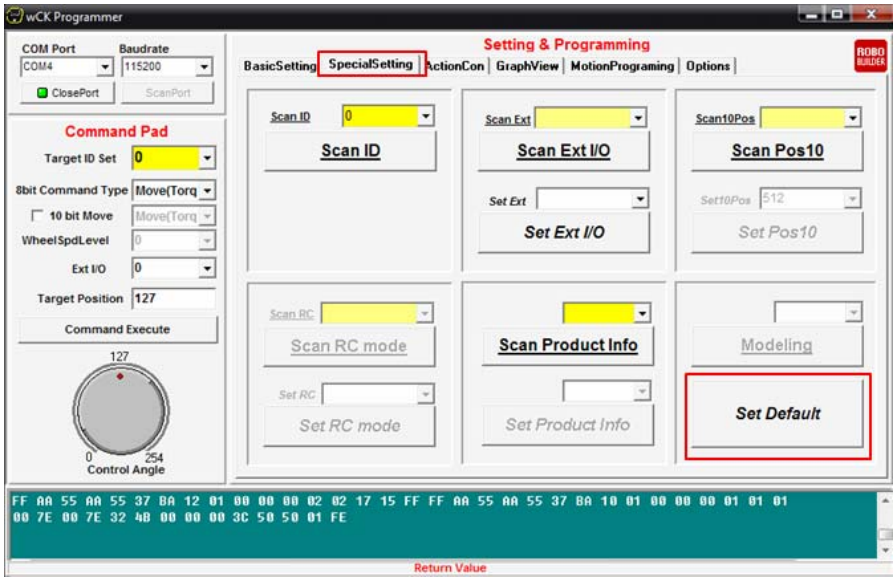
Now, wCK ID has changed.

2.2 Change various wCK Parameters

User can change not just wCK ID, but also Speed, Acceleration, Over Load, Boundary, PID gain, etc. All these parameter configuration is done in same way.



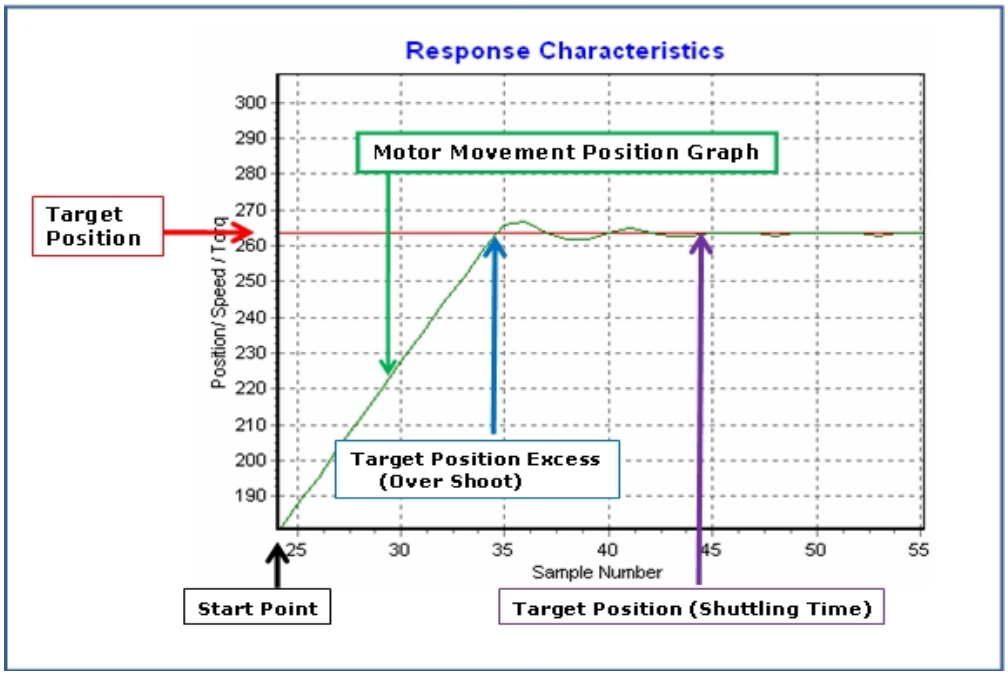
Changed parameter can be initialized to factory setting value as user click “Set Default” button in “Special Setting” tap.



As “Set Default” function is used to initialize the various parameters at one time, it is useful to check whether the parameters have been changed.

2.3 PID Gain Tuning and wCK Response Feature

When user gives the command to wCK to target point, wCK moves (rotates) to target position. DC motors have the movement value as shown in the below.



Motor needs some time to move to the target point. Because of this reason, many motor control method has been studied in order to reduce this time. wCK is applied PID control method, which is used in many industrial facility. Further, user can configure PID Gain parameter and study the various motor features as wCK Programmer has GraphView function.

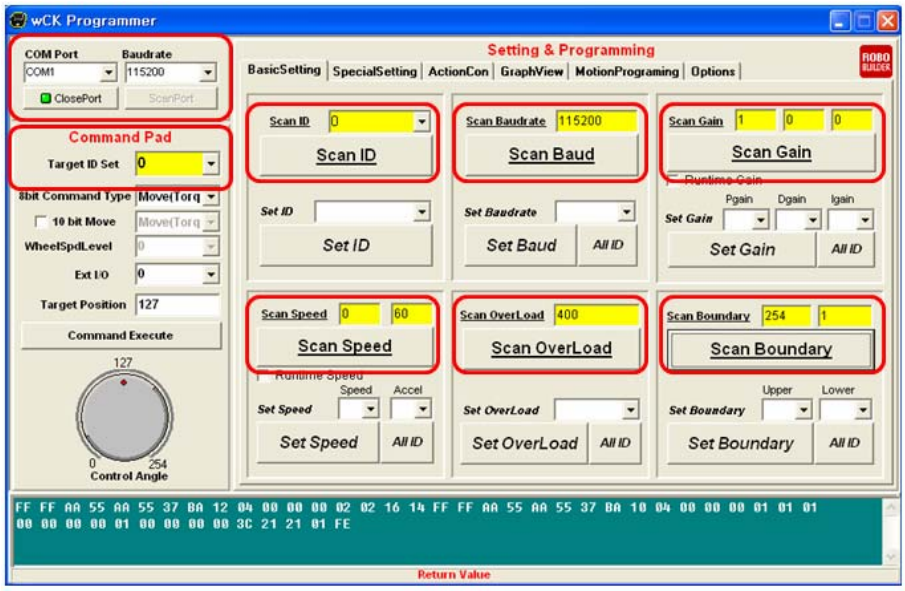
1. Requirements

- RBC Box : 1 EA
- wCK : 1 EA
- wCK Cable : 1 EA
- RS-232 Serial Cable(PC cable) : 1 EA
- Window XP based PC and wCK Programmer software

Please refer to the wCK Programmer user manual during this procedure. Connect the wCK, RBC, Power Adapter and PC cable as shown in the below.

※ wCK can be connected any connector in RBC Box.

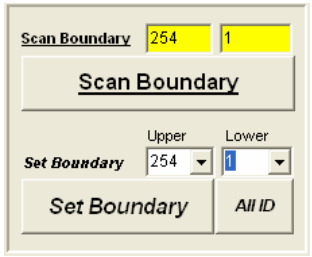
Run wCK Programmer software.



Click  button after select available COM Port.

Click Scan ID, Scan Baud, Scan Gain, Scan Speed, Scan Boundary buttons in order. It shows the present values of wCK.

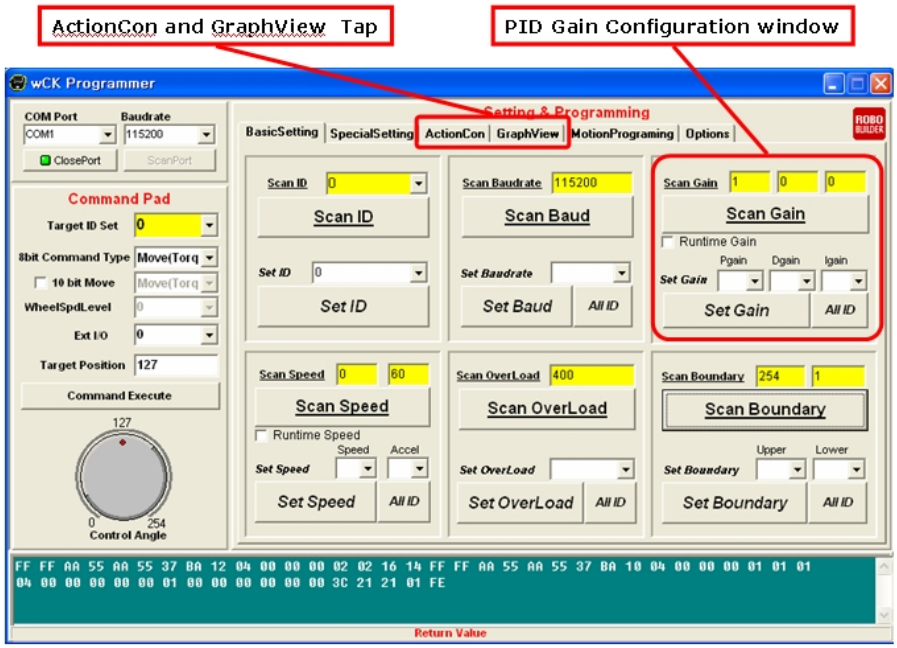
For PID control study, input Upper 254, Lower 1, then clickt “Set Boundary”.
If Boundary width is narrow, wCK movement range becomes narrow. Therefore, set maximum width as “(254, 1)”.



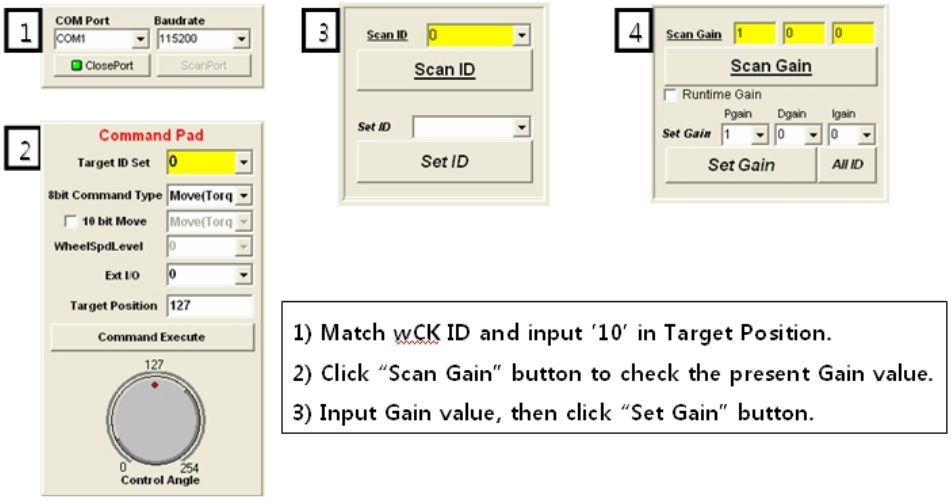
※ Boundary means wCK movement range width.

2. wCK Programmer function for PID control

- BasicSetting Dialogue to set PID Gain.
- AcionCon Dialogue to move wCK to target point. .
- GraphView function to check wCK movement.

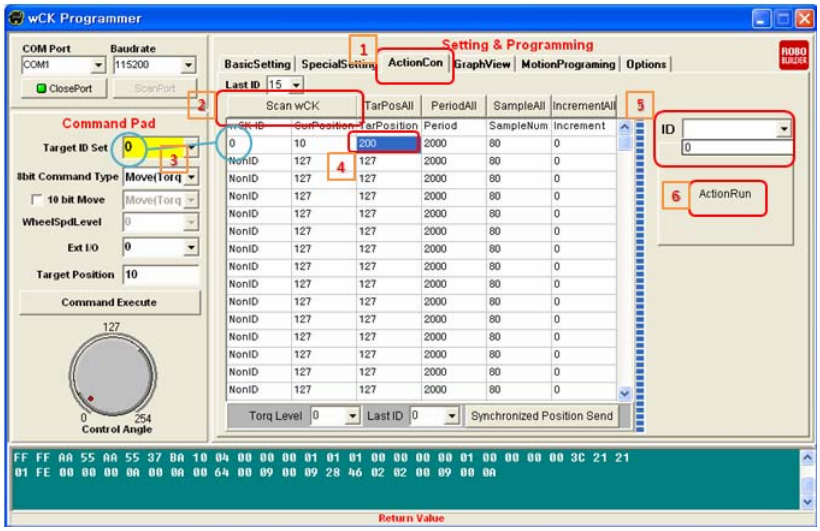


① Set wCK PID Gain.



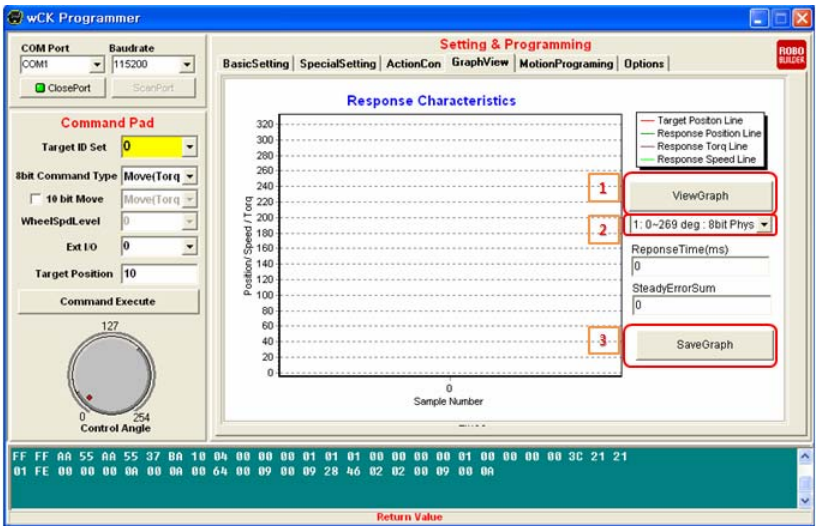
1) Match wCK ID and input '10' in Target Position.
2) Click "Scan Gain" button to check the present Gain value.
3) Input Gain value, then click "Set Gain" button.

② Command wCK movement to target point



A. Click “ActionCon” tap.
B. Click “Scan wCK” button, and wait around 10 seconds.
C. It shows the connected wCK ID.
D. Input desired wCK “Target Position”.
E. Select that wCK ID.
F. Click “ActionRun” button.

③ Check “GraphView”

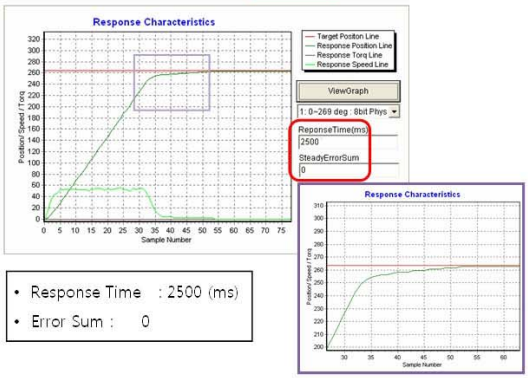


A. Click “GraphView” tap
B. Click “ViewGraph” button. Then it shows Graph.
C. Select proper scale.
D. Click “SaveGraph” button to save the BMP image, if necessary.

After check the wCK response graph, initialize wCK position in order to set Gain value. In Command Pad panel, click “Command Execute” to initialize the wCK position.

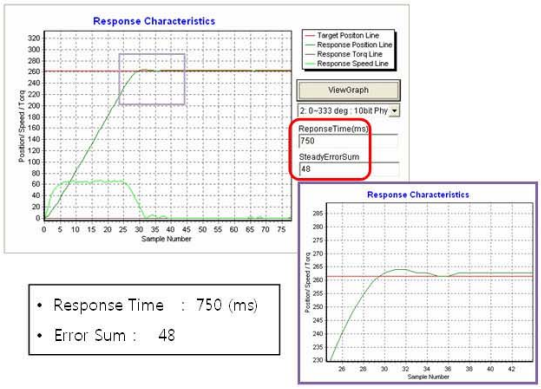
As user follows below examples, user can understand wCK movement features in accordance with PID parameters.

Graph #1 **[*P gain : 10* / D gain : 0 / I gain : 0]**



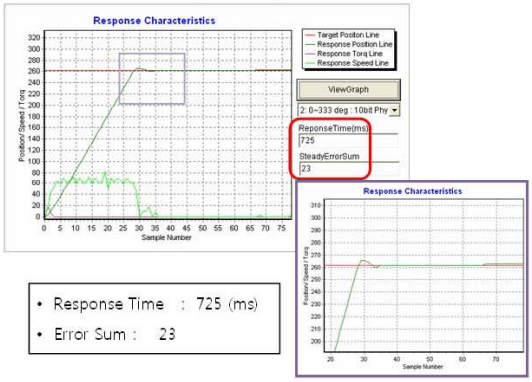
- Response Time : 2500 (ms)
- Error Sum : 0

Graph #2 **[*P gain : 20* / D gain : 0 / I gain : 0]**



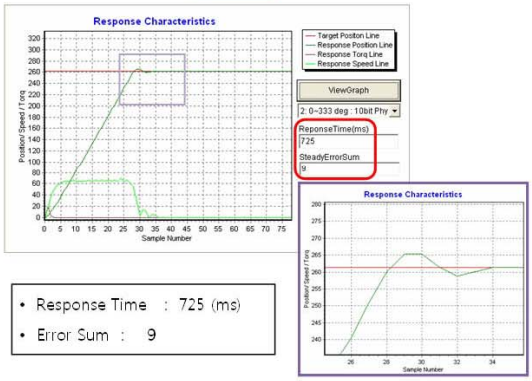
- Response Time : 750 (ms)
- Error Sum : 48

Graph #3 **[*P gain : 30* / D gain : 0 / I gain : 0]**



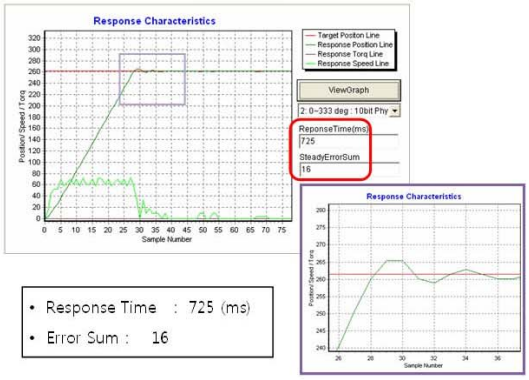
- Response Time : 725 (ms)
- Error Sum : 23

Graph #4 **[*P gain : 60* / D gain : 0 / I gain : 0]**



- Response Time : 725 (ms)
- Error Sum : 9

Graph #5 **[*P gain : 80* / D gain : 0 / I gain : 0]**

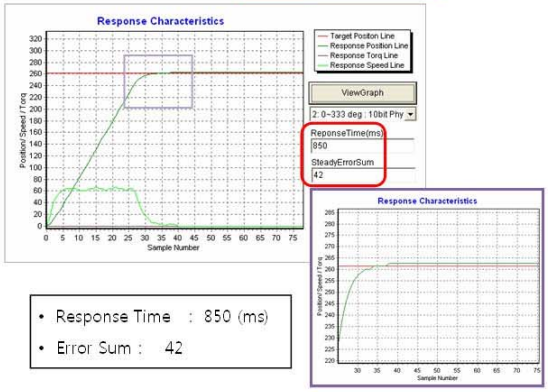


- Response Time : 725 (ms)
- Error Sum : 16

As P gain value is increased, wCK response time is decreased. But “Overshoot” value is increased as shown in the above graph. This means, it gives faster response, but target position value is unstable.

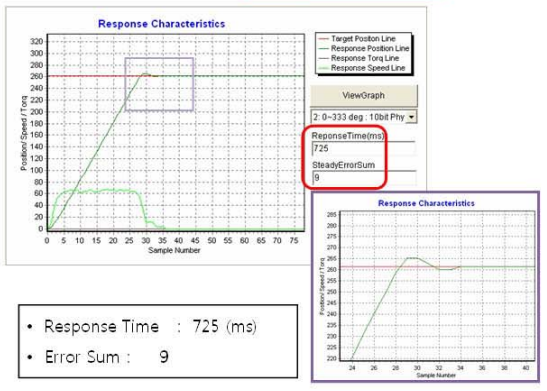
Let’s change P gain and D gain together.

Graph #6 [*P gain : 10* / *D gain : 30* / I gain : 0]



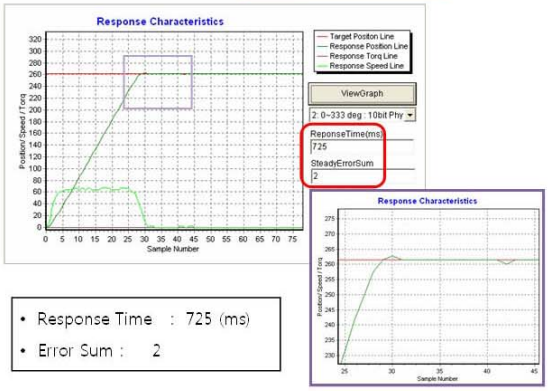
- Response Time : 850 (ms)
- Error Sum : 42

Graph #7 [*P gain : 50* / *D gain : 10* / I gain : 0]



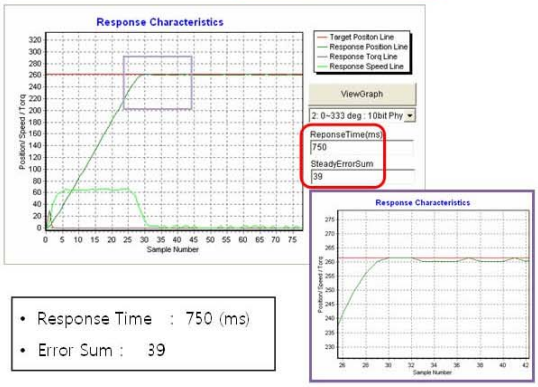
- Response Time : 725 (ms)
- Error Sum : 9

Graph #8 [*P gain : 50* / *D gain : 50* / I gain : 0]



- Response Time : 725 (ms)
- Error Sum : 2

Graph #9 [*P gain : 50* / *D gain : 70* / I gain : 0]

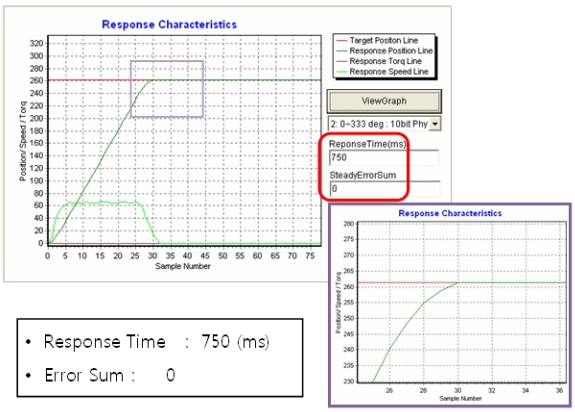


- Response Time : 750 (ms)
- Error Sum : 39

If user change P gain and D gain together, “overshoot’ value is decreased relatively.

But target position value is still unstable.

Graph #10 [*P gain : 40 / D gain : 70 / I gain : 2*]



- Response Time : 750 (ms)
- Error Sum : 0

Let's change P, D and I gain value together. wCK response time is decreased, there is no "overshoot" value and it goes target position exactly. wCK response feature would be much improved if PID gain is applied, if user use PID tune-up method.

P, I, D has each characteristics, respectively, as shown in the below.

- P Gain (Proportion Gain) : It reduces wCK response time.
- I Gain (Integral Gain) : It reduces tolerance.
- D Gain (Differential Gain) : It reduces "overshoot" value and make it stable.

Just increasing the Gain value does NOT means that is has good response feature.
(In example graph 10, P gain is 40, this is not so high value.)

Each parameter affects each other. Therefore, users should tests several times to optimize the gain value, and do PID control study.

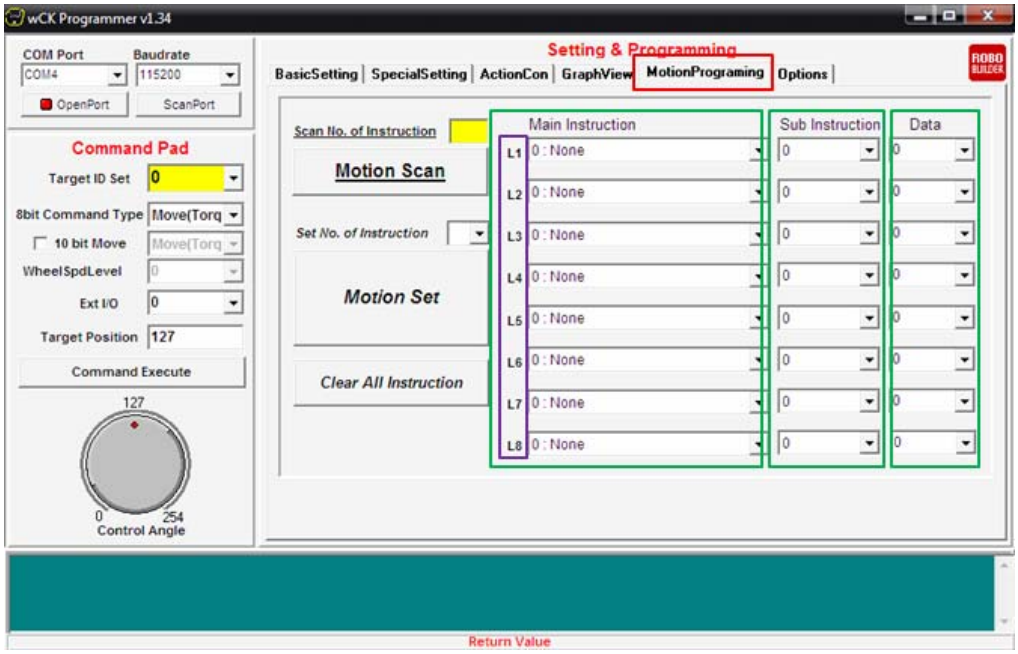
2.4 wCK Free Motion Programming

wCK has free motion programming function itself. When user does free motion programming, RBC should be selected “Non-Standard Platform (PF2 orange LED on)” mode. Then, wCK does movement independently as long as electrical power is provided.)

※ Requirements

- RBC : 1 EA / wCK : 2EA / Cds sensor : 1EA
- RS-232 Serial Cable (PC cable)
- wCK Programmer Tool

Below is MotionPrograming tap for wCK free motion programming.



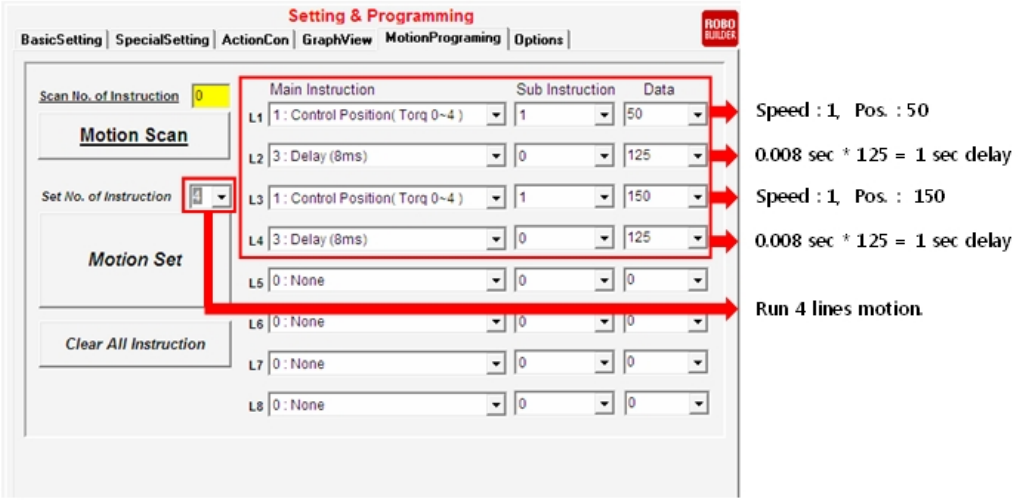
If you look at green box in wCK Programmer, you can see “Main Instruction”, “Sub Instruction”, “Data”, and it has 8 lines in left side from L1 to L8. wCK does 8 motions (L1~L8) in order.

Let's find out what is included and the meaning of "Main Instruction", "Sub Instruction", "Data" columns. The following is free motion programming logic table.

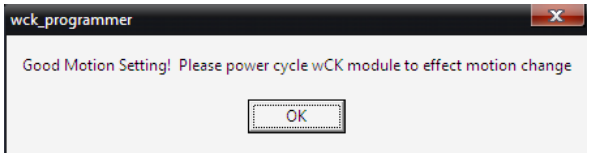
Motion Command(1 Byte)		Motion Data(1 Byte)
Main Instruction	**Sub Instruction**	**Data**
0: None	0	X
1: Position Control	Speed(0~4)	8 bit Position Value
2: Motion Type	1(Passive), 2(Power Down), 3(Wheel CCW), 4(Wheel CW)	The Speed value in Wheel Mode(0~15)
3: Delay Time(Max 4,095 ms)	Upper 4 bit delay value	Lower 8 bit delay value(in ms)
4: DIO	X	2 bit external port output value
5: Position Conditional Decision	1("==") 2(">") 3("<") 4(">=") 5("<=")	8 bit Position Value
6: A/D Conditional Decision	1("==") 2(">") 3("<") 4(">=") 5("<=")	8 bit external port A/D input value
7: No of Repetition	X	0 : Infinity, 1 to 254(No of repetition)
8: End of Program	X	X

Main Instruction had main instructions (0~8), and Sub Instruction is for detailed instruction for main instruction. Data is position or control values. User can input the values by keyboard.

Example 1] wCK movement from 100 degree to 160 degree in every second with speed 1.



If programming is finished as shown in the above, click “Motion Set” button to save in wCK.



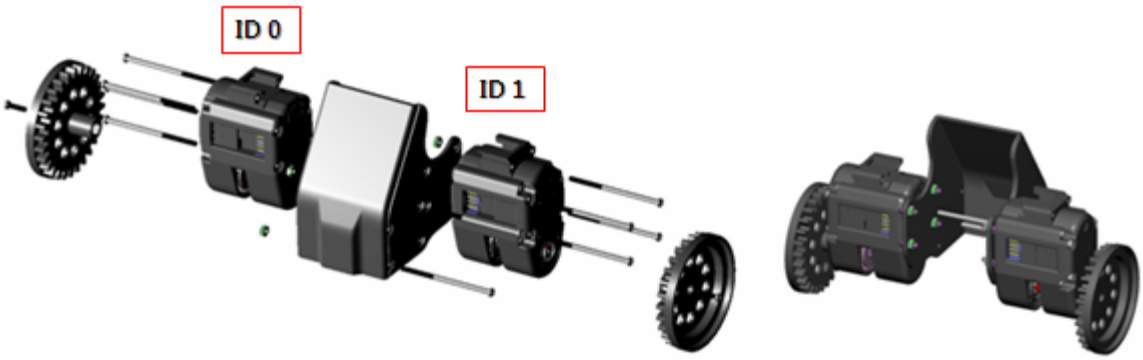
After above message box, click “OK”, then power off and on. Then wCK starts free motion itself if power is supplied.

In order to delete free motion programming, click “ Clear All Instruction button, and select “0” in Set No. of Instruction to delete the motion.

Let’s try simple robot with free motion programming.

[Wheel Robot 1]

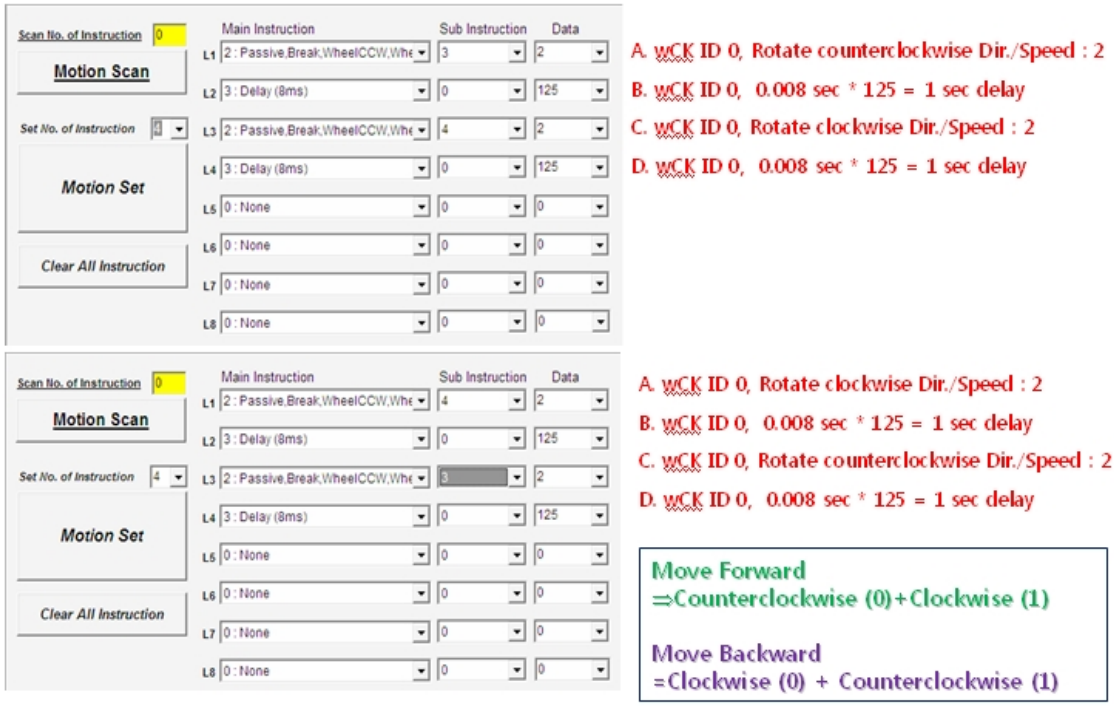
Prepare two wCKs, wCK ID0, wCK ID1. Assemble the wCK as shown in the below. This Robot will not move because it does NOT include RBC Box. However, it would be possible without RBC if you use free motion programming method.



Free motion programming is for each wCK. Therefore, two different programming are needed for two-wheeled Robot.

In order to move forward, one wCK should move in clockwise direction, and the other wCK should move counterclockwise direction. For backward movement, programming in vice versa.

Let's program two wCKs for forward movement and backward movement in every second interval when power is supplied.

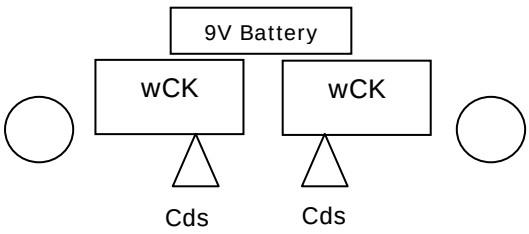


※ For ID 0 and ID 1 wCK , it should be programmed respectively.

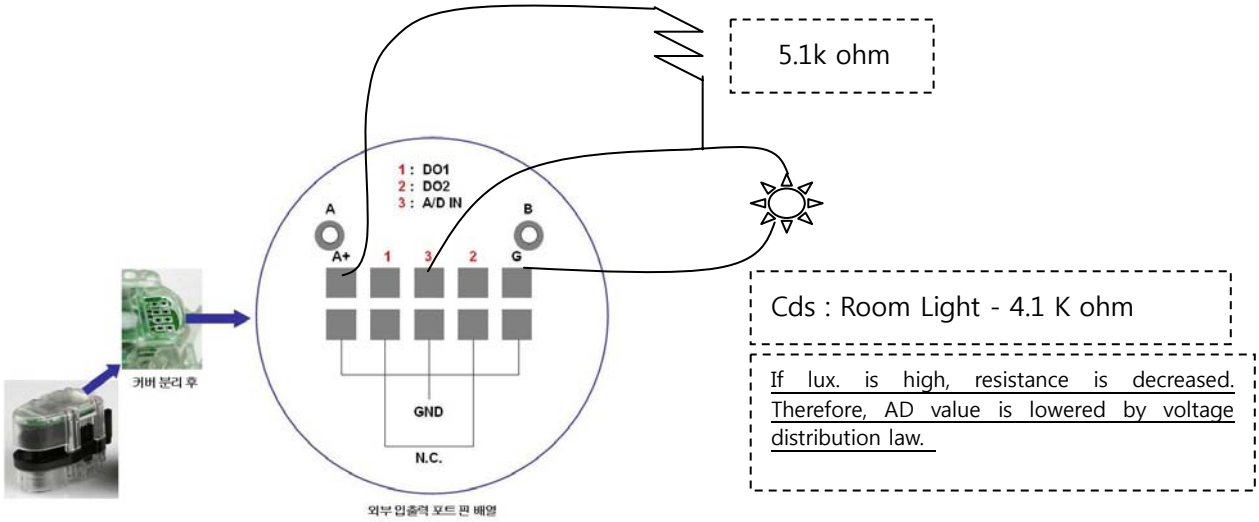
After programmed it as the above, supply the power into wCKs, then Robot will move forward and backward direction.

[Wheel Robot 2 – Sensor based Robot]

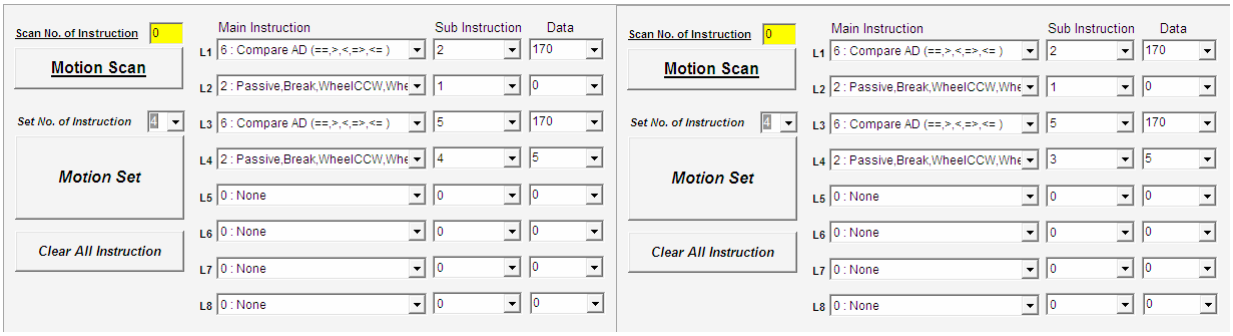
wCK has own I/O port itself. (Digital Output 2, AD Input 1). As you use AD Input, Robot will move as light level. In this example, "Cds sensor" is used.



For Cds sensor, solder with resistance 5.1k as shown in the below.



Program the wCK in wCK Programmer as shown in the below.



Robot measures wCK light level. Therefore, Robot moves depends on AD set value “170” as programmed in the above.

User can make various wCK application Robots if you use various wCK functions.

3. RBC Firmware Study for User Created Robot

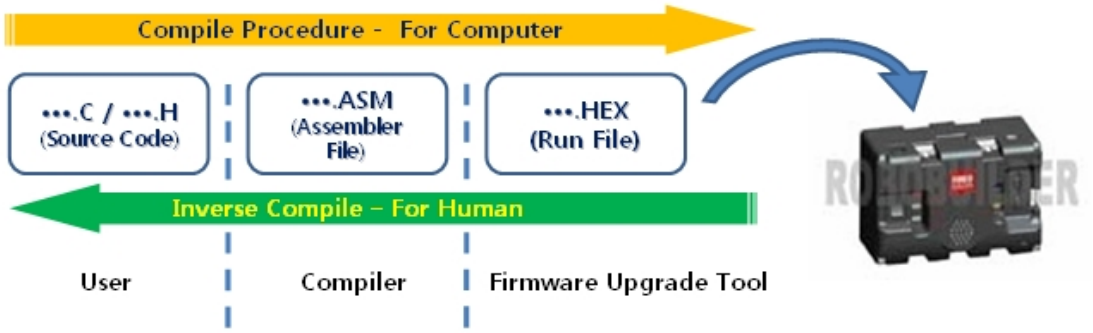
3.1 Firmware and C Program

Firmware is generally defined as a micro program that is saved in ROM memory. As an aspect of program, it is almost same as software, but it has much closer relationship with hardware. Firware is faster than software, but slower than hardware. And it is not comfortable for general user. It is quite dependent on hardware, therefore it should be changed when hardware component is changed. Firmware can be considered as a software that is close to hardware.

If you use RoboBuilder Firmware Upgrade Tool, user-created firmware can be saved into RBC and is possible to operate wCK. This motivates user to study firmware easily. Generally, user needs certain electronic boards or other hardware devices to study firmware, but RoboBuilder user does NOT need because RBC and RBC Firmware Tool are everything that user needs to study firmware.

If user-created firmware is not working at all, user can simply recover it with RoboBuilder published firmware version in RoboBuilder website.

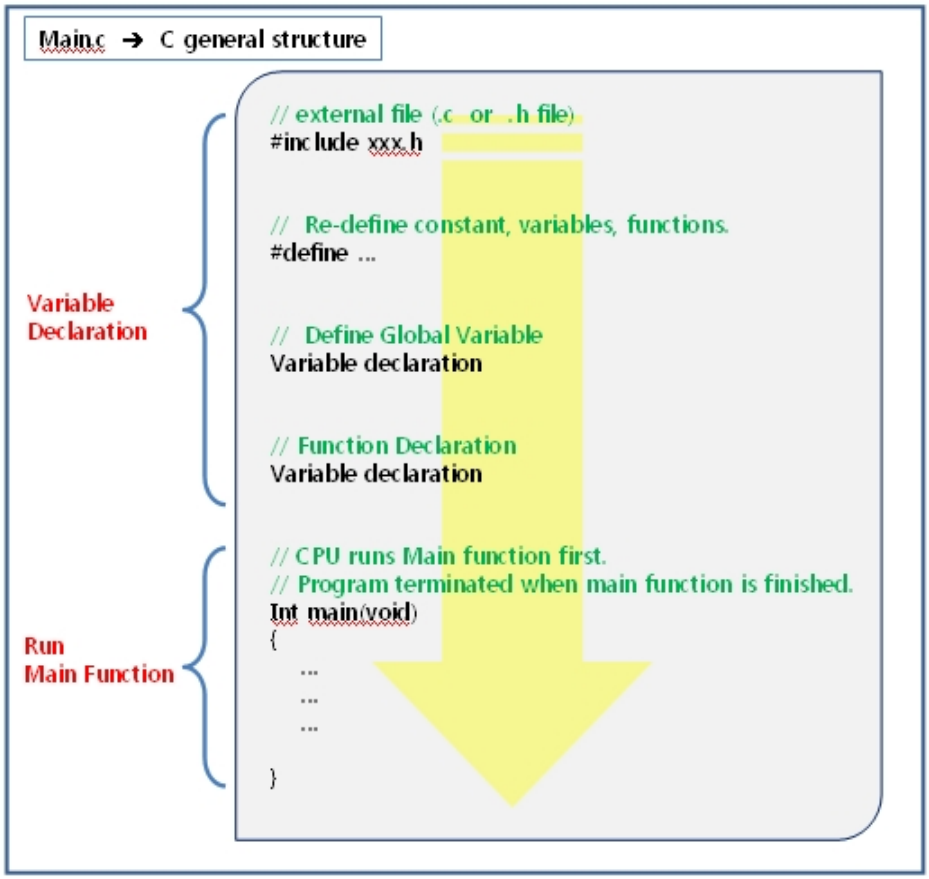
How general user can create own firmware or how it can upload the firmware into RBC box?
Let's find out in next pages. The below is the simple structure of RoboBuilder firmware generation.



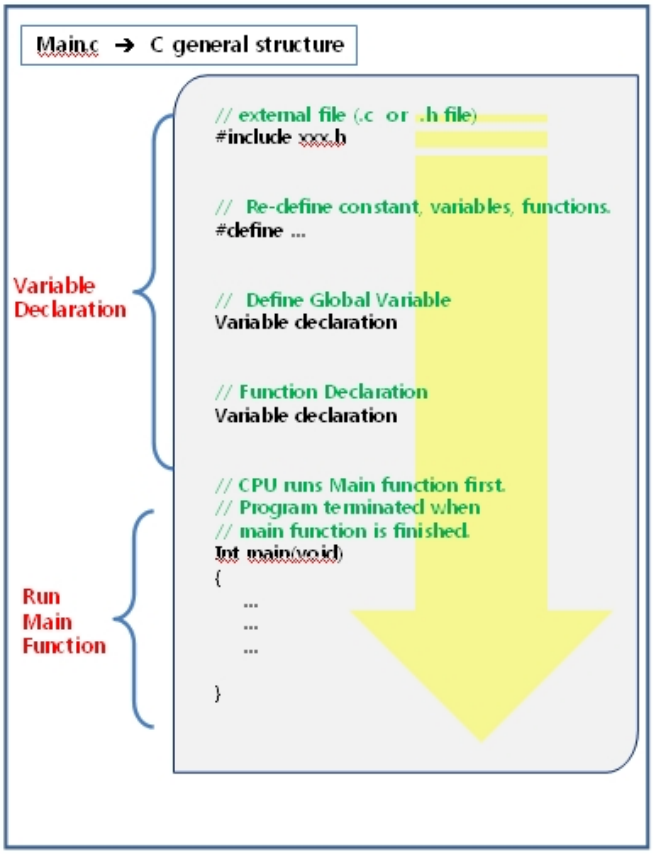
- Write C language program.
- User compiler (ex. CodeVisionAVR), to generate .ASM, or .HEX file.
- Upload firmware file (*.hex) into RBC Box.

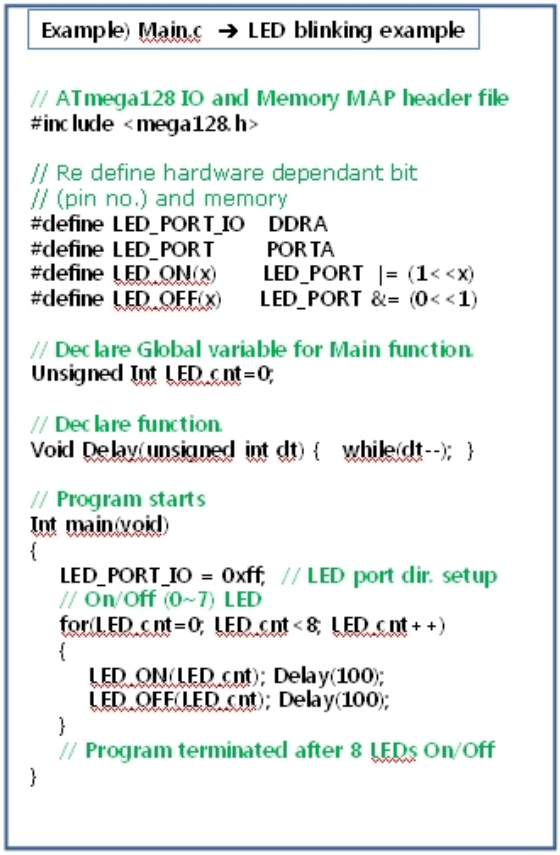
For C coding, user should know C language programming well. Please refer to the various C programming books in the bookstore.

In this example, we assume that user knows basic C programming method.



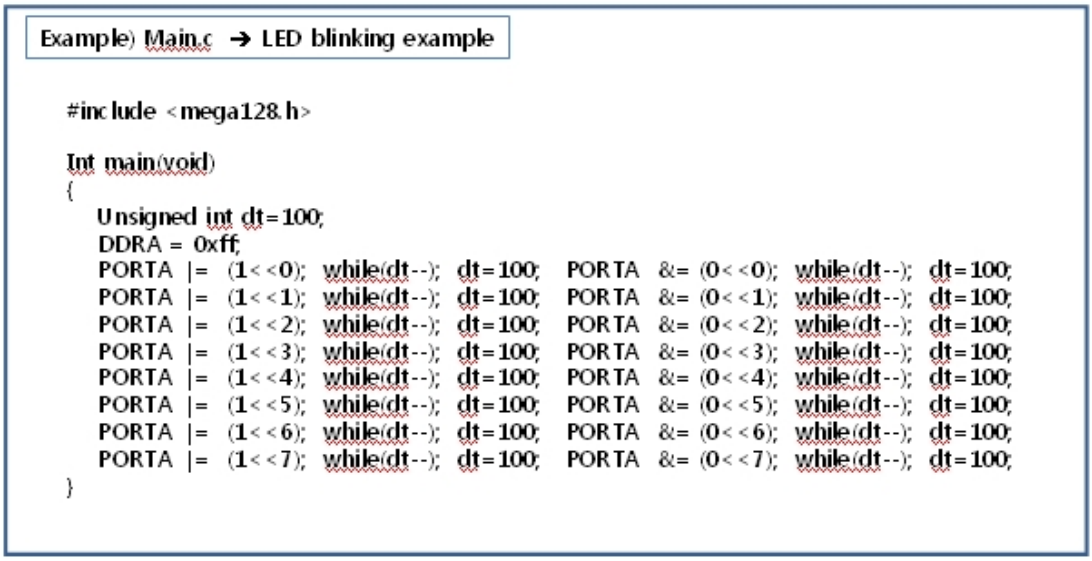
Below is LED ON/OFF program flows by C language.





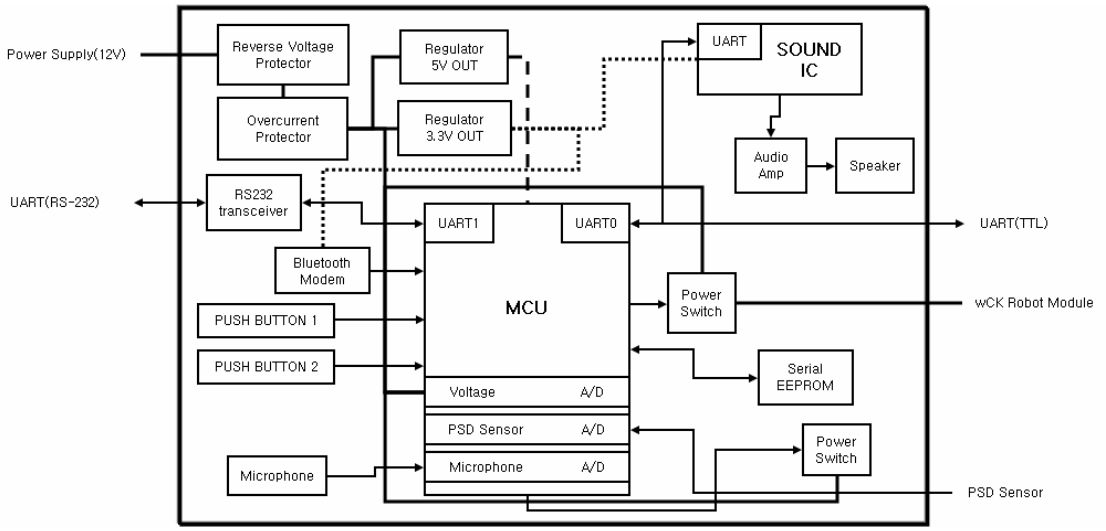
#include <mega128.h> is header file to use “DDRA”, “PORTA”.
Complicated sentences is redefined with “#define” word to understand program easily.

This program is for 8 LEDs On/Off one time. If “#define” word is not used, program would be like the below.



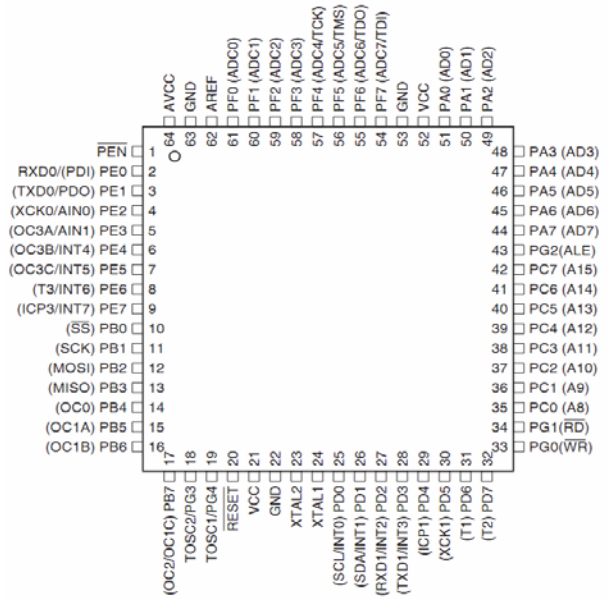
3.2 RBC Hardware Structure and I/O MAP

Below is RBC hardware block diagram.



User can understand how this hardware is connected. RBC is connected with power supply, RS-232 communication, TTL UART, wCK power line and PSD sensor.

Below is RBC micro-controller ATmega128 PIN allocation and I/O map.



PIN No.	PIN Name	I/O Dir	Description
1	PEN	X	Not Connected
2	(RXD0/PDI) PE0	I	Communcation with wCK, Sound IC (YNN Model)
3	(TXD0/PDO) PE1	O	Communcation with wCK, Sound IC (YNN Model)
4	(XCK0/AIN0) PE2	X	Not Connected
5	(OC3A/AIN1) PE3	O	Speaker Output (YNN Model)
6	(OC3B/INT4) PE4	X	3 Axis sensor (SCK)
7	(OC3C/INT5) PE5	X	3 Axis sensor (SDI)
8	(T3/INT6) PE6	I	IR Remote Controller Receiver Module (38kHz)
9	(ICP3/INT7) PE7	I	Bluetooth Signal Receiver
10	(SS)	X	Not Connected
11	(SCK) PB1	O	For ISP
12	(MOSI) PB2	O	Power Supply 24LC256T-I/SN (High : ON, Low : OFF)
13	(MISO) PB3	X	Not Connected
14	(OC0) PB4	O	Battery Charging (High : Charging ON, Low : Charging OFF)
15	(OC1A) PB5	O	PSD Sensor GP2Y0A21YK0F Power Control (High : ON, Low : OFF)
16	(OC1B) PB6	O	Sound IC Reset (High : Disabled, Low : Enabled)
17	(OC2/OC1C) PB7	X	Not Connected
18	TOSC2 / PG3	X	Not Connected
19	TOSC1 / PG4	X	Not Connected
25	(SCL/INT0) PD0	O	For Serial EEPROM Communication (SCL)
26	(SDA/INT1) PD1	I/O	For Serial EEPROM Communication (SDA)
27	(RXD1/INT2) PD2	I	Communication with PC
28	(TXD1/INT3) PD3	O	Communication with PC
29	(ICP1) PD4	X	Not Connected
30	(XCK1) PD5	X	Not Connected
31	(T1) PD6	X	Not Connected
32	(T2) PD7	O	wCK power supply (High : ON, Low : OFF)
33	(WR) PG0	X	Not Connected
34	(RD) PG1	X	Not Connected
35	(A8) PC0	X	Not Connected
36	(A9) PC1	X	Not Connected
37	(A10) PC2	X	Not Connected
38	(A11) PC3	X	Not Connected
39	(A12) PC4	X	Not Connected
40	(A13) PC5	X	Not Connected
41	(A14) PC6	X	Not Connected
42	(A15) PC7	O	Power LED (Red)
43	(ALE) PG2	O	Power LED (Green)
44	(AD7) PA7	O	Error LED (Red)
45	(AD6) PA6	O	Run LED (Green)
46	(AD5) PA5	O	Run LED (Blue)
47	(AD4) PA4	O	PF2 LED (Orange)
48	(AD3) PA3	O	PF1 LED (Red)
49	(AD2) PA2	O	PF1 LED (Blue)
50	(AD1) PA1	I	PF2 Switch (High : Not pressed, Low : Pressed)
51	(AD0) PA0	I	PF1 Switch (High : Not pressed, Low : Pressed)
54	(ADC7/TDI) PF7	I	For ISP

55	(ADC6/TDO) PF6	O	For ISP
56	(ADC5/TMS) PF5	O	For ISP
57	(ADC4/TCK) PF4	O	For ISP
58	(ADC3) PF3	I	MIC OUT(0~5V)
59	(ADC2) PF2	I	GND
60	(ADC1) PF1	I	Input Voltage (=wCK Output Voltage x 560 / 1560)
61	(ADC0) PF0	I	PSD Sensor - GP2Y0A21YK0F Signal (0~3.2V)

In order to program firmware, user should know I/O MAP, however, it is difficult to memorize all of the above. Therefore, check the necessary part when user program the certain firmware.

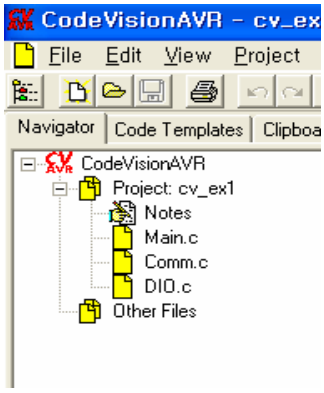
3.3 C Programming with Motion File

Download the C Programming with motion file data from RoboBuilder website (Support - Specialist Tip section).

In “cv_exam” folder, below files are located.

If CodeVisionAVR is installed, cv_ex1.prj is shown as the above. Click “cv_ex1.prj”.

File structure would be as the below.



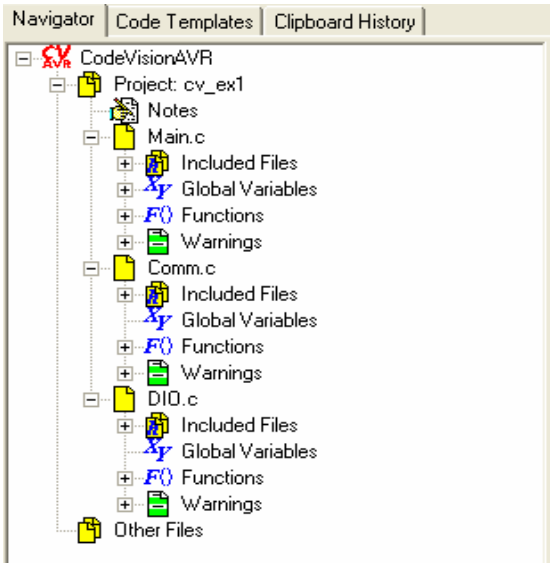
Example project has three files (Main.c / Comm.c / DIO.c).

Double-Click “Main.c”.

```
1  //===================================================================================================
2  //       RoboBuilder MainController Sample Program      1.0
3  //                                                        2008.04.14    Robobuilder co., ltd.
4  //       Tap Size = 4
5  //=============================================================================================
6  #include <mega128.h>
7
8  // Standard Input/Output functions
9  #include <stdio.h>
10 #include <delay.h>
11
12 #include "Macro.h"
13 #include "Main.h"
14 #include "Comm.h"
15 #include "Dio.h"
16 #include "math.h"
17
18 // Flag ------------------------------------------------------------------------------------------
19 bit     F_PLAYING;                  // Show the motion running
20 bit     F_DIRECT_C_EN;              // wCK direct control mode (1:Available, 0:Not available)
21
22 // Button Input-----------------------------------------------------------------------------------
23 WORD    gBtnCnt;                    // Button press process counter
24
25 // Time Measurement ------------------------------------------------------------------------------
26 WORD    gMSEC;
27 BYTE    gSEC;
28 BYTE    gMIN;
29 BYTE    gHOUR;
30
31 // UART Communication ----------------------------------------------------------------
32 char    gTx0Buf[TX0_BUF_SIZE];      // UART0 transmission buffer
33 BYTE    gTx0Cnt;                    // UART0 transmission idle byte number
34 BYTE    gRx0Cnt;                    // UART0 receiving byte number
35 BYTE    gTx0BufIdx;                 // UART0 transmission buffer index
36 char    gRx0Buf[RX0_BUF_SIZE];      // UART0 transmission buffer
```

It looks quite complicated code, however, you can just take a look at the outline now. Most “#define” word is included in “xxx.h” file.

Press [F9] to compile it, and click [OK].



It shows each include files, used variables and used functions. If you check out [Macro.h] file, there are many “#define” words.

```
//=====================================================================================
// DATA TYPE
//=====================================================================================
#define BYTE    unsigned char
#define WORD    unsigned int
#define DWORD   unsigned long
#define BYTEP   unsigned char*
#define WORDP   unsigned int*
#define SBYTE   signed char
#define SWORD   signed int
//=====================================================================================
// BIT SET
//=====================================================================================
#define         SET_BIT0(x)     (x |= 0x01)
#define         SET_BIT1(x)     (x |= 0x02)
#define         SET_BIT2(x)     (x |= 0x04)
#define         SET_BIT3(x)     (x |= 0x08)
#define         SET_BIT4(x)     (x |= 0x10)
#define         SET_BIT5(x)     (x |= 0x20)
#define         SET_BIT6(x)     (x |= 0x40)
#define         SET_BIT7(x)     (x |= 0x80)
//-------------------------------------------------------------------------------------
```

Dividing the code with several files are effective way to mange and understand it.

3.4 RBC LED Control – Understanding RBC I/O

Let's change the example project and control RBC Power, Run, Error, PF1 and PF2 LED button.

※ Requirements

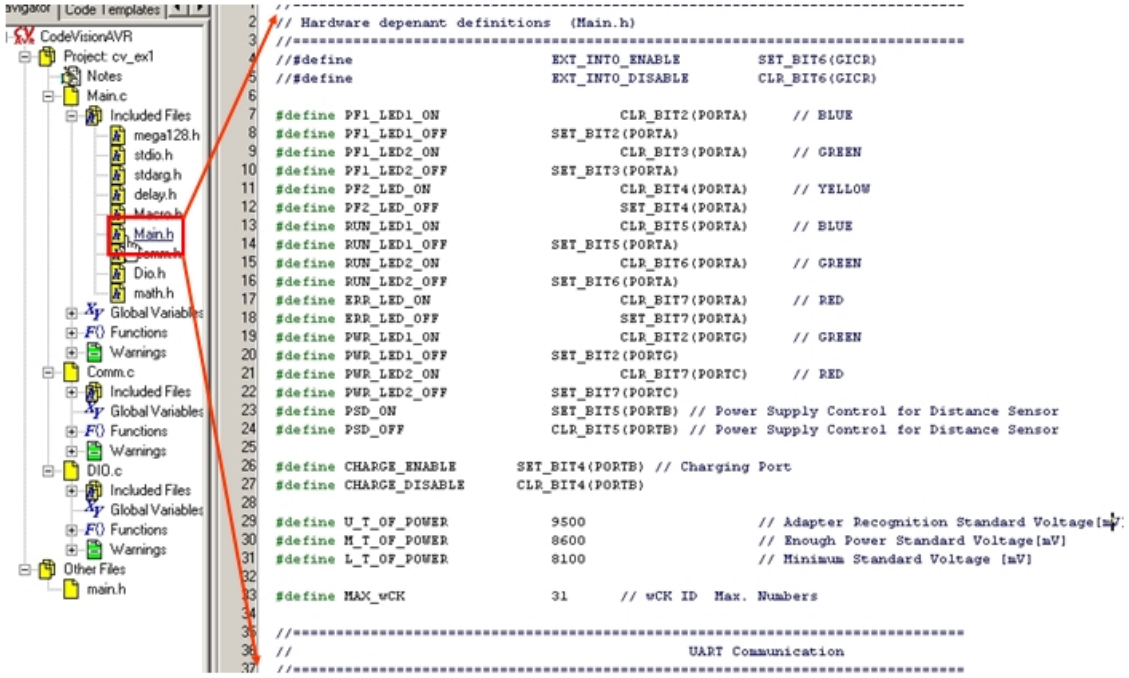
- RBC : 1EA
- RS-232 Serial Cable(PC cable)
- RBC Firmware Upgrade Tool
- CodevisionAVR Complier
- Published RBC Firmware

Let's check out RBC IO MAP first for controlling 5 LED in RBC.

42	(A15) PC7	O	Power LED (Red)
43	(ALE) PG2	O	Power LED (Green)
44	(AD7) PA7	O	Error LED (Red)
45	(AD6) PA6	O	Run LED (Green)
46	(AD5) PA5	O	Run LED (Blue)
47	(AD4) PA4	O	PF2 LED (Orange)
48	(AD3) PA3	O	PF1 LED (Red)
49	(AD2) PA2	O	PF1 LED (Blue)

It is "PIN No", "PIN Name", "IO Direction", "Description" in left order.

For example, Power LED(Red) is connected with ATmega128 No. 42 - PC7 Pin. Power LED (Red) is On / Off in accordance with Pin output. However, user can not check this only with IO MAP.
Therefore, let's find out in source code level.



If you check out “Main.h”, all lines are used “#define”. Power_LED (Red) is defined as the below.

#define PWR_LED2_ON CLR_BIT7(PORTC)
#define PWR_LED2_OFF SET_BIT7(PORTC)

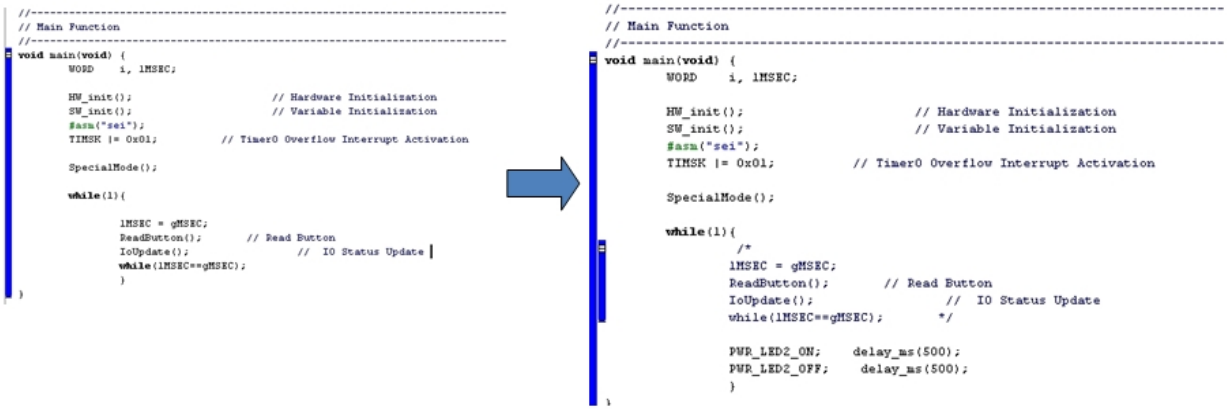
Therefore, you use “PWR_LED2_ON” in source code.
This meaning is clearing PORTC 7 Pin (Output “0”). In “Macro.h”, it is defined as

#define CLR_BIT7(x) (x &= 0x7F)

CLR_BIT7(PORTC) is defined as “PORTC &= 0x7F”.
CLR_BIT7(PORTC) means ‘0’, output 0XXX XXXX in PORTC. (X is ‘1’ or ‘0’, Don’t care). PORTC is hardware name in ATmega128. In order to understand “PORTC &= 0x7F”, please refer to the C programming with ATmega128 books.

In source code, use PWR_LED2_ON for Power_Led(Red) on, and use PWR_LED2_OFF for power-off.

Let’s revise the code for programming LED blinking in RBC.



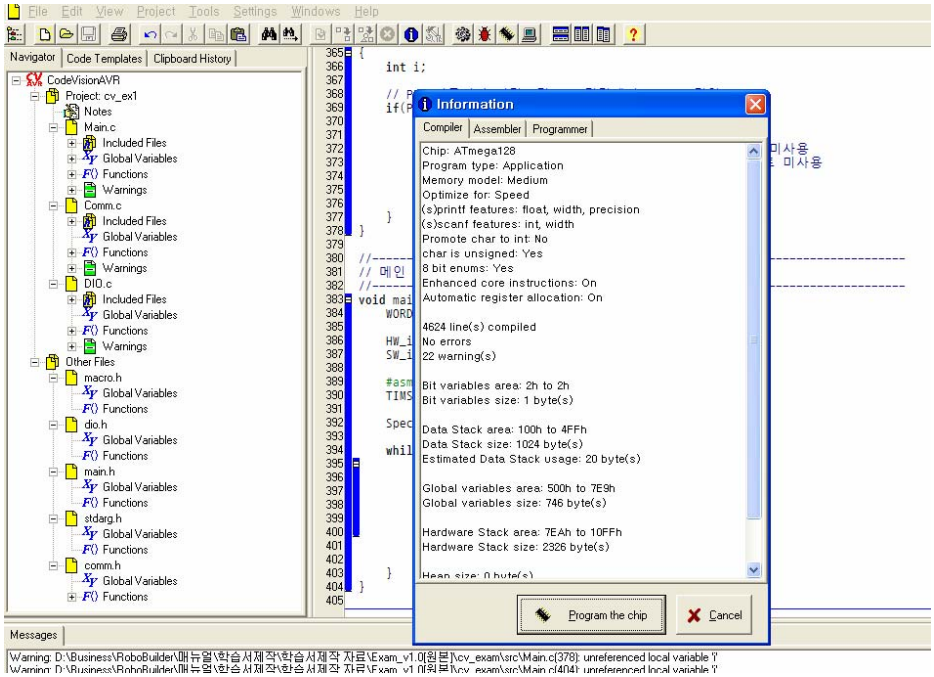
CPU starts from Main( ),

- WORD (unsigned int) I, lMSEC,
- HW_init(), SW_init(), SpecialMode()
- Power_LED(Red) On/Off loop in every 0.5 sec.

※ #asm(“sei”); TIMSK |= 0x01; is hardware dependency sentence.

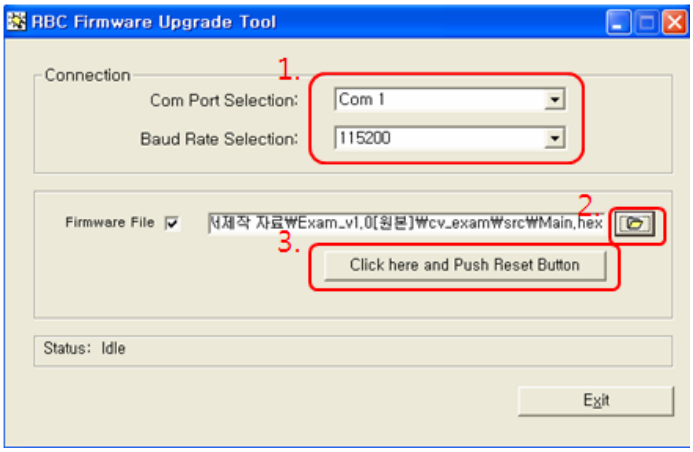
Revised part is “/*, */” and Power LED (Red) On/Off loop.

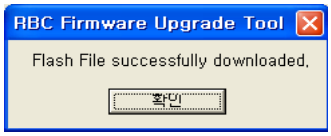
Press, “[SHIFT]+[F9]”.



In Information window, click “Cancel”, then run RBC Firmware Upgrade Tool.

Connect PC and RBC with PC cable, then power on RBC.



- Check COM PORT and Baud Rate,
- Select “Main.hex” file that is just made.
- Click [Click here and Push Reset Button].
- Use tweezers and press “Reset” button that is located between PF1 and PF2.
- Later, it shows message box. 

If succeeded, Power_LED (Red) is blinking in every 0.5 sec. In RBC, 8 LEDs are equipped. Try all other LED blinking in several ways.

※ If you would like to recover with normal published firmware, please download from RoboBuilder website (Support – Download section).
Then upgrade the firmware by using RBC Firmware Upgrade Tool.

3.5 Control wCK Position – 8 Bit Command Communication

Revise the example project file to control wCK position.

※ Requirements

- RBC : 1EA / wCK : 1EA
- RS-232 Serial Cable(PC Cable)
- RBC Firmware Upgrade Tool
- CodevisionAVR Compiler

wCK is smart actuator module through RBC UART communication. Therefore, user need to know the communication between wCK and RBC in order to control wCK.

Firstly, user should know the 'Packet-Communication' concept.
Packet-Communication is receiving and transmitting the pre-engaged information in order.
For example, let's say command '01' "Move wCK ID 0", and command '10' is "Stop wCK ID 1".
By doing this way, packet-communication would be very effective way to communicate each other.

Let's see how RBC and wCK communicates between them.
Download "wCK module protocol table file" from RoboBuilder website.
(Support – Tips for Specialist Section)

■ wCK protocol definition

Command		byte 1	byte 2		byte 3	byte 4
			7 6 5 4 3 2 1 0		7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
	Position Move	header	Torq	ID	target position	check sum
		0xFF		0~30	0~254	(note1)

Let's find out what this table means.

For example,
RBC commands "Move to 200 position with torque 1" to wCK ID 01.

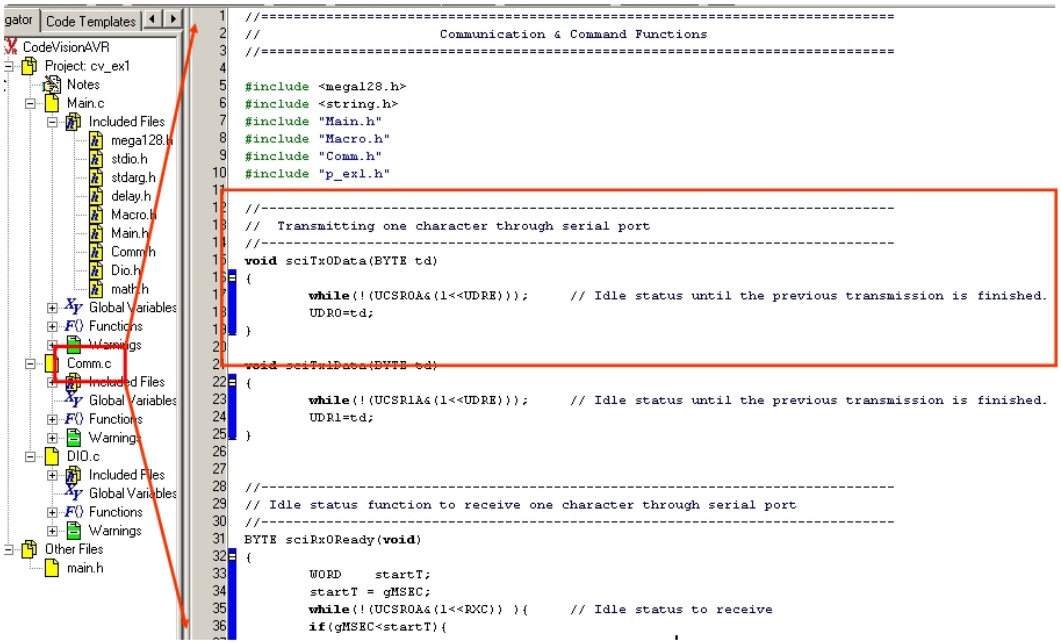
This can be re-written as follows;

0xFF(header) + 0x21(Torq(1)+ID(1)) + 0xC8(Target Position(200)) + 0x69(CheckSum)

We do NOT know what this means, but wCK ID 01 moves to "200" position with torque 1.

Firstly, RBC sends Header ‘0xFF’, then Torque, Target Position and CheckSum in order to wCK. Therefore, user should follow the communication information sequence.

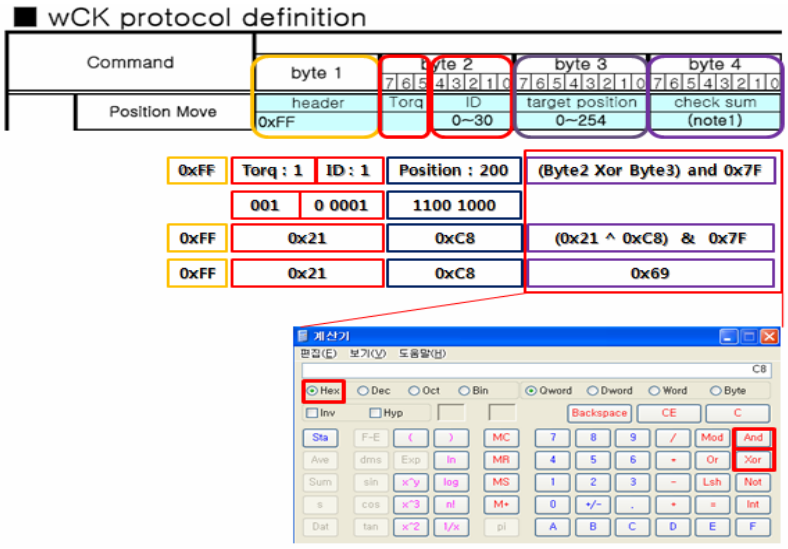
Let’s see the below firmware source to control wCK.



In “Comm.c” file, there is “void sciTx0Data(BYTE td)” function.
“void” means there is no return value. And “BYTE td” means input data to be processed.
Let’s understand this briefly that hardware send data if data is input for ‘td’ variable.

For example, if you want to send ‘200’ to wCK, just input “sciTx0Data(200)”.

To move wCK ID01 to ‘200’ position, it sends 4 hexadecimals.

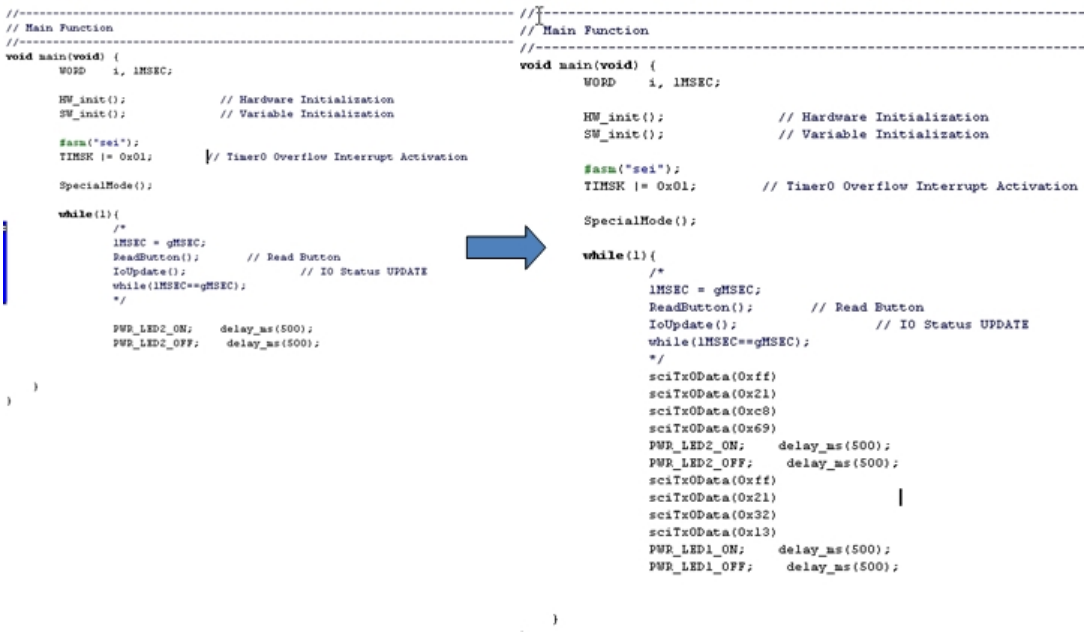


Sending data : [0xFF] + [0x21] + [0xC8] + [0x69]

Therefore, user uses “sciTx0Data” function and data “0xFF, 0x21, 0xC8, 0x69” for programming.

Let’s program that wCK moves ‘200’ position and moves ’50’ position.

Press [SHIFT] + [F9] to generate firmware file.

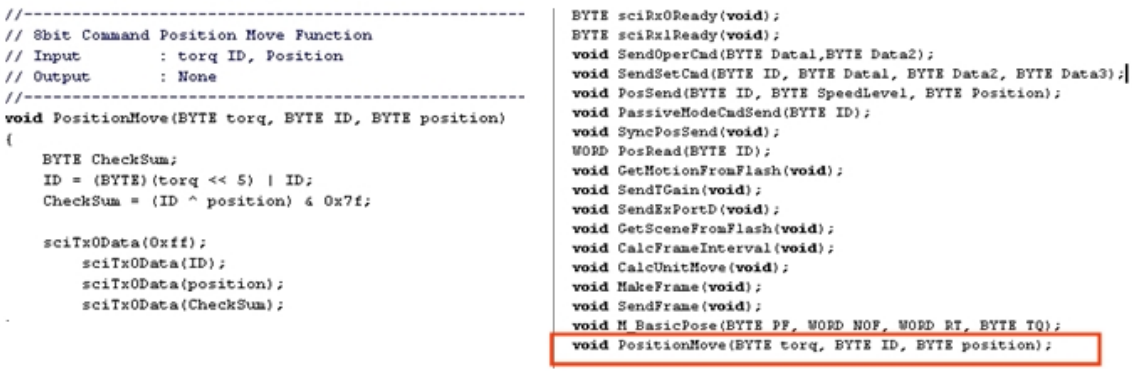


```
//-------------------------------------------------------------------------
// Main Function
//-------------------------------------------------------------------------
void main(void) {
        WORD    i, 1MSEC;

        HW_init();              // Hardware Initialization
        SW_init();              // Variable Initialization

        #asm("sei");
        TIMSK |= 0x01;          // Timer0 Overflow Interrupt Activation

        SpecialMode();

        while(1){
                /*
                1MSEC = gMSEC;
                ReadButton();       // Read Button
                IoUpdate();                 // IO Status UPDATE
                while(1MSEC==gMSEC);
                */

                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);


    }
}
```

```
//-------------------------------------------------------------------------
// Main Function
//-------------------------------------------------------------------------
void main(void) {
        WORD    i, 1MSEC;

        HW_init();              // Hardware Initialization
        SW_init();              // Variable Initialization

        #asm("sei");
        TIMSK |= 0x01;          // Timer0 Overflow Interrupt Activation

        SpecialMode();

        while(1){
                /*
                1MSEC = gMSEC;
                ReadButton();       // Read Button
                IoUpdate();                 // IO Status UPDATE
                while(1MSEC==gMSEC);
                */
                sciTx0Data(0xff)
                sciTx0Data(0x21)
                sciTx0Data(0xc8)
                sciTx0Data(0x69)
                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);
                sciTx0Data(0xff)
                sciTx0Data(0x21)
                sciTx0Data(0x32)
                sciTx0Data(0x13)
                PWR_LED1_ON;    delay_ms(500);
                PWR_LED1_OFF;    delay_ms(500);


}
```

RBC Power_LED will blink and wCK moves left and right side continuously.

However, it is very uncomfortable to calculate hexadecimal values every time it changes movement values. Therefore, put one function in “Comm.c” file and declare this function in “Comm.h” as shown in the below.

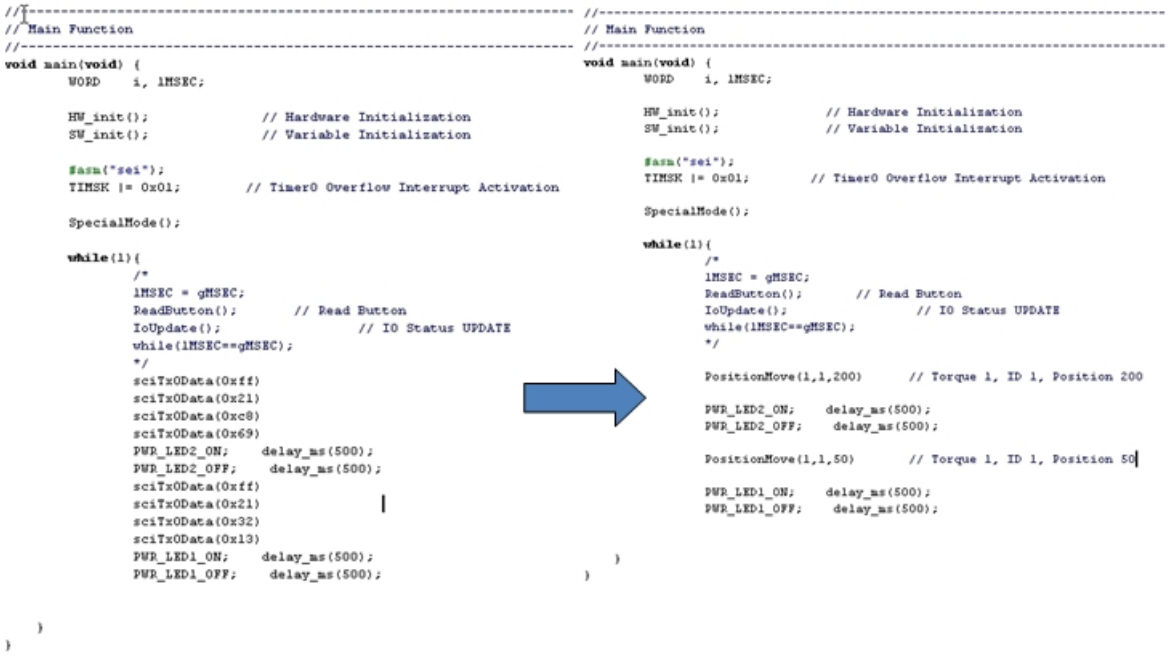


```
//-------------------------------------------------------
// 8bit Command Position Move Function
// Input        : torq ID, Position
// Output       : None
//-------------------------------------------------------
void PositionMove(BYTE torq, BYTE ID, BYTE position)
{
    BYTE CheckSum;
    ID = (BYTE)(torq << 5) | ID;
    CheckSum = (ID ^ position) & 0x7f;

    sciTx0Data(0xff);
        sciTx0Data(ID);
        sciTx0Data(position);
        sciTx0Data(CheckSum);
```

```
BYTE sciRx0Ready(void);
BYTE sciRx1Ready(void);
void SendOperCmd(BYTE Data1,BYTE Data2);
void SendSetCmd(BYTE ID, BYTE Data1, BYTE Data2, BYTE Data3);
void PosSend(BYTE ID, BYTE SpeedLevel, BYTE Position);
void PassiveModeCmdSend(BYTE ID);
void SyncPosSend(void);
WORD PosRead(BYTE ID);
void GetMotionFromFlash(void);
void SendTGain(void);
void SendExPortD(void);
void GetSceneFromFlash(void);
void CalcFrameInterval(void);
void CalcUnitMove(void);
void MakeFrame(void);
void SendFrame(void);
void M_BasicPose(BYTE PF, WORD NOF, WORD RT, BYTE TQ);
void PositionMove(BYTE torq, BYTE ID, BYTE position);
```

If you know C language operations <<, |, ^ and &, you can understand the above function.

Let’s revise the code in “Main()” function as the below.



```
//-----------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------
void main(void) {
        WORD    i, 1MSEC;

        HW_init();              // Hardware Initialization
        SW_init();              // Variable Initialization

        $asm("sei");
        TIMSK |= 0x01;        // Timer0 Overflow Interrupt Activation

        SpecialMode();

        while(1){
                /*
                1MSEC = gMSEC;
                ReadButton();        // Read Button
                IoUpdate();                 // IO Status UPDATE
                while(1MSEC==gMSEC);
                */
                sciTx0Data(0xff)
                sciTx0Data(0x21)
                sciTx0Data(0xc8)
                sciTx0Data(0x69)
                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);
                sciTx0Data(0xff)
                sciTx0Data(0x21)
                sciTx0Data(0x32)
                sciTx0Data(0x13)
                PWR_LED1_ON;    delay_ms(500);
                PWR_LED1_OFF;    delay_ms(500);


    }
}
```

```
//-----------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------
void main(void) {
        WORD    i, 1MSEC;

        HW_init();              // Hardware Initialization
        SW_init();              // Variable Initialization

        $asm("sei");
        TIMSK |= 0x01;        // Timer0 Overflow Interrupt Activation

        SpecialMode();

        while(1){
                /*
                1MSEC = gMSEC;
                ReadButton();        // Read Button
                IoUpdate();                 // IO Status UPDATE
                while(1MSEC==gMSEC);
                */

                PositionMove(1,1,200)       // Torque 1, ID 1, Position 200

                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);

                PositionMove(1,1,50)        // Torque 1, ID 1, Position 50

                PWR_LED1_ON;    delay_ms(500);
                PWR_LED1_OFF;    delay_ms(500);


    }
}
```

If you revised all source code, compile the code and make the “*.hex” file.

Download this firmware file to RBC as you use “RoboBuilder Firmware Upgrade Tool”.

The above-right side C code is much easier than left side C code to understand.

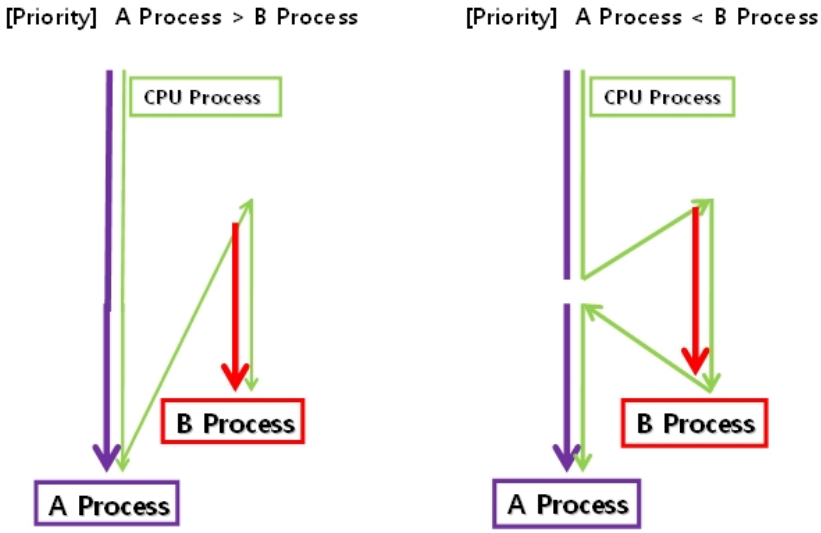
3.6 Control wCK LED

※ Requirements

- RBC : 1EA / wCK : 1EA
- RS-232 Serial Cable (PC cable)
- RBC Firmware Upgrade Tool
- Code VisionAVR Compiler

As wCK position control, let's control wCK I/O PORT (LED) through packet communication.
Actually, RBC and wCK use interrupt communication, therefore it increases CPU operation efficiency.

Then, what is "Interrupt"?



Atmega128 is a CPU that processes one instruction at one time. It could not process two tasks at the same time. But what would be happened if "B" task should be done when CPU is in the middle of processing the "A" task? It stops "A" task, then process "B" task first if "B" task is more important. This is called "Interrupt" method.

Depend on task importance, CPU possession is changed. Generally, main() and external functions are less priority than certain hardware function. RBC-ATmega 128 CPU can decide this priority.

Controlling wCK position is "Polling" method.
But if you use "Interrupt" method, it can response much faster from the environment change.

Let's see the "Polling" method code and "Interrupt" method code in the below.

```
void PositionMove(BYTE torq, BYTE ID, BYTE position)
{
    BYTE CheckSum;
    ID= (BYTE)(torq << 5) | ID;
    CheckSum = (ID ^ position) & 0x7f;

    sciTx0Data(0xff);
    sciTx0Data(ID);
    sciTx0Data(position);
    sciTx0Data(CheckSum);

}
```

```
void sciTx0Data(BYTE td)
{
    while(!(UCSR0A&(1<<UDRE)));
    UDR0=td;
}
```

1Byte Data Load => Call function manually

```
void IOwrite( BYTE ID, BYTE IOchannel)
{
    BYTE CheckSum;
    ID=(BYTE)(7<<5)|ID;
    IOchannel &= 0x03;
    CheckSum = (ID^100^IOchannel^IOchannel)&0x7f;

    gTx0Buf[gTx0Cnt]=HEADER;    gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=ID;        gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=100;       gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=IOchannel; gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=IOchannel; gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=CheckSum;  gTx0Cnt++;

    gTx0BufIdx++;
    sciTx0Data(gTx0Buf[gTx0BufIdx-1]);

}
```

```
interrupt [USART0_TXC] void usart0_tx_isr(void) {
    if(gTx0BufIdx<gTx0Cnt){
        while(!(UCSR0A&(1<<UDRE)));
        UDR0=gTx0Buf[gTx0BufIdx];
        gTx0BufIdx++;
    }
    else if(gTx0BufIdx==gTx0Cnt){
        gTx0BufIdx = 0;
        gTx0Cnt = 0;
    }
}
```

All data are loaded.
Call transmission interrupt function automatically.

It looks that c source code of Polling method is shorter and simple.

However, Interrupt method is faster since it uses Data Load method and Function calling process is more effective way. Transmission interrupt function is as the below.

```
interrupt [USART0_TXC] void usart0_tx_isr(void) {
  When 1 byte data transmission is completed,
  interrupt function is run automatically.
        if( gTx0BufIdx < gTx0Cnt ){
    Transmitted data number < Data number to be transmitted
            while( !(UCSR0A & (1<<UDRE)));
              Check whether the data transmission is completed.
            UDR0 = gTx0Buf[ gTx0BufIdx ];
              Data transmission
            gTx0BufIdx++;
              Transmitted data number + 1
        }

        else if(gTx0BufIdx==gTx0Cnt){
  Transmitted data number == Data number to be transmitted
          → Check data transmission completed
            gTx0BufIdx = 0;   Initialize transmitted data number
            gTx0Cnt = 0;      Initialize data number to be transmitted.
        }
}
```

Process from transmitting the first data to the second data, process time is different depends on automatic or manual. Process time is accumulated for this reason. So, CPU is used 100%, or sometime, 80% used. Handling RBC and wCK means that it makes the communications between RBC and wCK smoothly. Using interrupt method is more effective way for handling wCK.

For more detailed information about Interrupt method, please check out ATmega128 books.

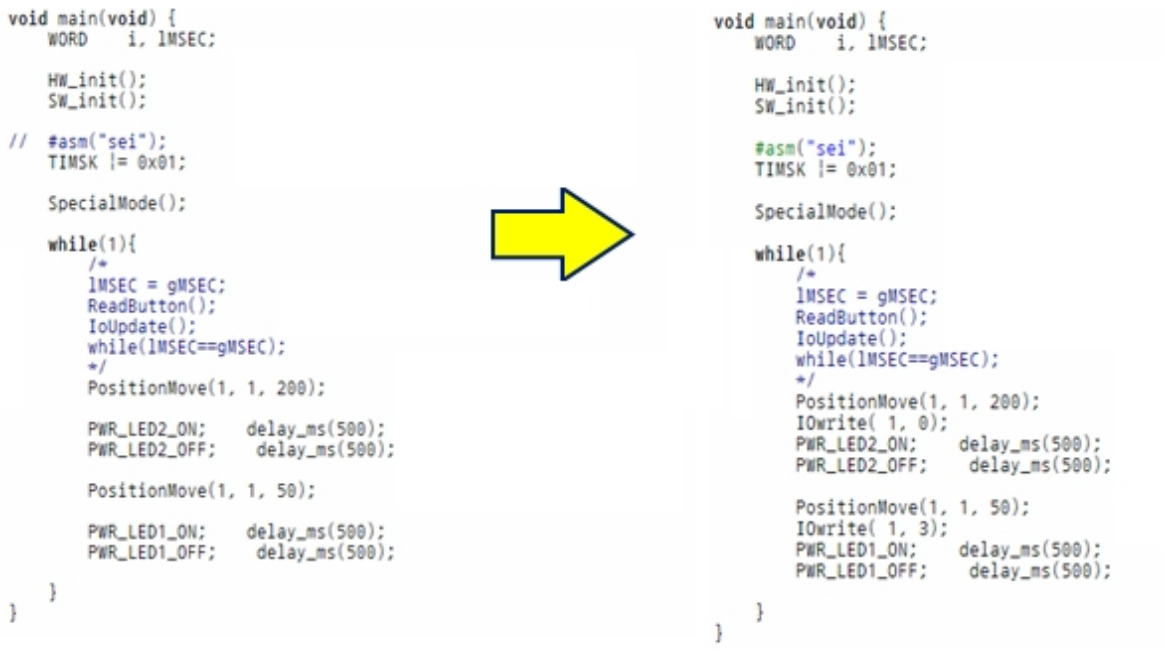
Let's control wCK-1108T, wCK-1111T module LED through I/O Write protocol.
In "Comm.c" file, write the function as the left side, declare the function as the right side in "Comm.h" file.

```
// Expansion command I/O write function
// Input : ID, IOchannel
// Output : None
void IOwrite( BYTE ID, BYTE IOchannel)
{
    BYTE CheckSum;
    ID=(BYTE)(7<<5)|ID;
    IOchannel &= 0x03;
    CheckSum = (ID^100^IOchannel^IOchannel)&0x7f;

    gTx0Buf[gTx0Cnt]=HEADER;    gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=ID;        gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=100;       gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=IOchannel; gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=IOchannel; gTx0Cnt++;
    gTx0Buf[gTx0Cnt]=CheckSum;  gTx0Cnt++;

    gTx0BufIdx++;
    sciTx0Data(gTx0Buf[gTx0BufIdx-1]);

}
```

```
void sciTx0Data(BYTE td);
void sciTx1Data(BYTE td);
BYTE sciRx0Ready(void);
BYTE sciRx1Ready(void);
void SendOperCmd(BYTE Data1,BYTE Data2);
void SendSetCmd(BYTE ID, BYTE Data1, BYTE Data2, BYTE Data3);
void PosSend(BYTE ID, BYTE SpeedLevel, BYTE Position);
void PassiveModeCmdSend(BYTE ID);
void SyncPosSend(void);
WORD PosRead(BYTE ID);
void GetMotionFromFlash(void);
void SendTGain(void);
void SendExPortD(void);
void GetSceneFromFlash(void);
void CalcFrameInterval(void);
void CalcUnitMove(void);
void MakeFrame(void);
void SendFrame(void);
void M_BasicPose(BYTE PF, WORD NOF, WORD RT, BYTE TQ);
void PositionMove(BYTE torq, BYTE ID, BYTE position);
void IOwrite( BYTE ID, BYTE IOchannel);
```

Let's analyze `IOwrite(ID, IOchannel)` function.

```
void IOwrite( BYTE ID, BYTE IOchannel)
{
        BYTE CheckSum; unsigned char 8bit checksum
        ID = (BYTE)(7<<5)|ID; Write wCK ID on sub 5 bit to control
         Shift 5 bit left side
         Overwrite on ID variable.
        IOchannel &= 0x03;  IOchannel & 0000 0011 = 0000 00XX
                            // Upper 6 bits are don't care, Activate sub 2 bits
        CheckSum = (ID^100^IOchannel^IOchannel)&0x7f;
        CheckSum = ( ID  Xor  100  Xor  IOchannel  Xor  IOchannel )  And  0x7F
        gTx0Buf[gTx0Cnt]=HEADER;                    gTx0Cnt++;
        gTxBuf[0] = 0xFF
        gTx0Buf[gTx0Cnt]=ID;                gTx0Cnt++;
        gTx0Buf[gTx0Cnt]=100;               gTx0Cnt++;
                          IO Write Command Mode 100
        gTx0Buf[gTx0Cnt]=IOchannel;         gTx0Cnt++;
        gTx0Buf[gTx0Cnt]=IOchannel;         gTx0Cnt++;
        gTx0Buf[gTx0Cnt]=CheckSum;          gTx0Cnt++;

        gTx0BufIdx++;
        sciTx0Data(gTx0Buf[gTx0BufIdx-1]);
       gTx0Buf [ 0 ]  Transmission Starts
}
```



```
void main(void) {
    WORD    i, lMSEC;

    HW_init();
    SW_init();

//  #asm("sei");
    TIMSK |= 0x01;

    SpecialMode();

    while(1){
        /*
        lMSEC = gMSEC;
        ReadButton();
        IoUpdate();
        while(lMSEC==gMSEC);
        */
        PositionMove(1, 1, 200);

        PWR_LED2_ON;    delay_ms(500);
        PWR_LED2_OFF;    delay_ms(500);

        PositionMove(1, 1, 50);

        PWR_LED1_ON;    delay_ms(500);
        PWR_LED1_OFF;    delay_ms(500);

    }
}
```

```
void main(void) {
    WORD    i, lMSEC;

    HW_init();
    SW_init();

    #asm("sei");
    TIMSK |= 0x01;

    SpecialMode();

    while(1){
        /*
        lMSEC = gMSEC;
        ReadButton();
        IoUpdate();
        while(lMSEC==gMSEC);
        */
        PositionMove(1, 1, 200);
        IOwrite( 1, 0);
        PWR_LED2_ON;    delay_ms(500);
        PWR_LED2_OFF;    delay_ms(500);

        PositionMove(1, 1, 50);
        IOwrite( 1, 3);
        PWR_LED1_ON;    delay_ms(500);
        PWR_LED1_OFF;    delay_ms(500);

    }
}
```

Please make “*.hex” file and download to RBC.

wCK will move from 200 position to 50 position as wCK internal LED is blinking.

3.7 Configure wCK Parameters - Configure Command and Read Data

※ Requirements

- RBC : 1EA / wCK : 1EA
- RS-232 Serial Cable(PC Cable)
- RBC Firmware Upgrade Tool
- CodevisionAVR Compiler

If wCK ID has been changed, existed programs wCK position control and wCK LED control does NOT work. wCK ID information is included in packet.
Therefore, it does not work if wCK ID is not matched.

Let's check the communication protocol first. ID set protocol is as the below.

Command	header	mode	ID	mode	Byte4		CheckSum
ID Set	0XFF	7	0~30	12	new ID 0~254	= byte4	

6 byte data is transmitted totally.

[header] + [mode | ID] + [mode] + [newID] + [Byte5 (newID)] + [CheckSum]

First [mode] means Configure Command, second [mode] means ID Set.

In "Comm.c" file, below source code is included.

```
//------------------------------------------------------------------------------
// wCK Parameter Configuration
// Input        : Data1, Data2, Data3, Data4
// Output       : None
//------------------------------------------------------------------------------
void SendSetCmd(BYTE ID, BYTE Data1, BYTE Data2, BYTE Data3)
{
        BYTE CheckSum;
        ID=(BYTE)(7<<5)|ID;
        CheckSum = (ID^Data1^Data2^Data3)&0x7f;

        gTx0Buf[gTx0Cnt]=HEADER;
        gTx0Cnt++;                      // Increase byte number to be transmitted.

        gTx0Buf[gTx0Cnt]=ID;
        gTx0Cnt++;                      // Increase byte number to be transmitted.

        gTx0Buf[gTx0Cnt]=Data1;
        gTx0Cnt++;                      // Increase byte number to be transmitted.

        gTx0Buf[gTx0Cnt]=Data2;
        gTx0Cnt++;                      // Increase byte number to be transmitted.

        gTx0Buf[gTx0Cnt]=Data3;
        gTx0Cnt++;                      // Increase byte number to be transmitted.

        gTx0Buf[gTx0Cnt]=CheckSum;
        gTx0Cnt++;                      // Increase byte number to be transmitted.
}
```

Data to be transmitted are 6 Bytes. 2 Bytes are Header and CheckSum, and in “SendSetCmd” function, Configure Command (mode(7)) is shifted left side automatically. Therefore, you should decide “ID, Data1, Data2, Data3”.

If you check out communication protocols again, “Mode(12)” value should be input in Data1.
In Data2 and Data3, you can input ID No. to be changed.

SendSetCmd (wCK ID to be controlled, 12, new ID, new ID).

For example, if you want to change the wCK ID with “10”, it should be like the below.

SendSetCmd (1, 12, 10, 10)

Let's include ID Set command in the control wCK LED source code. Last wCK ID was "1".
Change the source code and check out the RBC and wCK movement.

```
//-----------------------------------------------------------------------------
//  Main Function
//-----------------------------------------------------------------------------
void main(void) {
        WORD    i, lMSEC;

        HW_init();                      // Initialize Hardware
        SW_init();                      // Initialize Variables

        #asm("sei");
        TIMSK |= 0x01;          // Timer0 Overflow Interrupt activate

        SpecialMode();

        SendSetCmd(1, 12, 10, 10);      // Change wCK ID from '1' to '10'

        while(1){
                /*
                lMSEC = gMSEC;
                ReadButton();       //  Read Button
                IoUpdate();                 // IO Status UPDATE
                while(lMSEC==gMSEC);
                */
                PositionMove(1,1,200);
                IOwrite(1,0);
                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);

                PositionMove(1,1,50);
                IOwrite(1,3);
                PWR_LED1_ON;    delay_ms(500);
                PWR_LED1_OFF;    delay_ms(500);
    }
}
```

Does wCK LED is blinking and moving left and right side?
It blinks only wCK LED, and user will know the reason.

Let's look at the code again. Do you understand "SendSetCmd(1, 12, 10, 10);" function?
Let's change with "SendSetCmd(1, ID_SET, 10, 10)." In previous source code, it is difficult to know the meaning of "12" and "10") before you refer the communication protocol table.

Add "#define ID_SET 12" in "Main.h" file.

```
/*---------------------------------------------------------------------
/  Main Function
/---------------------------------------------------------------------*/
void main(void) {
        WORD    i, lMSEC;

        HW_init();                      // Initialize Hardware
        SW_init();                      // Initialize Variables

        #asm("sei");
        TIMSK |= 0x01;          // Timer0 Overflow Interrupt activate

        SpecialMode();

        SendSetCmd(1, ID_SET, 10, 10);    // Change wCK ID from '1' to '10'

        while(1){
                /*
                lMSEC = gMSEC;
                ReadButton();       //  Read Button
                IoUpdate();                 // IO Status UPDATE
                while(lMSEC==gMSEC);
                */
                PositionMove(1,1,200);
                IOwrite(1,0);
                PWR_LED2_ON;    delay_ms(500);
                PWR_LED2_OFF;    delay_ms(500);

                PositionMove(1,1,50);
                IOwrite(1,3);
                PWR_LED1_ON;    delay_ms(500);
                PWR_LED1_OFF;    delay_ms(500);
    }
```

```
#define RUN_LED2_OFF            SET_BIT6(PORTA)
#define ERR_LED_ON                      CLR_BIT7(PORTA)     // RED
#define ERR_LED_OFF                     SET_BIT7(PORTA)
#define PWR_LED1_ON                     CLR_BIT2(PORTG)     // GREE
#define PWR_LED1_OFF            SET_BIT2(PORTG)
#define PWR_LED2_ON                     CLR_BIT7(PORTC)     // RED
#define PWR_LED2_OFF            SET_BIT7(PORTC)
#define PSD_ON                  SET_BIT5(PORTB)
#define PSD_OFF                     CLR_BIT5(PORTB)

#define CHARGE_ENABLE       SET_BIT4(PORTB)
#define CHARGE_DISABLE      CLR_BIT4(PORTB)

#define U_T_OF_POWER            9500
#define M_T_OF_POWER            8600
#define L_T_OF_POWER            8100

#define MAX_wCK                 31

//=====================================================================
//                                                  UART Communication
//=====================================================================
#define TX0_BUF_SIZE    186                 // UART0
#define RX0_BUF_SIZE    0                   // UART0
#define RX1_BUF_SIZE    20                  // UART1

//=====================================================================
//                                                  Send Command
//=====================================================================
#define ID_SET          12
```

In order to change the wCK ID with “1”, revise the code as the below.

wCK ID is changed from “10” to “1”.

```
SendSetCmd( 10,    ID_SET,    1,    1 );
```

As shown in the above, parameters can be changed from source coding by user.

3.8 C Programming with Motion File

Change the motion file as a header file (*.h) format.

This is not for using RoboBuilder standard firmware, to use user-defined firmware to run motion file directly.

※ Requirements

- Robobuilder Robot : 1set
- RS-232 Serial Cable(PC Cable)
- RBC Firmware Upgrade Tool
- CodevisionAVR Compiler

Published RBC firmware data (Please see the Example project : 3-8 Motion Program)

1) Example Motion File

① Project File : p_ex1.prj

② Motion File : m_ex1.rbm

2) Example C source file (CodeVisionAVR version 1.24.8d)

① Project File : cv_ex1.prj

② Unit File : main.c, comm.c, dio.c

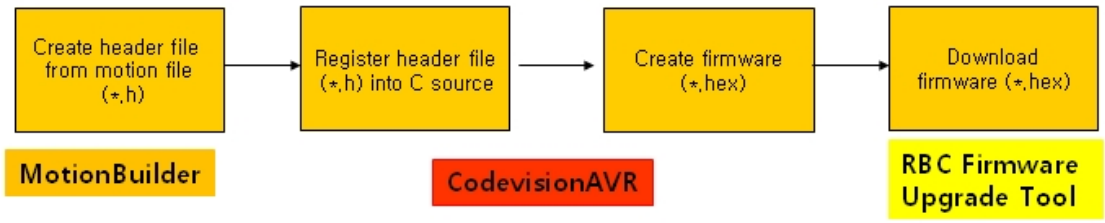
③ Header File : main.h, comm.h, dio.h, macro.h, m_ex1.h

NOTE)

MotionBuilder version : 1.10 beta or higher version.

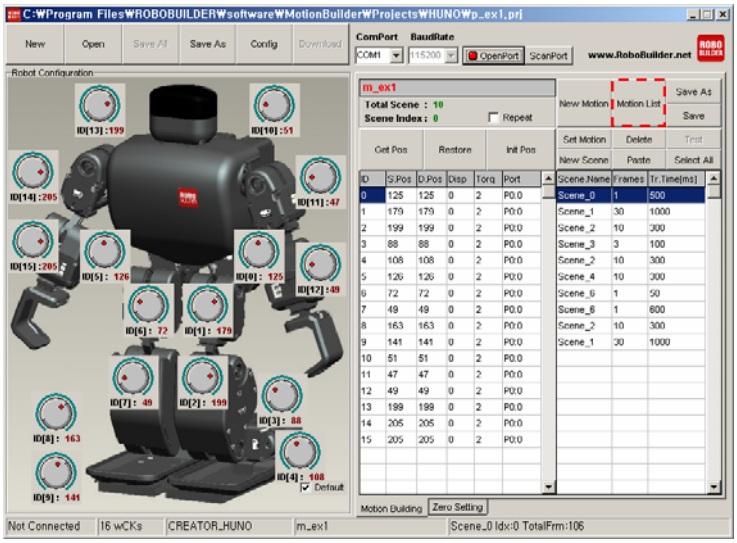
CodeVisionAVR : 1.24.8d

Job Procedure

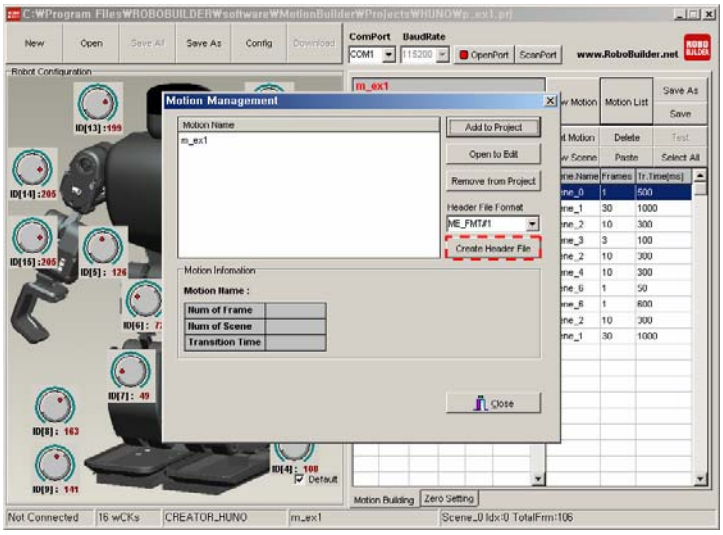


Change into Header File (*.h)

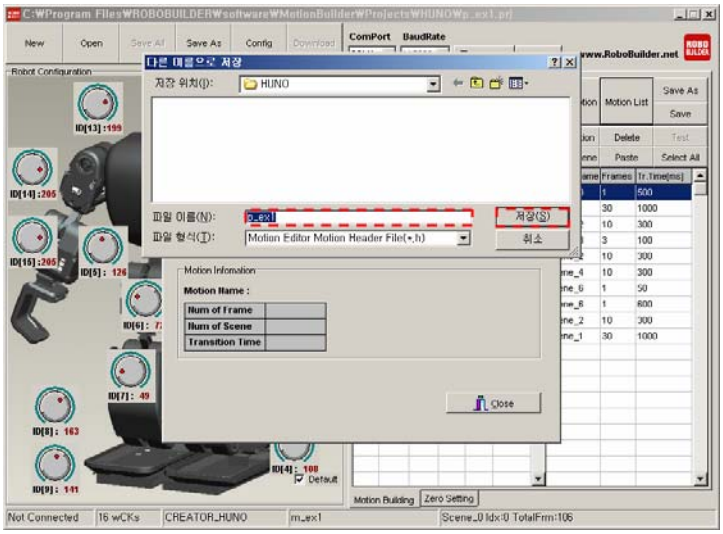
1. Run MotionBuilder.
2. Click “Open” button, then load “p_ex1.prj” in motion_exam folder.
3. Click “Motion List” button.



4. Click “Creator Header File” after you select “ME_FMT#1” in Header File Format.



5. Input Header file name (*.h), then, click “Save” button. In the below example, file name is “p_ex1.h”.



6. It is asked whether you see the generated header file. If you click “Yes”, it shows header file contents.
7. Header file is generated successfully.

Registering header file in C source code

1. Move 'p_ex1.h' file into “cv_exam\src” folder.
2. Load ‘cv_ex1.prj’ in CodeVision.
3. Add below code in “comm.c” file.

```
#include "p_ex1.h"
```

4. Match the below array name with motion name. (Use big letter.)
   For instance, motion name is “M_EX1”,

```
gpT_Table           = M_EX1_Torque;
gpE_Table           = M_EX1_Port;
gpPg_Table          = M_EX1_RuntimePGain;
gpDg_Table          = M_EX1_RuntimeDGain;
gpIg_Table          = M_EX1_RuntimeIGain;
gpFN_Table          = M_EX1_Frames;
gpRT_Table          = M_EX1_TrTime;
gpPos_Table         = M_EX1_Position;
Motion.NumOfScene   = M_EX1_NUM_OF_SCENES;
Motion.NumOfwCK     = M_EX1_NUM_OF_WCKS;
```

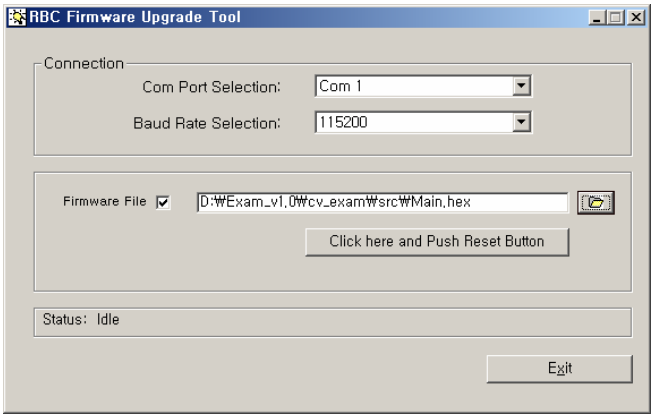
5. Header file registering finished.

Create firmware file (*.hex)

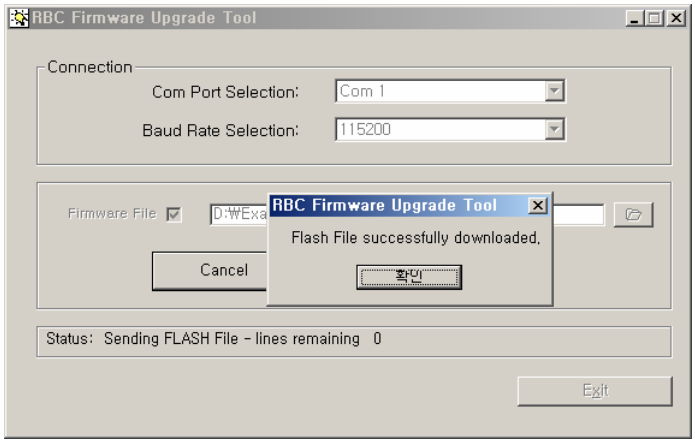
1. Click Project-Make menu in CodeVisionAVR, or press “Shift” + “F9” key.
2. firmware file (*.hex) will be generated.

Download firmware and Play RoboBuilder

1. Connect Power Adapter and PC Cable, then Power on RBC.
2. Run RBC Firmware Upgrade Tool and set proper COM Port.



3. Click 'Click here and Push Reset Button'.
4. Press Reset button (It is located between PF1 and PF2 button), then it starts upgrade.
5. If finished, it shows 'Flash File successfully downloaded.' message box.



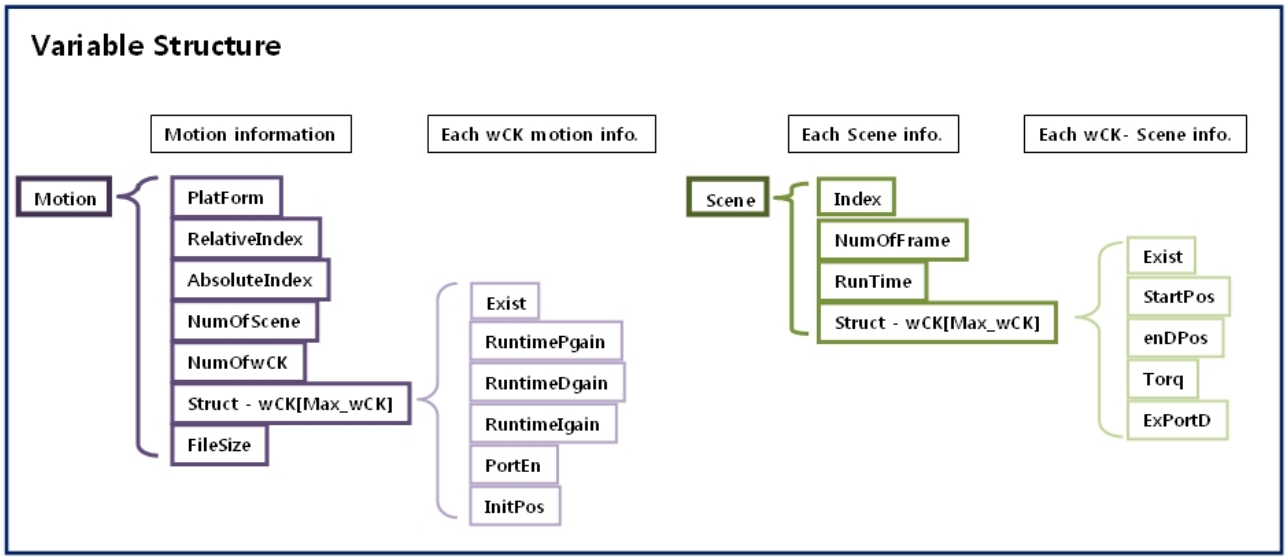
6. Disconnect RoboBuilder from PC, then press PF1 button to check out.

Generated motion header file has wCK torq, target position, frame no., etc. This information is saved in C program header file as an array. C program use these data to control wCK.

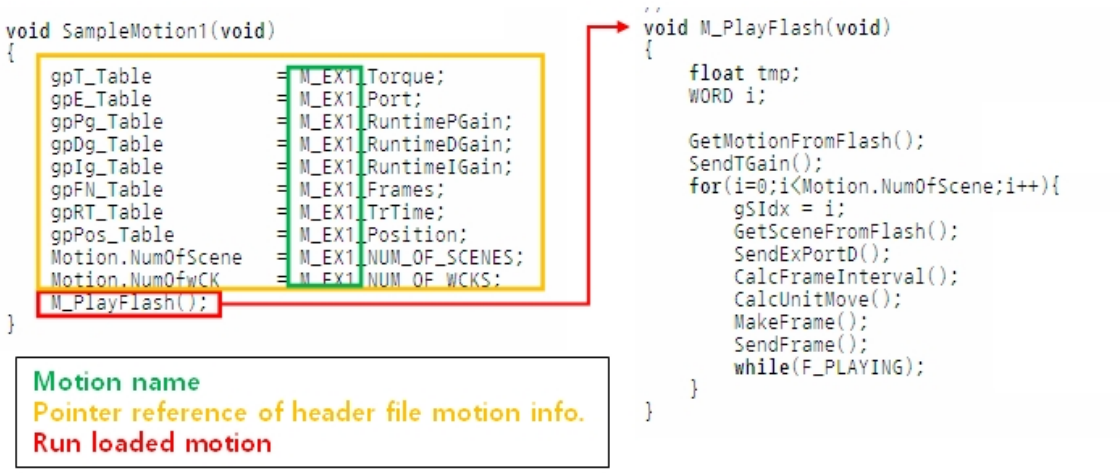
Created p_ex1.h File Structure

```
unsigned char  flash  M_EX1_Port [10][16] = { ... ... ...};

wCK_IDs[16]= { ... ... ...};                  // Used wCK ID
MotionZeroPos[16]= { ... ... ...};            // Each wCK Zero Position value
#define M_EX1_NUM_OF_SCENES 10                // Motion Scene Nos.
#define M_EX1_NUM_OF_WCKS 16                  // wCK Nos.
M_EX1_RuntimePGain[16]= { ... ... ...};       // Each wCK Runtime P gain
M_EX1_RuntimeDGain[16]= { ... ... ...};       // Each wCK Runtime P gain
M_EX1_RuntimeIGain[16]= { ... ... ...};       // Each wCK Runtime P gain
M_EX1_Frames[10]= { ... ... ...};             // Each Scene Frame Nos.
M_EX1_TrTime[10]= { ... ... ...};             // Each Scene Run Time
M_EX1_Position[11][16]= { ... ... ...};       // wCK Each Scene Target Position
M_EX1_Torque[10][16]= { ... ... ...};         // wCK Eacg Scene Torque
M_EX1_Port[10][16]= { ... ... ...};           // wCK Each Scene IO Port Control

                                              ※ 'M_EX1' is Motion Name
```

Generated header file data are assigned as the structure members. Data is divided Motion data and Scene data. Based on these two data, frame data is needed to connect each Scene data.
And it send the “Position move” instruction to wCK.



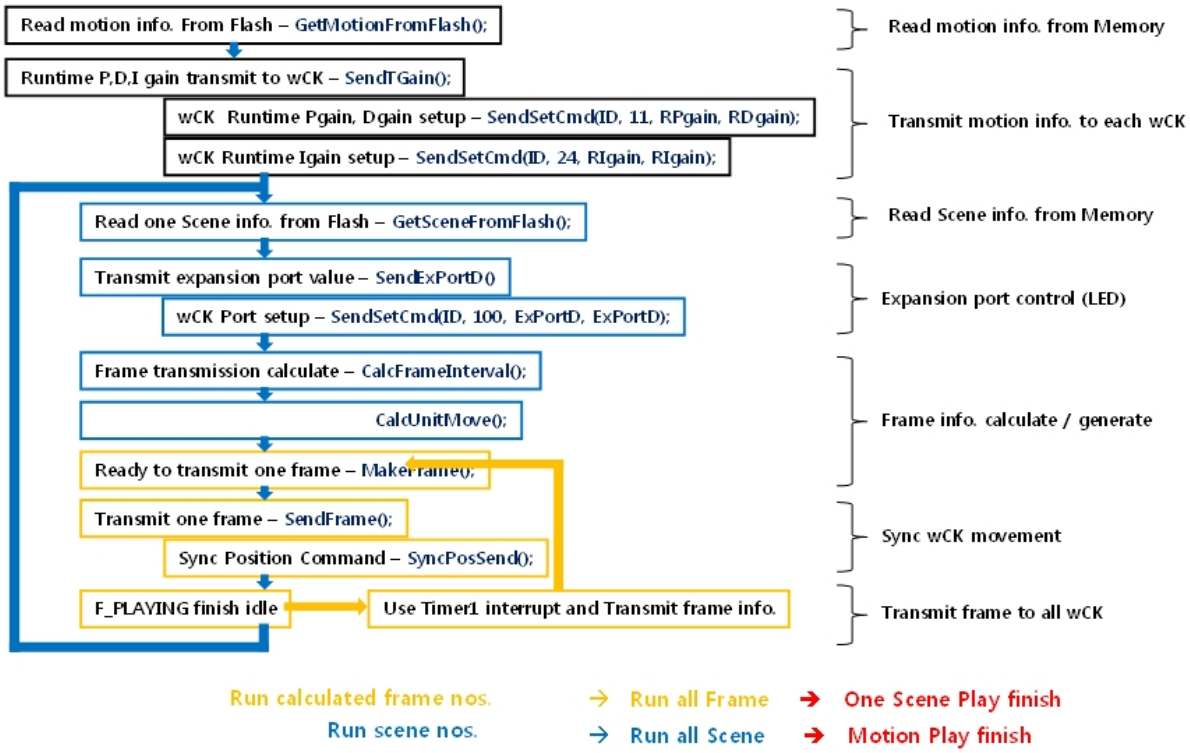
In example source code, SampleMotion() function should be included in order to run motion header file. SampleMotion() function load various data and run motion file. Therefore, you can just change the motion file name variable properly if you would like to run other motion file.



If you look at the above source code, “M_PlayFlash()” function gets motion information and make and send the frames to each wCK.

Let’s check out the below Flow_Chart how M_PlayFlash() function makes motion.

M_PlayFlash() function Flow-chart



M_PlayFlash function is running until motion is finished.
GetMotionFromFlash(), GetSceneFromFlash(), CalcFrameInterval(), SyncPosSend() functions are included in “Comm.c”.

If you don’t use wCK LED function, you can remove “SendExPortD()” function.
Advanced algorithm walking or sensor function can be included in this source code.

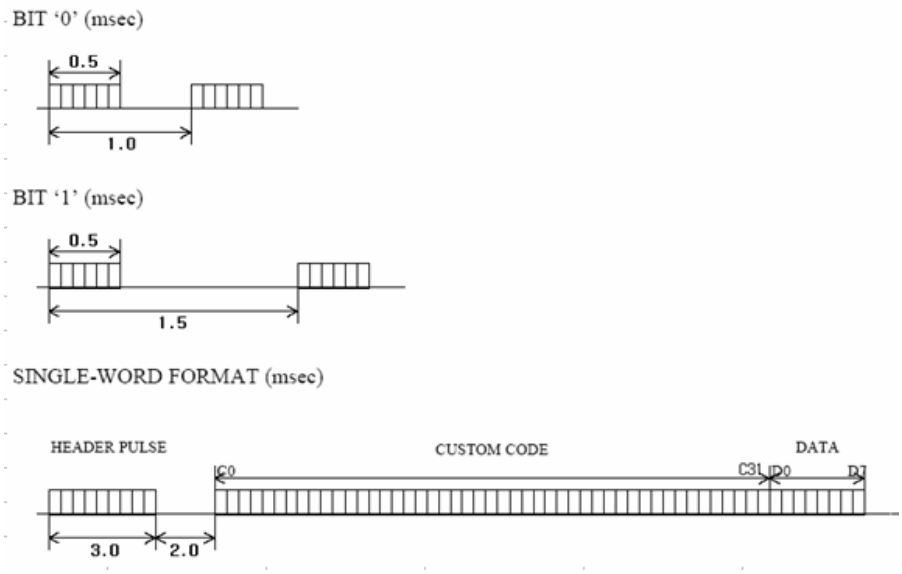
So you can build much smarter robot as you do C programming.

3.9 IR Remote Controller with C Programming

※ Requirements

- Robobuilder Robot : 1set
- IR remote controller : 1EA
- RS-232 Serial Cable(PC Cable)
- RBC Firmware Upgrade Tool
- CodevisionAVR Compiler

If you look at the below pictures, it shows IR remote controller pulse signal that divide ‘0’ and ‘1’. BIT ‘0’ and BIT ‘1’ is different in terms of signal length. BIT ‘0’ signal is 1 ms, while BIT ‘1’ signal length is 1.5ms. If you look at the below SINGLE-WORD FORMAT, IR remote controller sends 5ms Header Pulse at first time, then it sends transmission code.



Header Pulse is ready signal to send “CUSTOM CODE”. Real transmission code is “Custom Code”, and “Data Code”. Therefore, header pulse and data should be distinguished. The best way is to use signal length. If 1~1.5ms signal is in, it is data. If 5 ms signal is in, it is header pulse. In order to know the signal length, check the signal rise and signal fall. This is hardware dependence function. This code is included in published program source. Let’s check out the below source code to understand clearly.

```
//-----------------------------------------------------------------------------
// Remote Controller IR Receiver Interrupt
//-----------------------------------------------------------------------------
interrupt [EXT_INT6] void ext_int6_isr(void) {
        BYTE width;                     //  Signal Length
        WORD i;                         //  Temporary Variable

        width = TCNT2;                  //  Save Signal Length
        TCNT2 = 0;                      //  Initialize Register (Measure signal length)

        if(gIrBitIndex == 0xFF){        //  When signal input is idle status
                // Check whether input signal is 5ms length (header pulse)
                if((width >= IR_HEADER_LT) && (width <= IR_HEADER_UT)){
                        F_IR_RECEIVED = 0;          // Receive finish flag inactivate ➔ Receiving idle status
                        gIrBitIndex = 0;            // Initialize data code number variable
                        for(i = 0; i < IR_BUFFER_SIZE; i++)     // Initialize data buffer
                                gIrBuf[i] = 0;
                }
        }
        else{   // When data code is in the middle of receiving
        // Check whether the received data code '0' is 1ms. If '0', receiving code numbers are increased.
                if((width >= IR_LOW_BIT_LT)&&(width <= IR_LOW_BIT_UT))  gIrBitIndex++;

                // Check whether the received data code '1' length is 1.5ms
                else if((width >= IR_HIGH_BIT_LT)&&(width <= IR_HIGH_BIT_UT)){

                        // Input received '1' bit position in order
                if(gIrBitIndex != 0)    gIrBuf[(BYTE)(gIrBitIndex/8)] |= 0x01<<(gIrBitIndex%8);
                        else            gIrBuf[0] = 0x01;
                        gIrBitIndex++;                  // Increase the received data code number
                }
                // When the received code is NOT '0', or '1', either.
                else gIrBitIndex = 0xFF;

                if(gIrBitIndex == (IR_BUFFER_SIZE * 8)){        // If input code is filled in buffer
                        F_IR_RECEIVED = 1;              // Receive finish flag setup => Receiving finish
                        gIrBitIndex = 0xFF;             // Input idle status setup
                }
        }
}
```

As shown in the above, IR remote controller has own signal status and method. Therefore, you need to check IR remote controller method in advance. Received data has 4 Byte information as the below.

Byte1 + Byte2 + Byte3(remote controller own custom code) + Byte4(data code)

This received 2 data and information are forwarded through "ProcIR()" function.

```
//-------------------------------------------------------------------------------
// Process IR receiving
//-------------------------------------------------------------------------------
void ProcIr(void)
{
    WORD      i;                          // Temporary variable
    if(F_DOWNLOAD) return;                // When program download, IR receiving is NOT available.
    If 'C' button, '#' button, NON-Standard platform, IR receiving is NOT available
    if(F_FIRST_M && gIrBuf[3]!=BTN_C && gIrBuf[3]!=BTN_SHARP_A && F_PF!=PF2) return;
    if(F_IR_RECEIVED){                    // IR receiving flag activated ➔ 4 Byte received
        EIMSK &= 0xBF;                    // No   IR receiving
        F_IR_RECEIVED = 0;                // IR receiving flat inactivate

       // Check whether remote controller is registered in RBC ➔ Custom code check
        if((gIrBuf[0]==eRCodeH[0] && gIrBuf[1]==eRCodeM[0] && gIrBuf[2]==eRCodeL[0])
         ||(gIrBuf[0]==eRCodeH[1] && gIrBuf[1]==eRCodeM[1] && gIrBuf[2]==eRCodeL[1])
         ||(gIrBuf[0]==eRCodeH[2] && gIrBuf[1]==eRCodeM[2] && gIrBuf[2]==eRCodeL[2])
         ||(gIrBuf[0]==eRCodeH[3] && gIrBuf[1]==eRCodeM[3] && gIrBuf[2]==eRCodeL[3])
         ||(gIrBuf[0]==eRCodeH[4] && gIrBuf[1]==eRCodeM[4] && gIrBuf[2]==eRCodeL[4])){
            switch(gIrBuf[3]){                  // Check received data code
                case BTN_A:                     // When received data code is BTN_A.
                    M_Play(BTN_A);              // Run BTN_A motion
                    break;
                case BTN_B:                     // When received data code is BTN_B.
                    M_Play(BTN_B);              // Run BTN_B motion
                    break;
                ...
            }
        }
        for(i=0;i<IR_BUFFER_SIZE;i++) gIrBuf[i]=0;          // IR receiving initialize
        EIMSK |= 0x40;                                      // Permit IR receiving
    }
}
```

※ BTN_A, BTN_B, BTN_C command values are defined in "Main.h".

If you make own function and source code instead of "M_Play()" function, you can play other program with IR remote controller.

3.10 Humanoid Robot Maze Escape

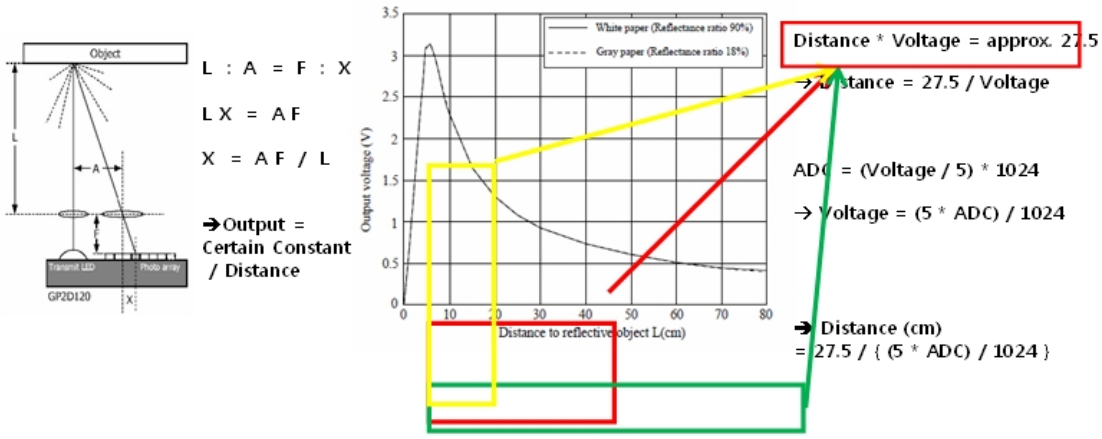
Let's make the maze escape program to use HUNO PSD sensor.

※ Requirements

- RoboBuilder (Huno)
- RoboBuilder IR Remote Controller
- RS-232 Serial Cable(PC Cable)
- RBC Firmware Upgrade Tool

Let's check out PSD sensor principle and usage.

PSD Sensor Principle **PSD Sensor Usage**



PSD is the sensor that reflected light angle is changed in accordance with distance. Let's look at the above left figure. If the object become more distant, 'L' value is increased while 'A' value is decreased. And 'X' value is also decreased. For this reason, output value is changing.

The above graph is the relationship between actual distance and PSD sensor voltage. In order to measure the distance, the formula is ;

Distance = 27.5 / ((5 * AD result value) / 1024)

This has included a little tolerance. But this is to judge the object detection, not a distance measure. Therefore, you can use from 15cm to 45cm distance without problem.

Let's check out the distance measurement program using PSD sensor and ATmega128 ADC(Analog Digital Converter),

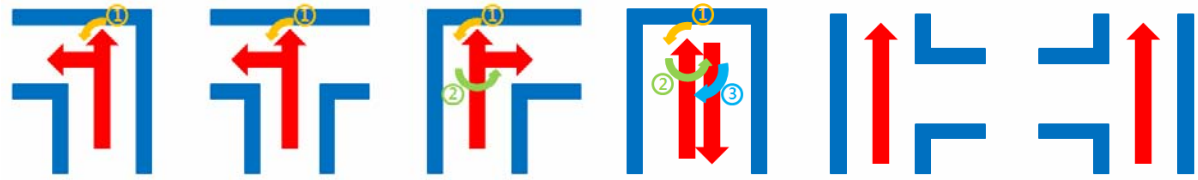
```
//-----------------------------------------------------------------------------
// Read Distance from PSD / "Adc.c"
//-----------------------------------------------------------------------------
void Get_AD_PSD(void)
{
    float  tmp = 0;                         // variable for distance calculation
    float  dist;                            // variable for distance calculation

    EIMSK &= 0xBF;                          // outer interrupt no.6 inactivate ( IR receiving inactivate)
    PSD_ON;                                 // PSD ON
    delay_ms(50);                           // Idle status until PSD Power On
    gAD_Ch_Index = PSD_CH;                  // ADC PSD channel selection
    F_AD_CONVERTING = 1;                    // AD convert finish flag set
    ADC_set(ADC_MODE_SINGLE);               // select AD mode
    while(F_AD_CONVERTING);                 // AD convert finish flag clear
    tmp = tmp + gPSD_val;                   // save AD convert result
    PSD_OFF;                                // PSD Power Off
    EIMSK |= 0x40;                          // IR receiving reactivate

    dist = 1117.2 / (tmp - 6.89);           // PSD value calculate ➔ calculation by experiment
    //dist = 27.5 / (5.0*(float)(gAD_val&0x03ff)/1024.0);   // ➔ PSD calculation formula
    if(dist < 0) dist = 50;                 // check distance limit
    else if(dist < 10) dist = 10;
    else if(dist > 50) dist = 50;
    gDistance = (BYTE)dist;                 // save the result into gDistance
}



//-----------------------------------------------------------------------------
// A/D convert finish interrupt / "Main.c"
//-----------------------------------------------------------------------------
interrupt [ADC_INT] void adc_isr(void) {            // Run interrupt when AD convert is finished.
        WORD i;                                     // Temp. variable
        gAD_val = (signed char)ADCH;                // Save 10bit AD convert result
        switch( gAD_Ch_Index ){                     // Select channel
                case  PSD_CH    :                   // PSD channel
                        gPSD_val = (BYTE) gAD_val;          // Save into PSD variable
                        break;
                case  VOLTAGE_CH        :           // Battery voltage check channel
                        i = (BYTE) gAD_val;
                        gVOLTAGE = I * 57;
                        break;
```

```
                case   MIC_CH     :                              // Mic. Input channel
                       if((BYTE)gAD_val < 230)
                              gMIC_val = (BYTE)gAD_val;
                       else   gMIC_val = 0;
                       break;
        }
        F_AD_CONVERTING = 0;                                     // AD convert flag clear)
}
```

As Get_AD_PSD() function is called, measured distance value is saved in gDistance variable.

In order to escape the maze, use humanoid basic motion "move forward", "move back ward", "turn left", "turn right" in "HunoBasic.h" file. Below is the method to escape the maze.

Example)



1. If there is no wall, move forward.
2. If the wall is detected, check left side. → Turn ①
3. If front and left side wall are detected, check right side. → Turn ②
4. If front, left and right side walls detected, go to opposite way. → Turn ③

In order to do this, robot should turn 90 degree. 90 degree turning is possible if HUNO's turn left or and turn right 3 times continuously in flat floor. In the above example, ① robot turns left 90 degree (turn left 3 times), ② 180 degree turn left (turn left 6 times) ③ 90 degree turn right (turn right 3 times), and move forward motion are needed.

Let's find out how HUNO motions are conducted through which functions.
Below is the part of "M_Play (BYTE BtnCode)" function in "Comm.c".

```
void M_Play(BYTE BtnCode)
{
	// ① Check whether remote controller BTN_C is pressed, and run BasicPose
	if(BtnCode == BTN_C){
		BasicPose(F_PF, 50, 1000, 4);
	}
	if(F_PF == PF1_HUNO){				// ② Check platform (Huno, Dino, Dogy)
		switch(BtnCode){			// ③ Depends on the pressed button code
			case BTN_A:
				SendToSoundIC(7);
				gpT_Table= HUNOBASIC_GETUPFRONT_Torque;
				gpE_Table= HUNOBASIC_GETUPFRONT_Port;
				gpPg_Table	= HUNOBASIC_GETUPFRONT_RuntimePGain;
				gpDg_Table	= HUNOBASIC_GETUPFRONT_RuntimeDGain;
				gpIg_Table	= HUNOBASIC_GETUPFRONT_RuntimeIGain;
				gpFN_Table	= HUNOBASIC_GETUPFRONT_Frames;
				gpRT_Table	= HUNOBASIC_GETUPFRONT_TrTime;
				gpPos_Table	= HUNOBASIC_GETUPFRONT_Position;
				Motion.NumOfScene = HUNOBASIC_GETUPFRONT_NUM_OF_SCENES;
				Motion.NumOfwCK = HUNOBASIC_GETUPFRONT_NUM_OF_WCKS;
				break;
			case BTN_B:
			default:	return;
		}
	}
	else if(F_PF == PF1_DINO){			// ② Check Platform (Huno, Dino, Dogy)
		switch(BtnCode){			// ③ Depend on the pressed button code
			case BTN_A:
			default:	return;
		}
	}
	else if(F_PF == PF1_DOGY){			// ② Check Platform (Huno, Dino, Dogy)
		switch(BtnCode){			// ③ Depend on the pressed button code
			case BTN_A:
			default:	return;
		}
	}
	else if(F_PF == PF2){				// If platform is non-standard platform, return
		return;
	}
	Motion.PF = F_PF;
	M_PlayFlash();
}
```

You can see that HUNO run motions are different in accordance with conditions.

```
case BTN_LR :   HUNOBASIC_TURNLEFT_xxx
                ....
case BTN_U  :   HUNOBASIC_WALKFORWARD_xxx
                ....
case BTN_RR :   HUNOBASIC_TURNRIGHT_xxx
                ....
case BTN_D  :   HUNOBASIC_WALKBACKWARD_xxx
                ....
```

For move forward, M_Play(**BTN_U**) function is called.

For turn left, M_Play(**BTN_LR**) function is called.

For turn right, M_Play(**BTN_RR**) is called.

For move backward, M_Play(**BTN_D**) is called.

Only different point is the using the sensor and algorithm instead of IR remote controller.

Let's see the C program that robot escapes the maze.

```
void Robot_Turn_Left_90(void){                                  // 90 degree left turn
    M_Play(BTN_LR); M_Play(BTN_LR); M_Play(BTN_LR);             // left turn 3 times
}

void Robot_Turn_Left_180(void){                                 // 180 degree left turn
    M_Play(BTN_LR); M_Play(BTN_LR); M_Play(BTN_LR);             // left turn 5 times
    M_Play(BTN_LR); M_Play(BTN_LR);
}

void Robot_Turn_Right_90(void){                                 // 90 degree right turn
    M_Play(BTN_RR); M_Play(BTN_RR); M_Play(BTN_RR); M_Play(BTN_RR);   // right turn 4 times
}

void Robot_Forward(void){                                   // Move forward
    M_Play(BTN_U);   M_Play(BTN_U);     M_Play(BTN_RR);    // Move forward 2 times + right turn once
}

void Robot_Backward(void){                                      // Move backward
    M_Play(BTN_D);                                              // Move backward once
}

void User_Func_1(void){                                     // Maze escape function
    while(1){                                               // Run continuously
        Get_AD_PSD();                                       // Check front distance (1)
        if( gDistance <= 12 ) Robot_Backward();             // If too close, Move backward
```

```
        else if( gDistance <= 30 ){   Robot_Turn_Left_90();          // If there is wall, 90 degree turn left
            Get_AD_PSD();                                           // Check left side distance (2)
            if((gDistance <= 30)){   Robot_Turn_Left_180();         // If there is wall, 180 degree turn left
                Get_AD_PSD();                                       // Check right side distance (3)
                if((gDistance <= 30))   Robot_Turn_Right_90();      // If there is wall, 90 degree turn right
            }
        }
        else   Robot_Forward();                                     // If there is no wall, move forward
    }
}
```

※ Turn right or left run times could be different depends on the floor status.

In order to use the above program,

1. Include the above code in “main()” function in “Main.c”.

2. Open “DIO.c” file. Add user function in “case BTN_1” line.
   (In basic posture, “User_Func_1()” function runs if you press IR remote controller “1” button.)

```
case BTN_1   :   User_Func_1();
```

3. Press Shift + F9 button to compile it and download “hex” file into RBC box.

4. Press “1” button to run the user program after robot basic posture.

4. C Program Summary

4.1 Variables

A variable is just a named area of storage that can hold a single value (numeric or character). The C language demands that you declare the name of each variable that you are going to use and its type, or class, before you actually try to do anything with it. Variable value initially '0'.

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

```
Ex) int VariableNumber1;                    // int type variable
    int VariableNumber2 = 0;                // int type variable value '0'.
    char VariableNumber3;                   // char type variable
    unsigned char VariableNumber4 = 255;    // char type variable value '255'
```

The Programming language C has two main variable types

Local Variables

Local variables scope is confined within the block or function where it is defined. Local variables must always be defined at the top of a block. When a local variable is defined - it is not initialize by the system, you must initialize it yourself. When execution of the block starts the variable is available, and when the block ends the variable 'dies'.

Global Variables

Global variable is defined at the top of the program file and it can be visible and modified by any function that may reference it. Global variables are initialized automatically by the system when you define them! If same variable name is being used for global and local variable then local variable takes preference in its scope. But it is not a good practice to use global variables and local variables with the same name.

4.2 Operators

Assignment (=)

The assignment operator assigns a value to a variable.

```
a = 5;
```

This statement assigns the integer value 5 to the variable a. The part at the left of the assignment operator (=) is known as the lvalue (left value) and the right one as the rvalue (right value). The lvalue has to be a variable whereas the rvalue can be either a constant, a variable, the result of an operation or any combination of these.
The most important rule when assigning is the right-to-left rule: The assignment operation always takes place from right to left, and never the other way:

```
a = b;
```

This statement assigns to variable a (the lvalue) the value contained in variable b (the rvalue). The value that was stored until this moment in a is not considered at all in this operation, and in fact that value is lost.

Consider also that we are only assigning the value of b to a at the moment of the assignment operation. Therefore a later change of b will not affect the new value of a.

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see is modulo; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

```
a = 11 % 3;
```

the variable a will contain the value 2, since 2 is the remainder from dividing 11 between 3.

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators:

expression	is equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C++:

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical operators (!, &&, ||)

The operator! is the operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

```
!(5 == 5)     // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)     // evaluates to true because (6 <= 4) would be false.
!true         // evaluates to false
!false        // evaluates to true.
```

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise.

Bitwise Operators (&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise Inclusive OR
^	XOR	Bitwise Exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift Left
>>	SHR	Shift Right

4.3 Control Statement

The if else Statement

This is used to decide whether to do something at a special point, or to decide between two courses of action. The following test decides whether a student has passed an exam with a pass mark of 45

```
if (result >= 45)
        printf("Pass\n");
else
        printf("Fail\n");
```

Each version consists of a test, (this is the bracketed statement following the if). If the test is true then the next statement is obeyed. If is is false then the statement following the else is obeyed if present. After this, the rest of the program continues as normal. If we wish to have more than one statement following the if or the else, they should be grouped together between curly brackets. Such a grouping is called a compound statement or a block.

```
if (result >= 45)
{       printf("Passed\n");
        printf("Congratulations\n")
}
else
{       printf("Failed\n");
        printf("Good luck in the resits\n");
}
```

The for Loop

The for loop works well where the number of iterations of the loop is known before the loop is entered. The head of the loop consists of three parts separated by semicolons.

The example is a function which calculates the average of the numbers stored in an array. The function takes the array and the number of elements as arguments.

```
float average(float array[], int count)
{       float total = 0.0;
        int i;

        for(i = 0; i < count; i++)
                total += array[i];

        return(total / count);
}
```

The for loop ensures that the correct number of array elements are added up before calculating the average.

The while Loop

The while loop repeats a statement until the test at the top proves false.

As an example, here is a function to return the length of a string.

```
int string_length(char string[])
{       int i = 0;

        while (string[i] != '\0')
                i++;
        return(i);
}
```

The string is passed to the function as an argument. The size of the array is not specified, the function will work for a string of any size. The while loop is used to look at the characters in the string one at a time until the null character is found. Then the loop is exited and the index of the null is returned. While the character isn't null, the index is incremented and the test is repeated.

4.4 Functions

A C function definition consists of return value (void if no value is returned), a unique name, a list of parameters in parentheses (void if there are none), and various statements. A function with non-void return type should include at least one return statement:

```
<return-type> functionName( <parameter-list> )
{
      <statements>
      return <expression of type return-type>;
}
```

where <parameter-list> of n variables is declared as data type and variable name separated by a comma:

```
      <data-type> var1, <data-type> var2, ... <data-type> varN
```

The following program shows use of a function pointer for selecting between addition and subtraction:

```
#include <stdio.h>

int add(int x, int y)
{
      return x + y;
}
 int subtract(int x, int y)
{
      return x - y;
}
 int main(int argc, char* args[])
{
    int    foo = 1, bar = 1;
      printf("%d + %d = %d\n", foo, bar, add(foo, bar));
      printf("%d - %d = %d\n", foo, bar, subtract(foo, bar));
    return 0;
}
```

4.5 Arrays and Pointers

To fully understand the workings of C you must know that pointers and arrays are related.

An array is actually a pointer to the 0th element of the array. Dereferencing the array name will give the 0th element. This gives us a range of equivalent notations for array access.

Array Access	Pointer Equivalent
`arr[0]`	`*arr`
`arr[2]`	`*(arr + 2)`
`arr[n]`	`*(arr + n)`

There are some differences between arrays and pointers. The array is treated as a constant in the function where it is declared. This means that we can modify the values in the array, but not the array itself, so statements like arr ++ are illegal, but arr[n] ++ is legal.

Since an array is like a pointer, we can pass an array to a function, and modify elements of that array without having to worry about referencing and de-referencing. Since the array is implemented as a hidden pointer, all the difficult stuff gets done automatically.

A function which expects to be passed an array can declare that parameter in one of two ways.

```
int arr[];  or int *arr;
```

Either of these definitions is independent of the size of the array being passed. This is met most frequently in the case of character strings, which are implemented as an array of type char. This could be declared as char string[]; but is most frequently written as char *string; In the same way, the argument vector argv is an array of strings which can be supplied to function main. It can be declared as one of the following.

```
char **argv;  or  char *argv[];
```

4.6 Structure

A structure type is usually defined near to the start of a file using a typedef statement. typedef defines and names a new type, allowing its use throughout the program. typedefs usually occur just after the #define and #include statements in a file.

Here is an example structure definition.

```
typedef struct {
        char name[64];
        char course[128];
        int age;
        int year;
} student;
```

This defines a new type student variables of type student can be declared as follows.

```
student st_rec;
```

Notice how similar this is to declaring an int or float.
The variable name is st_rec, it has members called name, course, age and year.

Each member of a structure can be used just like a normal variable, but its name will be a bit longer. To return to the examples above, member name of structure st_rec will behave just like a normal array of char, however we refer to it by the name

```
st_rec.name
```

Here the dot is an operator which selects a member from a structure.
Where we have a pointer to a structure we could dereference the pointer and then use dot as a member selector. This method is a little clumsy to type. Since selecting a member from a structure pointer happens frequently, it has its own operator -> which acts as follows. Assume that st_ptr is a pointer to a structure of type student We would refer to the name member as

```
st_ptr -> name
```

As we have seen, a structure is a good way of storing related data together. It is also a good way of representing certain types of information. Complex numbers in mathematics inhabit a two dimensional plane (stretching in real and imaginary directions). These could easily be represented here by

```
typedef struct {
        double real;
        double imag;
} complex;
```

doubles have been used for each field because their range is greater than floats and because the majority of mathematical library functions deal with doubles by default.

In a similar way, structures could be used to hold the locations of points in multi-dimensional space. Mathematicians and engineers might see a storage efficient implementation for sparse arrays here.

Apart from holding data, structures can be used as members of other structures. Arrays of structures are possible, and are a good way of storing lists of data with regular fields, such as databases.

Another possibility is a structure whose fields include pointers to its own type. These can be used to build chains (programmers call these linked lists), trees or other connected structures. These are rather daunting to the new programmer, so we won't deal with them here.

Appendix A. wCK Communication Protocol

Command		Command Packet: byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	…	byte	byte	byte	Response Packet: byte 1	byte 2
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
8bit Command	Position Move	header	Torq / ID 0~30	target position 0~254	check sum (note1)											current 0~254	position 0~254
8bit Command	Syncronized Position Move	0xFF	0~4 / 31	X / lastID+1 1~31	ID0 target 0~254	ID1 target 0~254	ID2 target 0~254	ID3 target 0~254	ID4 target 0~254	ID5 target 0~254	ID6 target 0~254	…	lastID-1 target 0~254	lastID target 0~254	check sum (note2)		
8bit Command	Status Read	0xFF	mode 5 / ID 0~30	X	check sum											current 0~254	position 0~254
8bit Command	Passive wCK	0xFF	6 / ID 0~30	mode 1 / X	(note1)											byte3	position 0~254
8bit Command	Wheel wCK	0xFF	6 / 0~30	3=CCW 4=CW / speed 0~15	(note1)											Rotation Count	position 0~254
8bit Command	Break wCK	0xFF	6 / 31	2 / X	(note1)											ID	position 0~254
Config. Command	Baudrate Set	0xFF	7 / ID 0~30	mode 8	new baudrate 0~191	= byte4	check sum									new baudrate	new baudrate
Config. Command	P,D gain Set	0xFF	7 / 0~30	9	new P gain 1~254	new D gain 0~254	(note3)									new P gain	new D gain
Config. Command	P,D gain Read	0xFF	7 / 0~30	10	X	= byte4	(note3)									P gain	D gain
Config. Command	Runtime P,D gain Set	0xFF	7 / 0~30	11	P gain 1~254	D gain 0~254	(note3)									P gain	D gain
Config. Command	ID Set	0xFF	7 / 0~30	12	new ID 0~254	= byte4	(note3)									new ID	new ID
Config. Command	SPEED Set	0xFF	7 / 0~30	13	speed 0~30	accel 20~100	(note3)									new Speed	new Accel
Config. Command	SPEED Read	0xFF	7 / 0~30	14	X	= byte4	(note3)									Speed	Accel
Config. Command	Over Load Set	0xFF	7 / 0~30	15	new overcur T 33~199	= byte4	(note3)									new overcur T	new overcur T
Config. Command	Over Load Read	0xFF	7 / 0~30	16	X	= byte4	(note3)									overcur T	overcur T
Config. Command	Boundary Set	0xFF	7 / 0~30	17	new U bound 0~254	new L bound 0~254	(note3)									new U bound	new L bound
Config. Command	Boundary Read	0xFF	7 / 0~30	18	X	= byte4	(note3)									U bound	L bound
Config. Command	I gain Set	0xFF	7 / 0~30	21	new I gain 0~10	= byte4	(note3)									new I gain	new I gain
Config. Command	I gain Read	0xFF	7 / 0~30	22	X	= byte4	(note3)									I gain	I gain
Config. Command	Runtime Speed Set	0xFF	7 / 0~30	23	speed 0~30	accel 20~100	(note3)										
Config. Command	Runtime I gain Set	0xFF	7 / 0~30	24	I gain 0~10	I gain 0~10	(note3)									P gain	D gain
Extended Command	I/O Write	0xFF	7 / 0~30	100	X / ch 1 0	= byte4	(note3)									Output Value	Output Value
Extended Command	I/O Read	0xFF	7 / 0~30	101	X	= byte4	(note3)									Output Value	8bitAD
Extended Command	Motion DATA Write	0xFF	7 / 0~30	150	Motion Count 0~8	Motion Cmd 1 0~254	Motion DATA 1 0~254	Motion Cmd 2 0~254	Motion DATA 2 0~254	Motion Cmd 3 0~254	Motion DATA 3 0~254	…		check sum (note5)		Motion Count 0~8	Motion Count 0~8
Extended Command	Motion DATA Read	0xFF	7 / 0~30	151	X	= byte4	check sum (note3)									Motion Count 0~8	Motion Count 0~8
10bit Command	Position Move	0xFF	7 / x	200	ID 0~253	Torq 0~254	target(H3) 0~1023	target(L7) 0~1023 X	check sum (note7)							position(H3) 0~1023	position(L7) 0~1023 X
10bit Command	Position Read	0xFF	7 / x	201	ID 0~253	= byte4	check sum (note6)									position(H3) 0~1023	position(L7) 0~1023 X

※ X : don't care
※ **note1** : CheckSum = (byte2**XOR** byte3) **AND** 0x7F
※ **note2** : CheckSum = (byte4**XOR** … (byte(lastID+3)**AND** 0x7F
※ **note3** : CheckSum = (byte2**XOR** byte3 **XOR** byte4 **XOR** byte5) **AND** 0x7F
※ **note4** : CheckSum = (byte2**XOR** byte3 **XOR** byte4) **AND** 0x7F
※ **note5** : CheckSum = (byte2**XOR** byte3 **XOR** … byteN) **AND** 0x7F
※ **note6** : CheckSum = (byte1**XOR** byte2 **XOR** … byteN) **AND** 0x7F
※ **note7** : CheckSum = (byte2**XOR** byte3 **XOR** byte4 **XOR** byte5 **XOR** byte6) **AND** 0x7F
※ Motion DATA commands : Selft-running motion mode is activated only when the No. of Instruction is more than 0.