Learning With Robot

AVR C Programming



- ATmega128 Micro Controller
- Learn to write Robot controller DRC firmware

Dongbu Robot Archive

(Download manuals and applications.)

www.dongburobot.com

Dongbu Robot

Dongbu Robot

Using Robot to Learn AVR C Programming

1st Edition published date :
Publisher : Dongbu Robot Co., Ltd
Address : 11th Fl Bucheon Techno Prk Bldg, 401, Yakdae-dong, Wonmi-gu, Bucheon-si, Gyeonggi-do, S Korea
Customer Inquiry : 032-329-5551
FAX : 032-329-5569

No portion of this purblication may be reproduced or transmitted in any form without prior written authorization from Dongbu Robot Co., Ltd.

Dongbu Robot Co.,LtdAll rights reserved

Contents

PART 01 Learning Hovis DRC & ATMega128

Chapter 01 Hovis DRC & Sensor ----------

PART 02 Learning Visual Logic Related C Language Grammar

Chapter 01 C Language with Robot ----------
Chapter 02 C Brief Glance ----------
Chapter 03 Variable ----------
Chapter 04 Array, Pointer, Struct ----------
Chapter 05 Memory ----------
Chapter 06 Struct ----------
Chapter 07 Iteration & branching----------
Chapter 08 Function ----------

PART 03 Learning AVR Robot Controller Firmware

Chapter 00 **AVR Studio Installation & Hello Programming ---------**

Chapter 01 **LED ----------**
AVR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 02 **Button ----------**
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 03 **Buzzer ----------**
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 04 **External Interrupt MIC ----------**
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 05 **Timer/Counter ----------**
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 06 **PSD (ADC) ----------**
AVR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution -------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 07 **USART (Serial) ----------**
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 08 **Servo Motor** ----------
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 09 **Busser Advanced** ----------
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 10 **EEPROM** ----------
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 11 **Acceleration / Gyro Sensor** ----------
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

Chapter 12 **IR Remote Control**----------
VR Fundamentals & H/W Fundamentals ----------
DRC & H/W Preparation & AVR Studio & S/W Execution ------
Example & Description ----------
Creating Hex & DRC Operation ----------

PART 04 Applying to the Robot

Chapter 01

Robot Motion Control----------

VR Fundamentals & H/W Fundamentals ----------

DRC & H/W Preparation & AVR Studio & S/W Execution ------

Example & Description ----------

Creating Hex & DRC Operation ----------

Chapter 02

Dump Truck with Remote Control ----------

VR Fundamentals & H/W Fundamentals ----------

DRC & H/W Preparation & AVR Studio & S/W Execution ------

Example & Description ----------

Creating Hex & DRC Operation ----------

Recommended Standard Curriculum

Lect 1 : DRC Outline, Hovis Outline, AVR Outline, AVR Installation, Hello DRC AVR

Lect 2 : C Language – Variable , Control Statement,Function, Array, Pointer, Struct

Lect 3 ~ 7 : AVR Example – LED, Button, Buzzer,Acceleration,MIC,Timer

Lect 8 : Mid Term Test

Lect 9 ~ 15 : AVR Example – Humanoid Motion, Dump Truck, PSD, EEPROM, USART, Motor, ZigBee Applied Robot

Lect 16 : Final Exam

Appendix : DRC Register, Useful Info

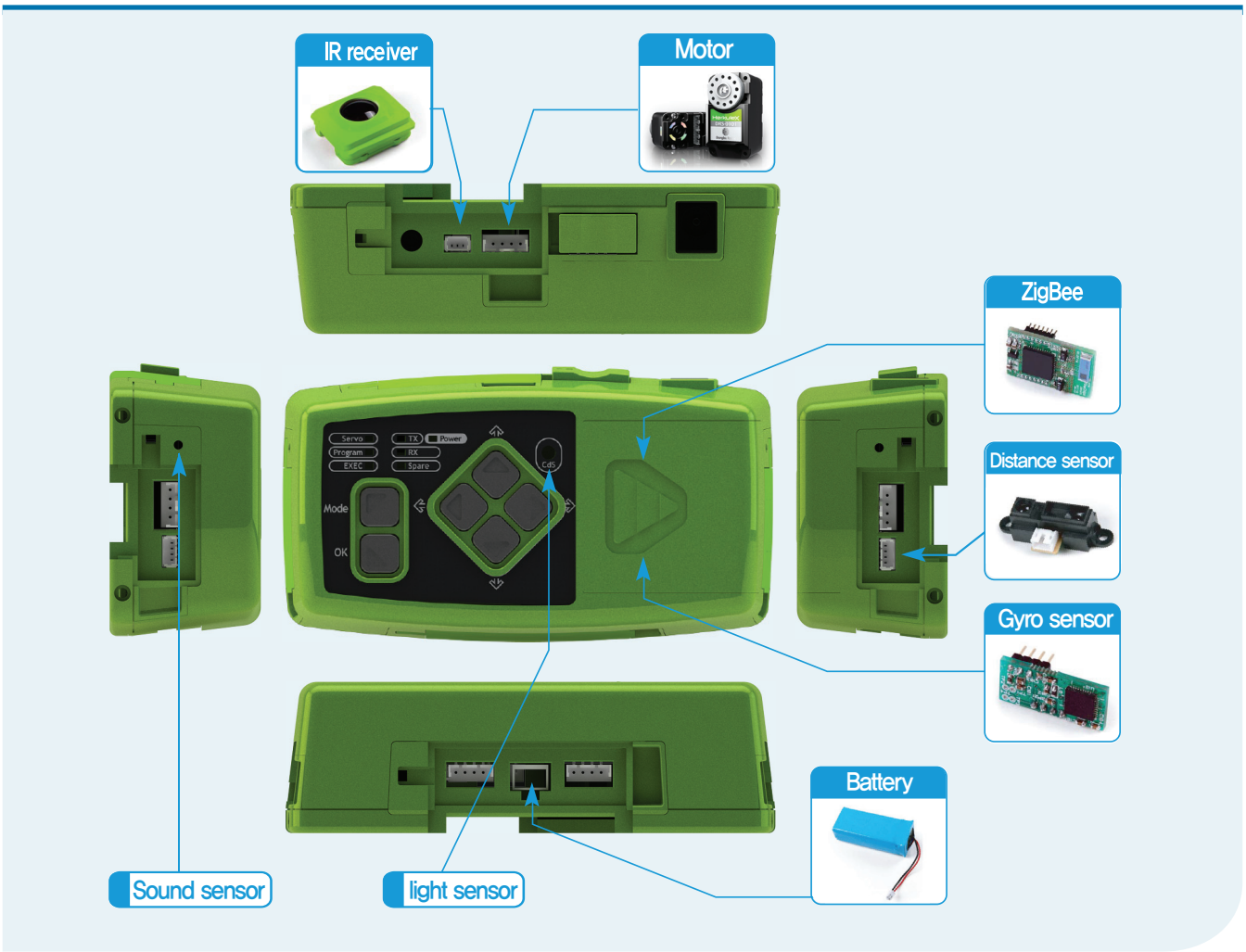
PART 01

Learning

Hovis DRC & ATMega128

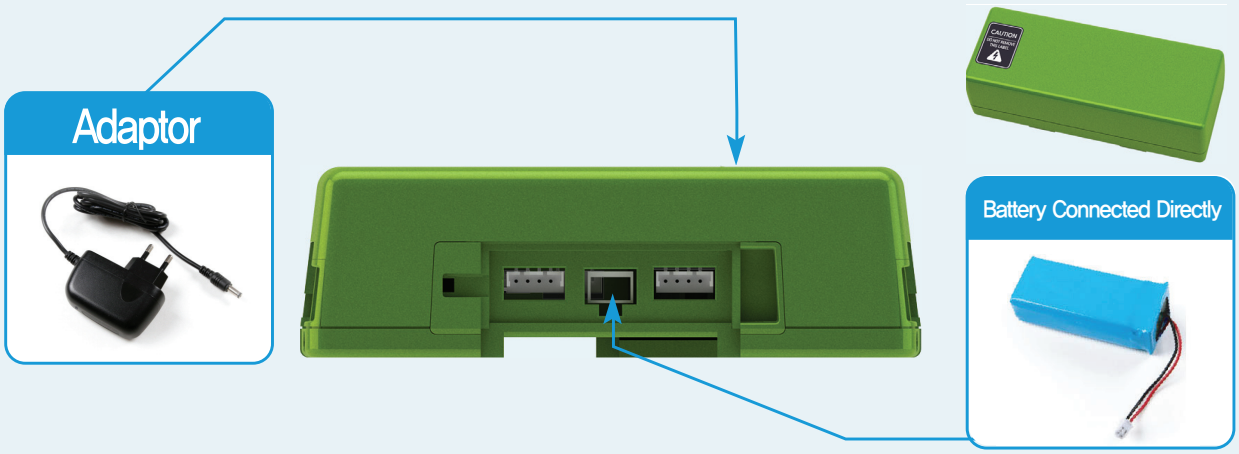
Overview

DRC Controller is the main component and brain of the Hovis Lite. Controller has variety of connectors and interfaces including 6 servo motor ports, 2 PSD sensor ports, Gyro sensor, and ZigBee interface. Light sensor and the sound sensor in built-in to the controller. DR-Visual Logic program is a visual robot programming language that uses DRC functions to program the robot. Various sensors and 1~32 motors can be programmed and tested.



CPU	ATMega 128
Size, Weight	108 x 58.5 x 33 (mm), 82 g
Operating Voltage	Tolerance Range : 6.5V ~ 10V, Recommended Voltage : 7.4V
Serial Speed	115,200 bps ~ 666,667 bps
Consumed Current	When IDLE : 50mA, Overall Max Current : 3A (PTC Fuse)
Interface	Button : 6ea, MIC : 2ea, LED : 7ea
External I/O	Servo Motor : 6ea, PSD Sensor : 2ea
Back Cover I/O	ZigBee : 1ea, Gyro Sensor : 1ea
Internal I/O	Sound Sensor : 2ea, Light Sensor : 1ea

DRC Battery Installation



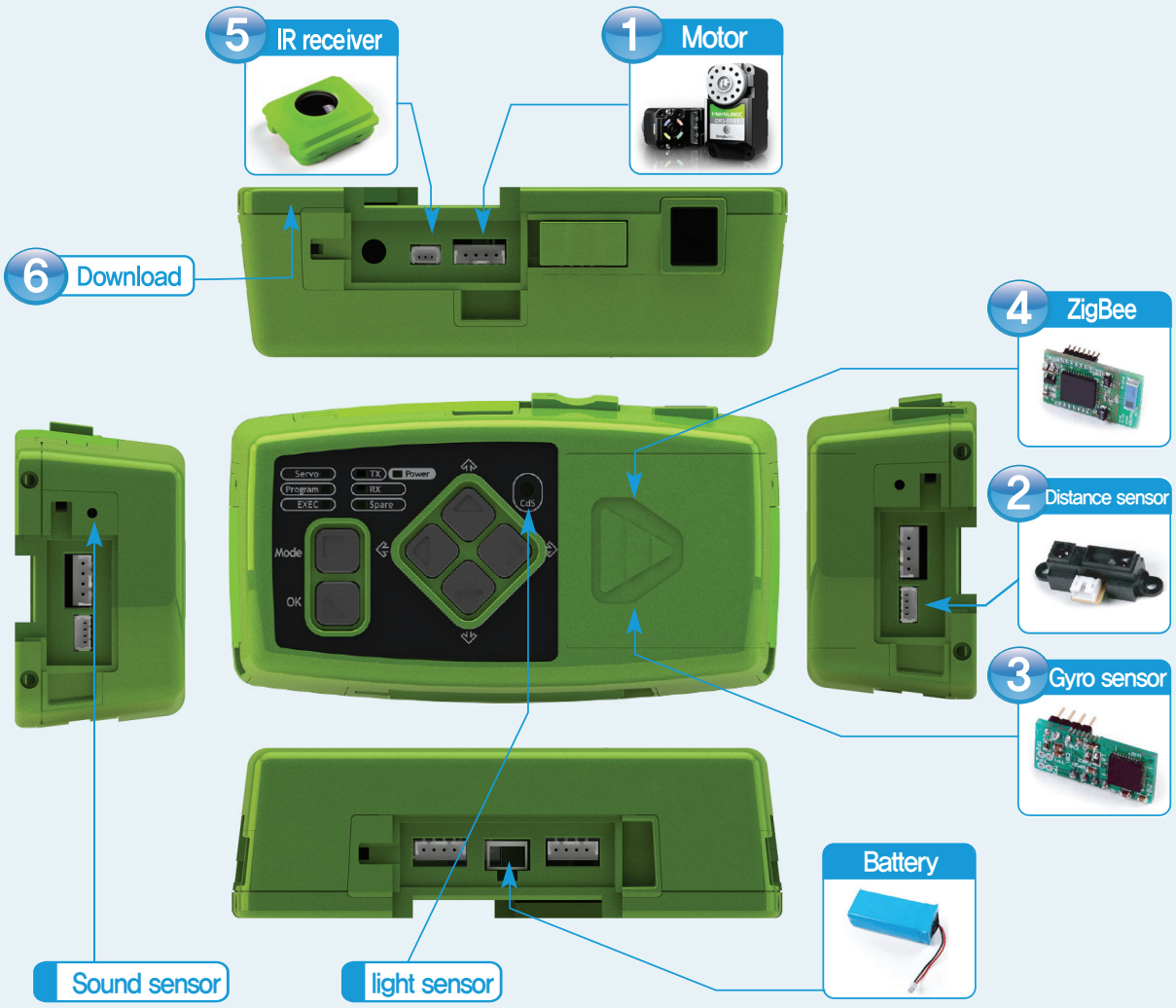
- **Battery**
  To supply power to the DRC, connect the battery to controller by the power connector found at bottom.

- **Battery Charging**
  Battery can be charged directly using the battery cable and the adapter. Another charging method is to connect the adapter to the adapter connector found at top of the controller. Battery installed on the controller will start to charge automatically when the adapter is connected to the controller.



- **Low Battery**
  Power LED will start to blink when the battery level falls below 20%.

DRC Connection



1 Motor Connection
There are five ports around the controller

2 Distance Sensor Connection
There are two ports, one on each side.

3 Gyro Sensor Connection
Open the controller cover and install internally.

4 ZigBee Connection
Open the controller cover and install internally.

5 IR Receiver Connection
1 port, used for receiving remote control signal.

6 Download Connection
Ear phone Jack connection used to downolad program from PC to the robot.

DRC Interface



DRC controller has Input/Output buttons and LEDs at the front and motor and sensor ports at the back. Interface buttons at the front are used to give input commands the LEDs are used to verify data output.

	Name	Short Cut	Standard Task Mode
Main Button	Mode	Run Task	
	Ok	Confirm	
Navi Key	(Left)	Battery Level	Check Mode
	(Up)	DRC Self Test	Autonomous Movement Mode
	(Right)	Switch wired/wireless com	Remote Control Mode
	(Down)	Motor ID Scan	Sound Demo Mode
LED	Servo	HerkuleX Running	
	Program	DR-SIM/Visual Logic Running	
	EXEC	Task Running	
	TX	Data Transmit	
	RX	Data Receive	
	Spare	User Defined	
	Power	Power	
Sensor	Cds	Light Sensor	

DRC Register Map

Register

DRC has a registers which contains current controller state, settings, and various sensor related data.
For example, number of motors connected to the robot and their ID, error status, and current error codes are all part of current controller state. Controller settings include such data as Min/Max input voltage, Ack Policy, and etc. Sensor readings such as luminosity detected by the light sensor and location of the detected sound are part of sensor data.

Controller register is divided into (Non-Volatile, EEPROM) register and (Volatile, RAM) register. Non-Volatile registers retain data even when the power has been turned off and contain basic sestup values pertaining to the controller operation. Values in the Non-Volatile registers are copied to Volatile registers as soon as the power is turned on. Volatile registers contain controller settings, state, and sensor values. Data in the Volatile registers have direct effect on the operation of the controller.

Knowing the content of the the registers and how the content changes allow the user to write more refined robot motion progam using DR-Visual program. Knowledge about registers also help the user to read the the controller status and to change the operatonal settings, making robot operation more convenient.

Protocol

Protocol is a predefined format or rules for commands that are given to read or write to registers. Protocol is defined not only for read/write commands but also for other commands such as run commands for running saved tasks or sounds , reboot command for rebooting the controller, and host of other commands.

Communication between the PC and the controller use such predefined protocols to send and receive packets. DR-SIM and DR-Visual prgrams provided by Dongbu Robot were also created using such protocols. User should become familiar with the protocols in order to controll the DRC using their custom made programs.

Refer to DRC Regisers and Protocols section in the manual for more information.

Learning Hovis DRC & ATMega128

DRC Funcitons

1 Program Overview

Firmware : Internal program that cannot be modified by the user.

Task : User defined prgram that can be modified using the Task Editor (DR-Visual Logic).

At the time of release, basic humanoid Task Program is defined. Program can be modified by the user.

2 Operating Method

Basic functions in the firmware will start to operate when power is turned on and Navi key pressed. Pressing the Mode button will run the saved task. From running a basic task, press the Navi key and OK after the task to select which mode to go into

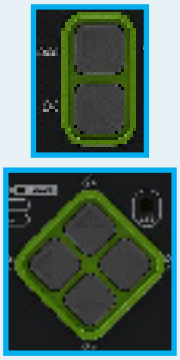
---

Firmware (Cannot be changed by the user)

1 Navi button click



2 Mode or Navi click

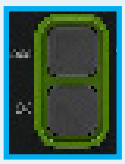


3 Controller self test



Task Mode (Can be changed by the user)

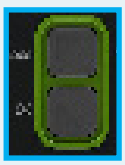
1 Mode click



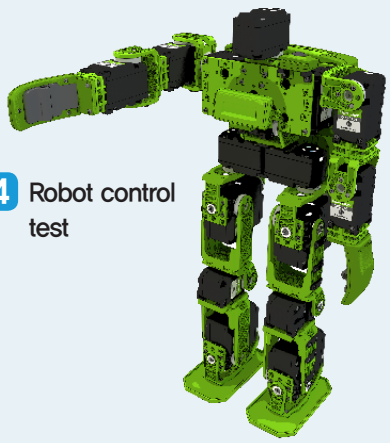
2 Navi click



3 Ok click



4 Robot control test



DRC Functions

3 Operation & Functions

	Name	Description	Operation & Functions
Main Button	Mode	Mode Change	Start Task For standard Task, Mode→ Navi key → Ok (to select operating mode)
	OK	Confirm button	
Navi Key (Firmware)	(L)	Battery level check	Battery level check → Shown by 3 left LED, low 1개 LED, medium 2 LED, high 3 LED
	(Up)	Test	Motor & sensor test using the controller Method : (Up) → Button → Test according to motor response from sensor Test : Motor/Light/Sound/Distance/Accel/Gyro
	(R)	Wired/Wireless	Wired using Ear phone jack / Wireless using ZigBee
	(Down)	Motor ID Scan	Rescan connnected motor ID
Navi Key (Basic Task for Humanoid)	(L)	Check Mode	Mode → (L) → Ok : Check Mode : Individual motors, Arm/Leg module connection and assembly check.
	(Up)	Autonomous Mode	Mode → (Up) → Ok : Autonomous Mode : Robot operates by itself.
	(R)	Remote Control Mode	Mode → (R) → Ok : Remote Control Mode: Run predefined motion saved in the remote control.
	(Down)	Sound Dem Mode	Mode → (Down) → Ok : Sound Demo Mode : Run motions based on sound input.
LED_mode	Servo	HerkuleX running	Blinks when HerkuleX Manager is in operation
	Program	DR-SIM/Visual Logic running	Blinks when DR-SIM / Visual Logic is being used for editing. LED on when downloading data or firmware
	EXEC	Mode chage/Task	On while the task is running when Task mode is entered using the mode button.
	TX	Data Transmit	Blinks when transmitting data, User spae when task in operation.
	RX	Data Receive	Blinks when receiving data, user spae when task in operation.
	Spare	User Defined	
	LED Blink	Error	3 right side LEDs will blink when in error
LED_Power	Power	Power level	Blinks when battery level is below 20%
Sensor	CdS	Light Sensor	Light sensor

DRC Basic Test

DRC is capable of running basic tests through the test mode even when the robot is not assembled. Proceed with the motor and sensor test by turning on the power and pressing the (up) button. Sensor test is performed by checking the motor response from the motors ID1 and 2 attached to left and right. Tests can be performed for motor, light sensor, sound sensor, distance sensor, and gyro sensor. Testing methods are as follows.

Light sensor and sound sensor tests are done by menu and OK as they are built in to the controller
For PSD, Acc/Gyro, pressing the (down) button will check to see if only one of the sensor is connected. Test will proceed if only one sensor is connected and stop if more than one sensor is connected.

1 Motor : Connect motors to the left and right(No 1 on left, No 2 on right)
(Up) → (L) : left motor will move, (R) right motor will move, (Up) both motors will move .

2 Lighte Sensor : (Up) → (Menu) : Light sensor in operation, Both motors will move when Cds blocked

3 Sound Sensor : (Up) → (OK) : Sound Sensor in operation,
Clap from left, left motor will move,
Clap from right, right motor will move

Functions below will operate when (Down) button is pressed after connecting the sensor. Only the sensor to be tested should be connected as testing will not work if more than one sensor is connected.

4 PSD Digital : (Up) → Connect digital distance sensor→ (Down) PSD in operation
When object moves within 10cm → Both motors will move
When object moves beyond 10cm (cliff detection) → Both motors will stop
→ Cliff detection

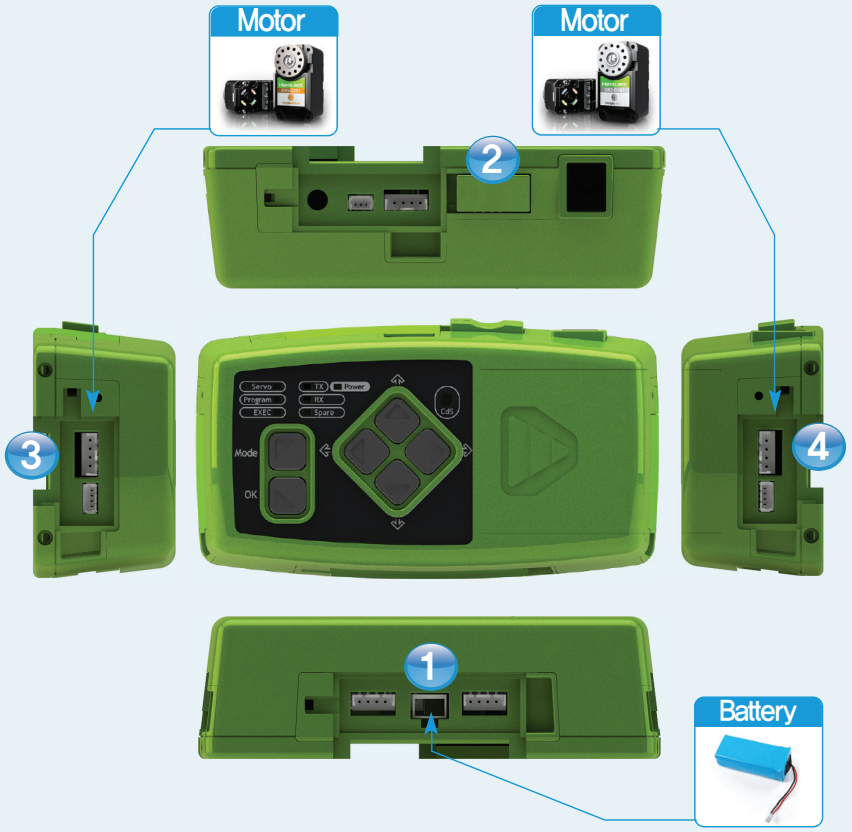
5 PSD Analog : (Up) → Connect analog PSD sensor → (Down) PSD in operation
Both motors will turn in same direction. Farther the object, faster the motor movement
Movement will slow when object comes closer → When object is <10 cm, motors will move in opposite direction.
→ Collision avoidance after wall/object detection

6 Acc : (Up) → Connect Acc/Gyro → (Down) Acc in operation
Motors stopped when the controller angle is same as when the robot is standing straight.
Motor speed will vary depending on the angle. The greater the angle faster the movement.

7 Gyro : (Up) → Connect Acc/Gyro → (Down) Acc in operation → (Down) Gyro in operation
No motore movement when the controller is not moving.
Motor moves at approximately the same speed as the revolving controller.

Follow the detailed test instructions below.

DRC Basic Test : Servo Motor



1 Connect the battery

2 Turn on the power

3 Connect left motor : Make sure to connect Motor ID 1 .(Other motors will not operate.)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not operate.)

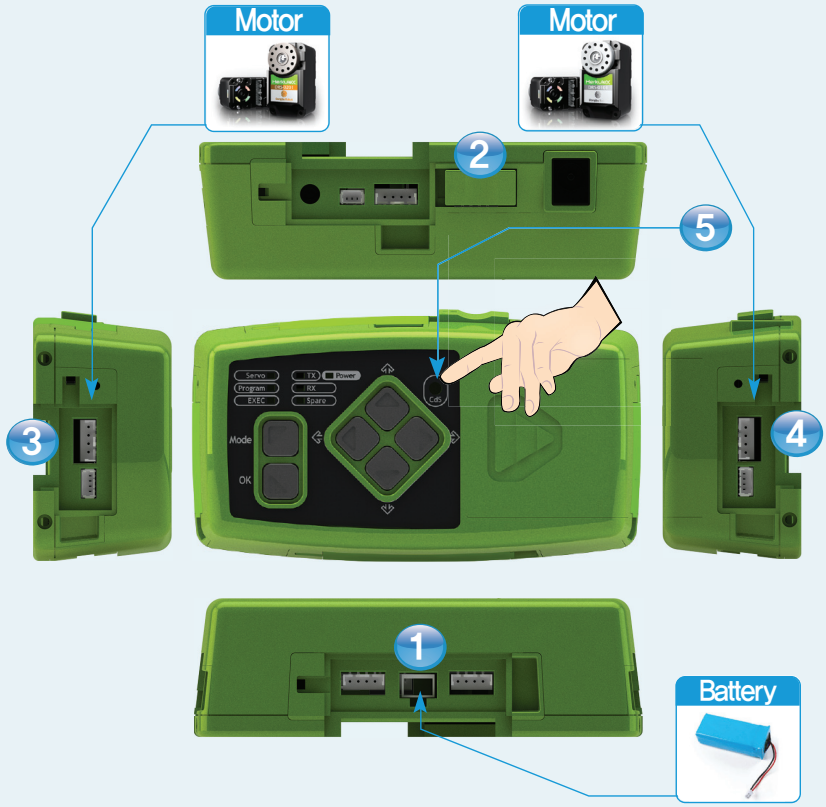
Place the motor outward to test. Simulates wheels turning.

→ Test Process

- Turn on power, Press (Up) button to enter Test Mode
- Press Navi Key (L) button. → Left motor will turn.
- Press Navi Key 의 (R) button. → Right motor will turn.
- Press Navi Key (Up) button. → Both motors will turn in forward direction.

Motors are operating without error if they worked according to the directions above, Results of all following tests will be shown by how the two motors behave. Do not disconnect the motors and continue on with sensor tests.

DRC Basic Tes : Light Sensor



1 Connect Battery

2 Turn on the power

3 Connect left motor : Make sue to connect Motor ID 1. (Other motors will not operate.)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not work.)

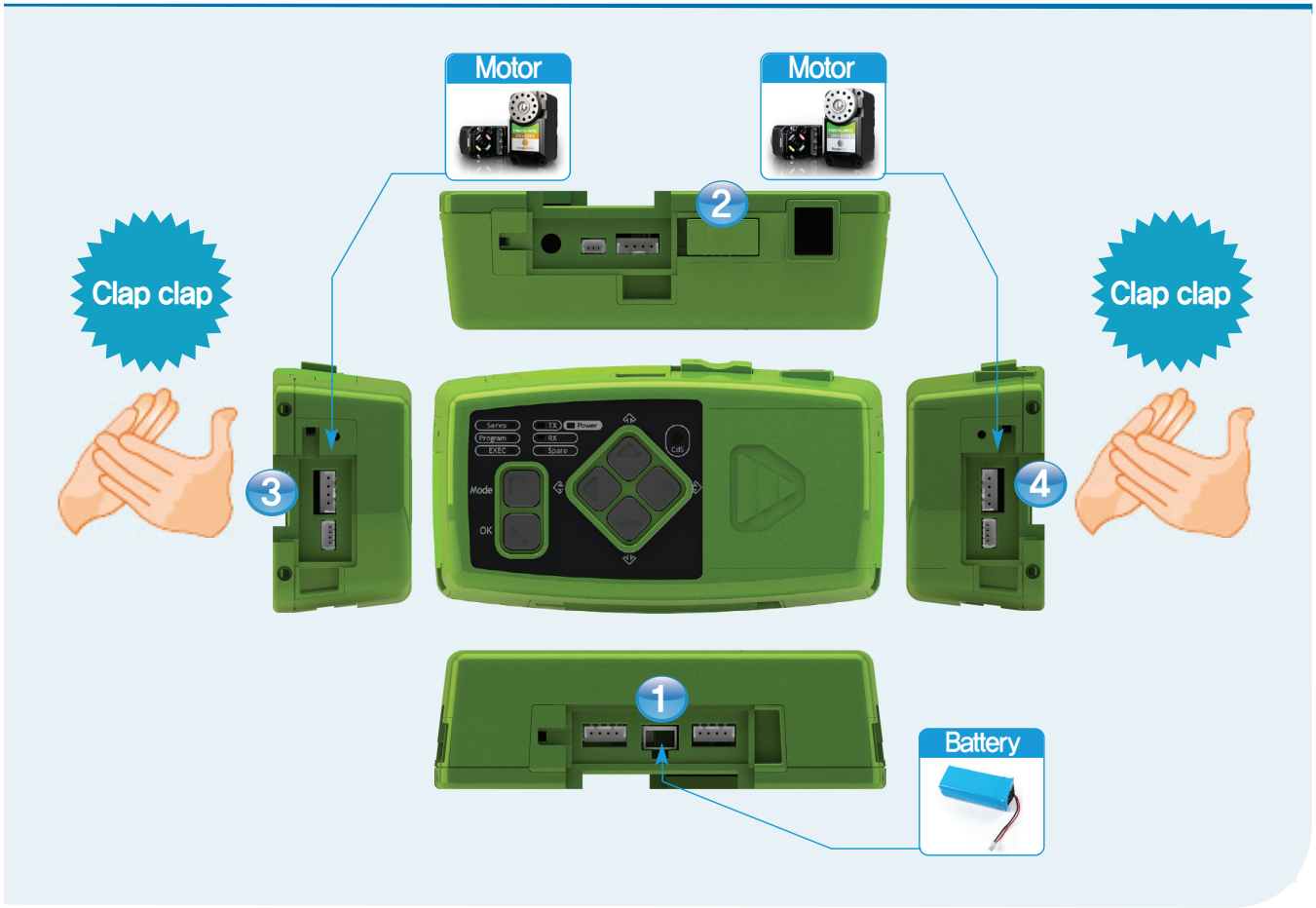
This test simulates robot arms grabbing the air when the light disappears.

→ Test Process

- Turn on power, Press (Up) button to enter Test Mode
- Cover the Cds window with hand. → both motors will turn at the same time.

Light sensor is operating without error if the motors turned accordingly.End light sensor test.

DRC Basic Test : Sound Sensor



1 Connect Battery

2 Turn on the power

3 Connect left motor : Make sure to connect Motor ID 1 .(Other motors will not operate)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not operate)

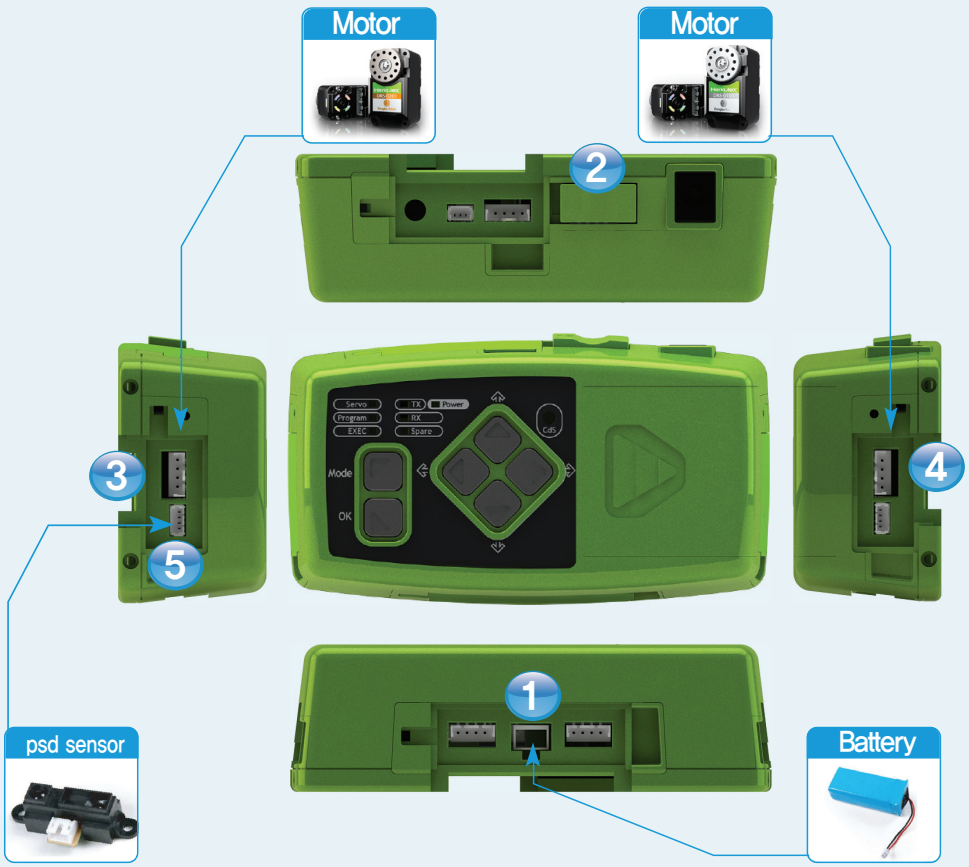
Motor near the direction of the clapping sound will turn.

→ Test Process

- Turn on power, Press (Up) button to enter Test Mode .
- Press (OK) button. → Sound Sensor in operation.
- Clap from left side. → Left motor will turn.
- Clap from right side. → Right motr will turn.

Sound sensor is operating without error if the motors turned accordingly. End sound sensor test.

DRC Basic Test : PSD Digital Distance Sensor



1 Connect Battery

2 Turn on the power

3 Connect left motor : Make sure to connect Motor ID 1 .(Other motors will not operate)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not operate)

5 Connect PSD Digital Sensor

PSD Digital sensor uses certain distance as a base of measure and checks to see how far or near it is. It is normally used to check the depth of the ground to detect steep drop (cliff) and stop the robot.

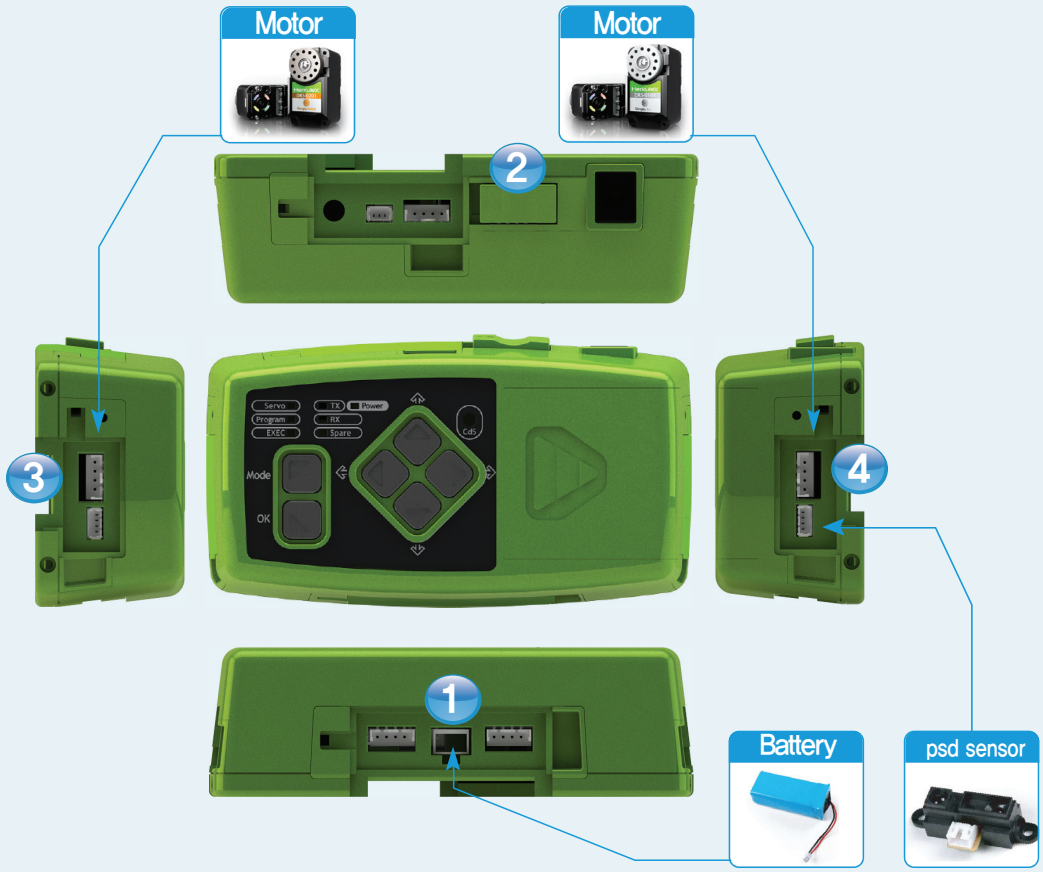
→ Test Process

- Turn on power, Press (Up) button to enter Test Mode.
- Connect the PSD line and press (Down) button. → PSD Digital Sensor in operation.
- Move your hand or an object within 10cm from the senson. → Both motors will turn.
- Move your had or an object away from the sensor to the distance > 10cm . → Both motors will stop.

* PSD Digital has only On/Off mode with certain distance as a base of measure.

PSD Digital Sensor is operating without error if the motors turned accordingly. End PSD Digital Sensor test.

DRC Basic Test : PSD Analog Distance Sensor



1 Connect Battery

2 Turn on the power

3 Connect left motor : Make sure to connect Motor ID 1 .(Other motors will not operate)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not operate)

5 Connect PSD Analog Sensor

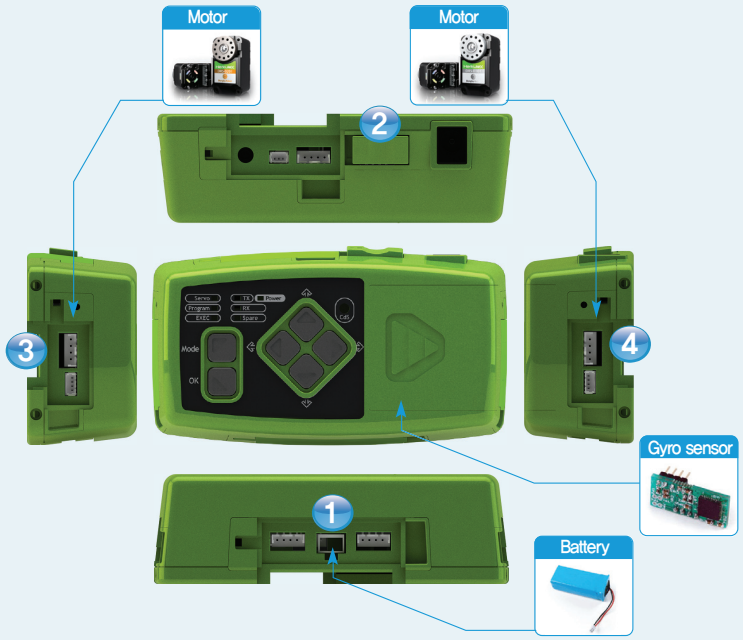
PSD Analog Sensor is able to measure the distance in realtime and control the motor speed according to the distance from an object. Normally used to avoid obstacles by slowing down and chaning direction.

→ Test Process

- Turn on power, press (Up) button to enter Test Mode. Connect PSD line and press (Down)button.
  → PSD Digital Sensor in operation. → Both motors will turn in same direction.
- Place a hand or an object near the sensor and mover away. → Farther the object faster the motor movement.
  → Nearer the object the slower the motor movement.
- When the object is less than 5cm away from the sensor, motors will turn in opposite direction.

PSD Analog Sensor is operating without error if the motors turned accordingly. End PSD Analog Sensor .

DRC Basic Test : Acc/Gyro Sensor



1 Connect Battery

2 Turon on power

3 Connect left motor : Make sure to connect Motor ID 1 .(Other motors will not operate)

4 Connect right motor : Make sure to connect Motor ID 2 .(Other motors will not operate)

Place the motors outward to simulate wheels turning.

5 Connect Acc/Gyro Sensor : Open the controller cover and connect the Acc/Gyro Sensor.

→ Test Process : ACC

- Turn power on and press (Up) button to enter Test Mode. Connect Acc/Gyro and press (Down) button.
  → Acc sensor is in operation.
- Motors stopped when the controller angle is same as when it is attached to the robot standing up straight.
- Tilt the controller slowly. → Speed of the motor will vary with the angle of the tilt. Greater the angle the faster the motr will turn.

Acc sensor is operating without error if the motors turned accordingly.

→ Test Process : Gyro

- Press (Down) button one more time from the ACC test mode. → Gyro sensor is in operation.
- Motors stopped when the controller is not moving.
- Move the controller. → Motors will turn at approximately similar speed to the moving controller.

Acc/Gyro sensor is operating without error if the motors tunred accordingly. End Acc/Gyro sensor test. Both the Acc and Gyro sensor is on a single chip board.

PART 03

AVR robot controller learning Firmware

AVR Installaition & User Guide

AVR programing requires WinAVR compiler and AVR studio .

Download WinAVR from http://sourceforge.net/projects/winavr/files/WinAVR/ .

Download AVR studio from http://www.atmel.com/ .

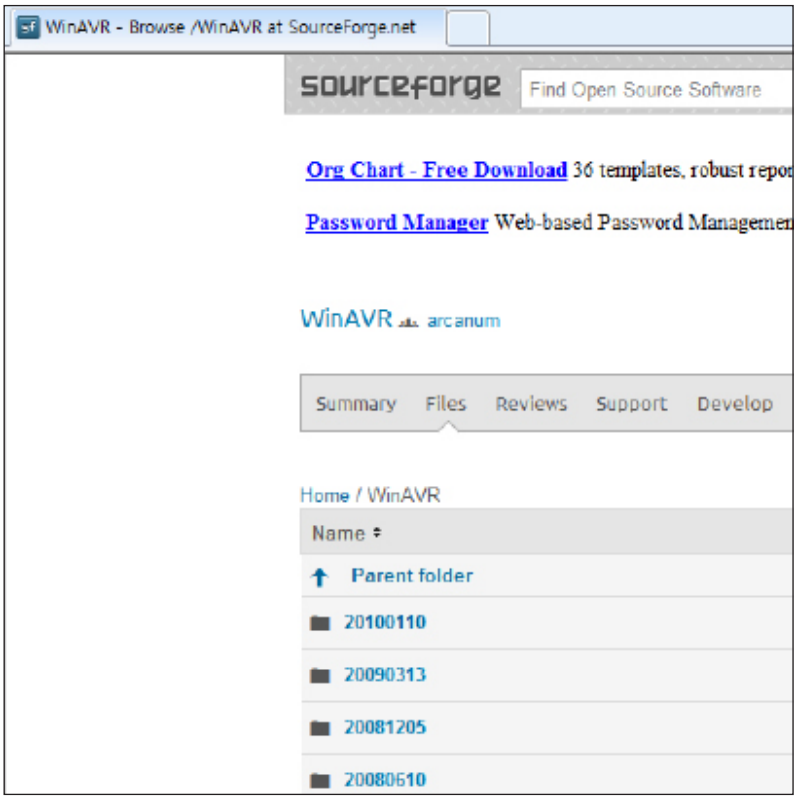


WinAVR Installation

01 Enter URL

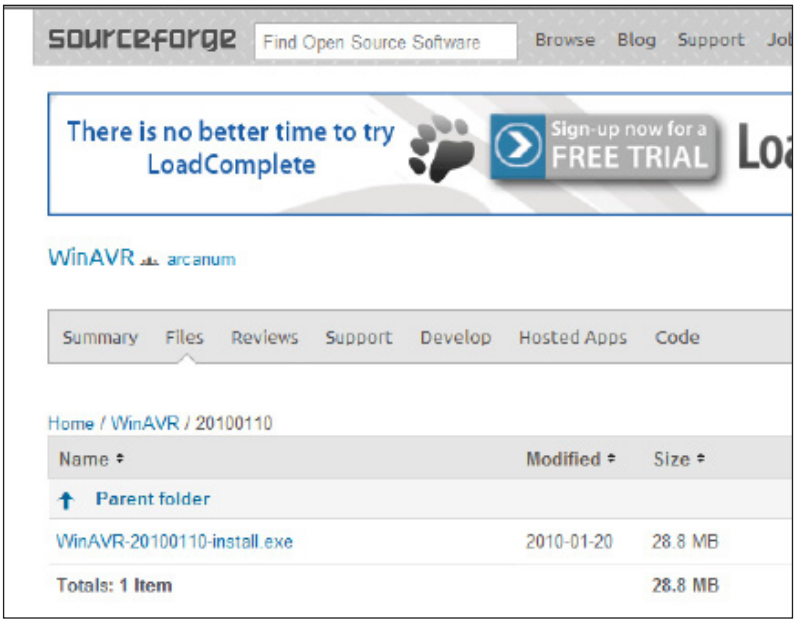
Enter URL in the address window.

http://sourceforge.net/projects/winavr/files/WinAVR/



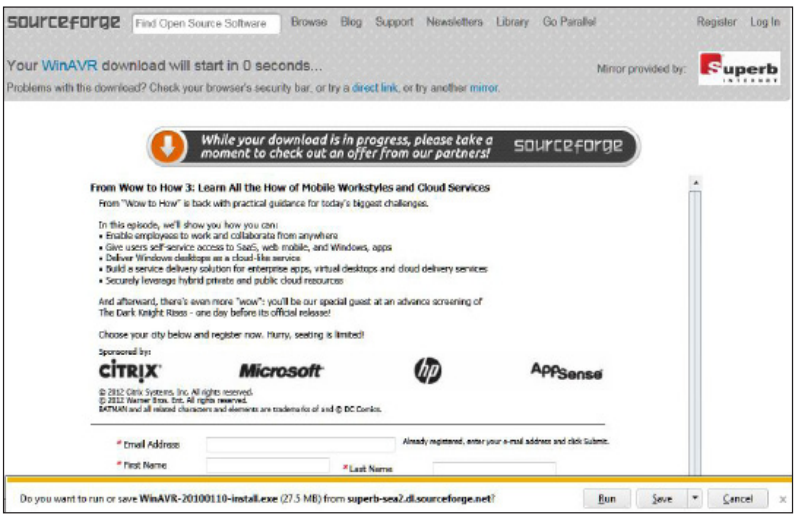
02 Download Link

Click check box next to 20100110.



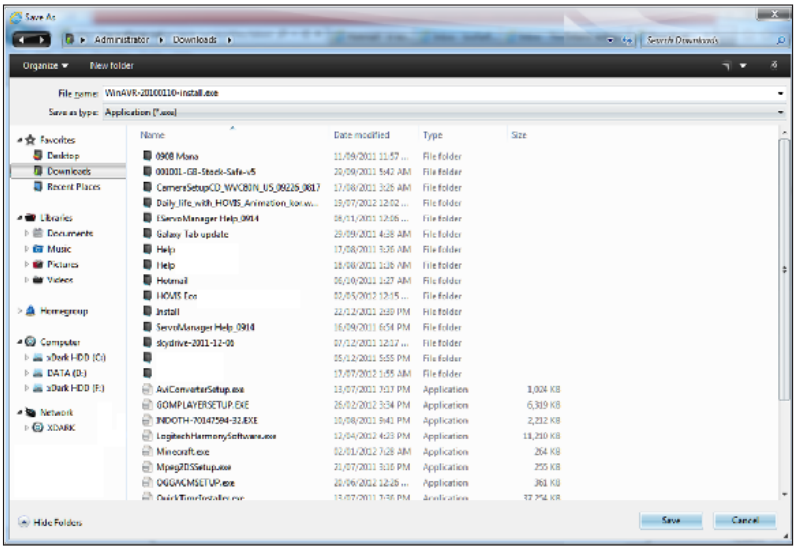
03 Download

Click check box next to WinAVT–20100110–install.exe.



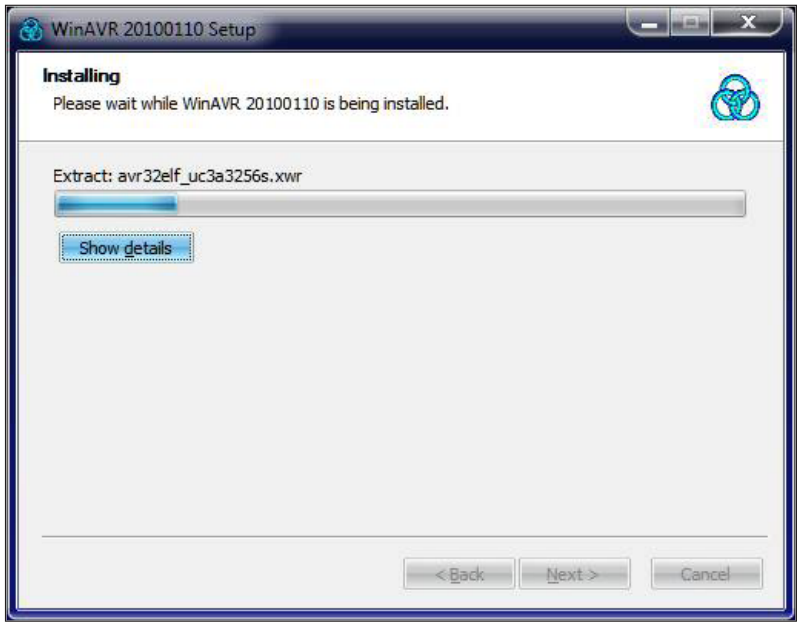
04 Save Installation File

Click “Save” button.



05 Select Destination Folder

Select the folder to save the installtion file and click “save” button.



06 Program Download

Click “Next” button.



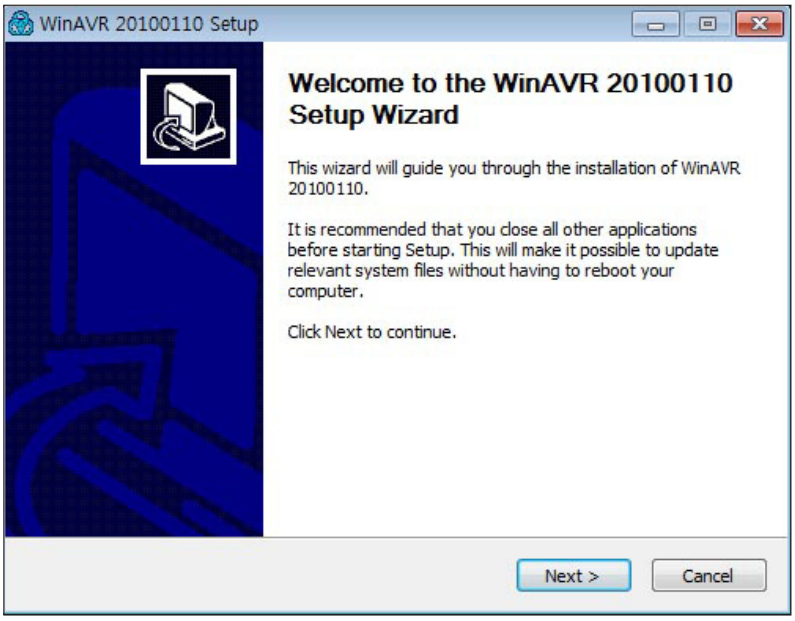
07 Run Installation Program

Double click newly created icon.



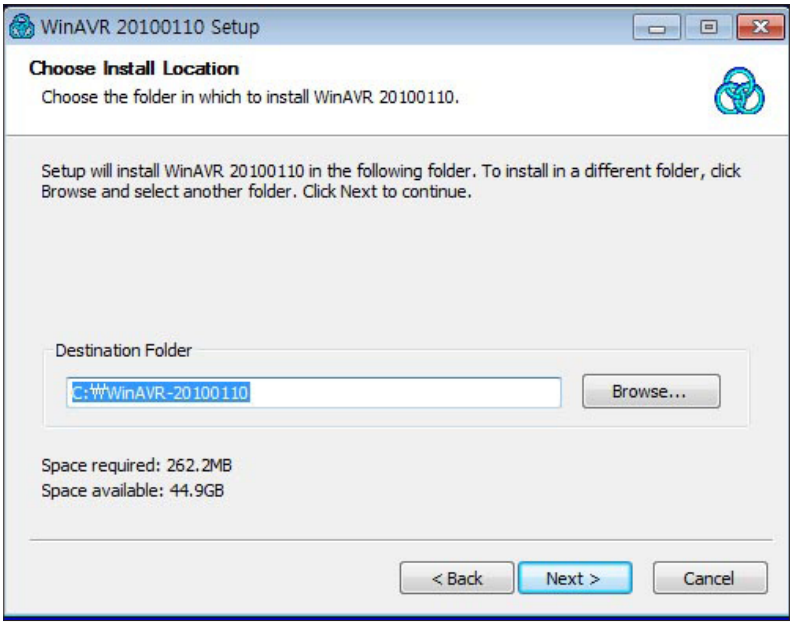
08 Confirm Installation

Select “English” and then click “OK”.



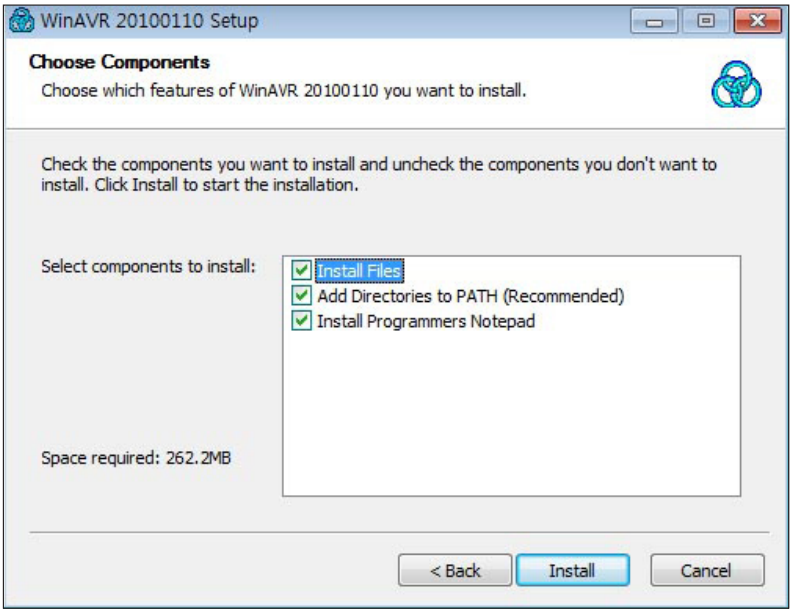
09 Start Setup Wizard

Running Setup Wizard. Click "NEXT".



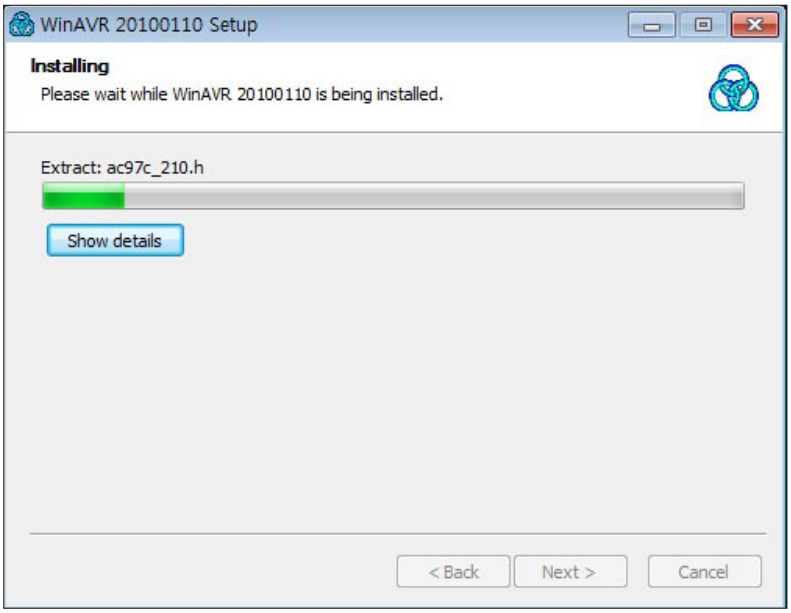
10 ConfirmDestinationFolder

Click "NEXT"



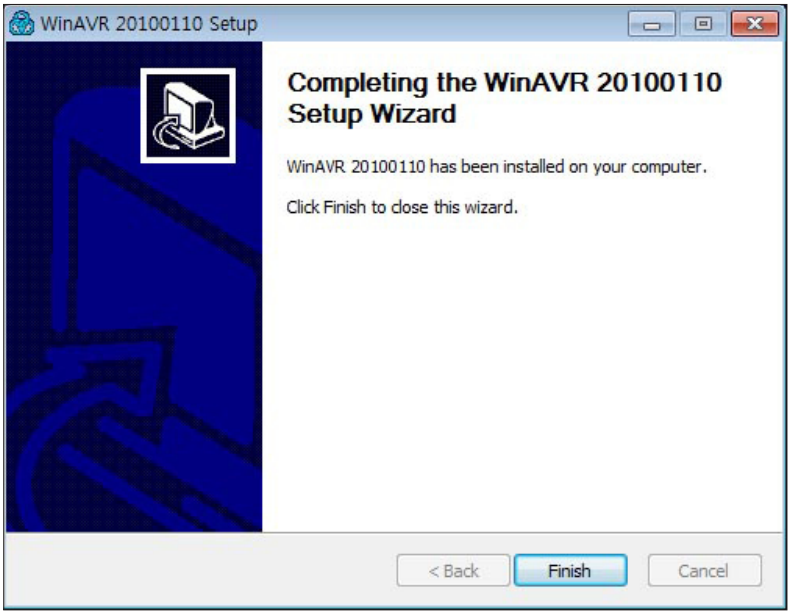
11 Select Components

Click "Install".



12 Starting Installation

Installation started.



13 Complete Installation

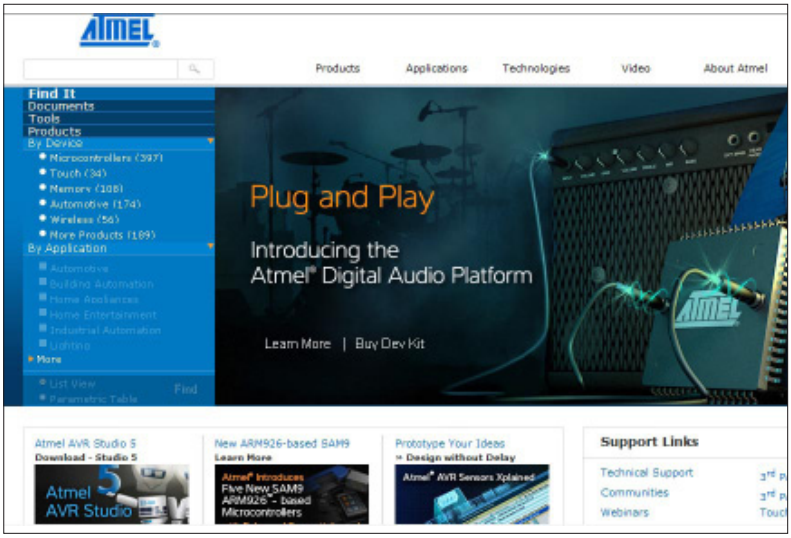
Click “Finish” button to complete installation.



AVR Studio Installation

01 Enter URL

Enter URL www.atmel.com in the address window.



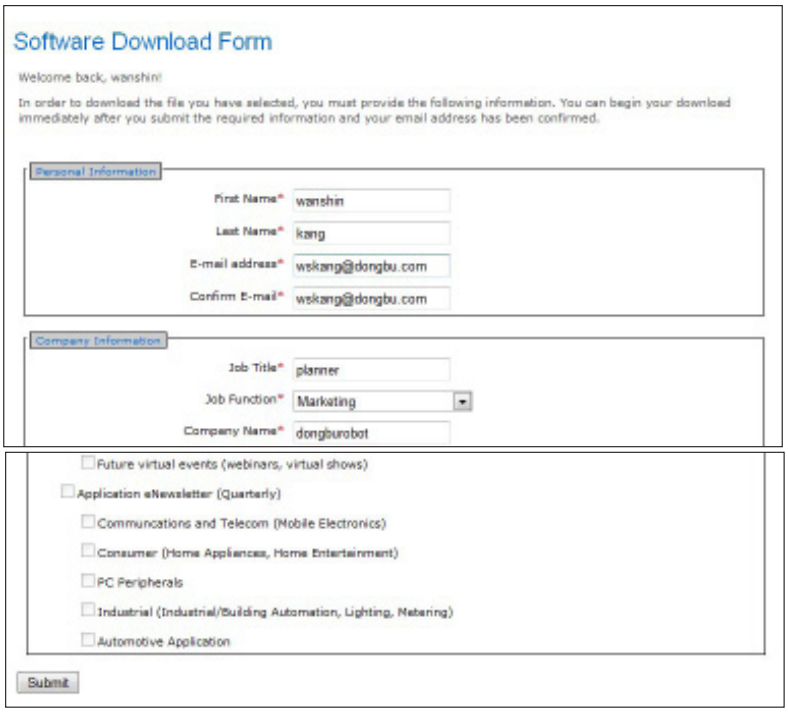
02 Download Link

Click 'DownloadAVRStudio5' in "Support Links".



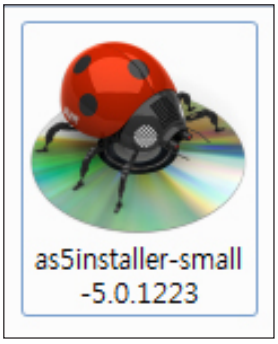
03 Select Software

Click "AVR Studio 5.1 Installer Full".



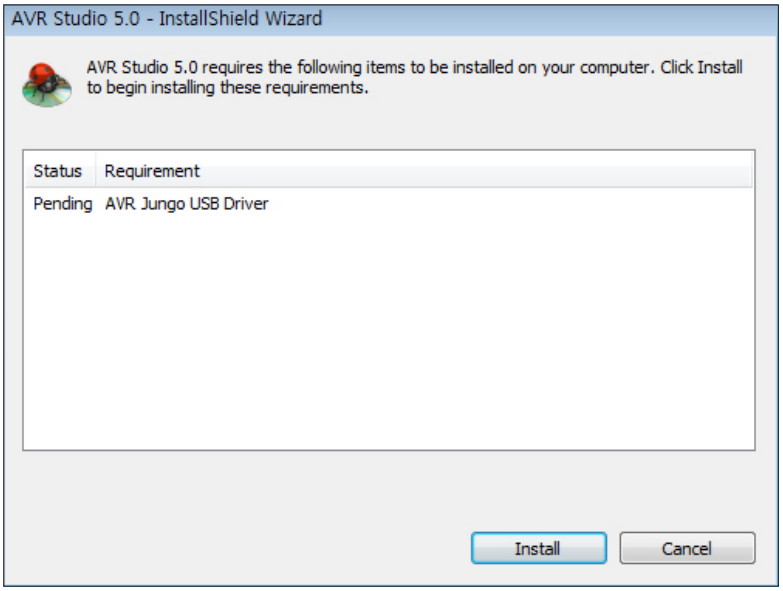
04 Software Download Form

Enter the the user information and click 'submit'.



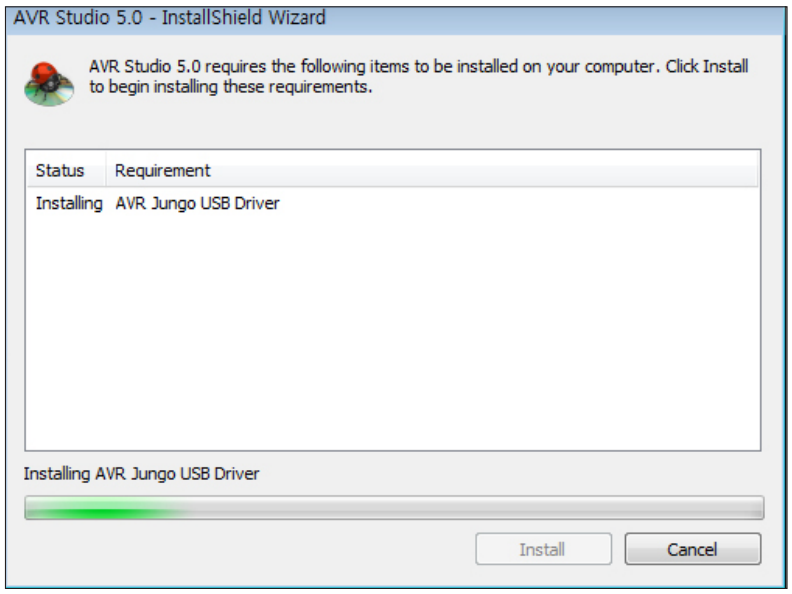
05 Run Installation program

Double click icon.



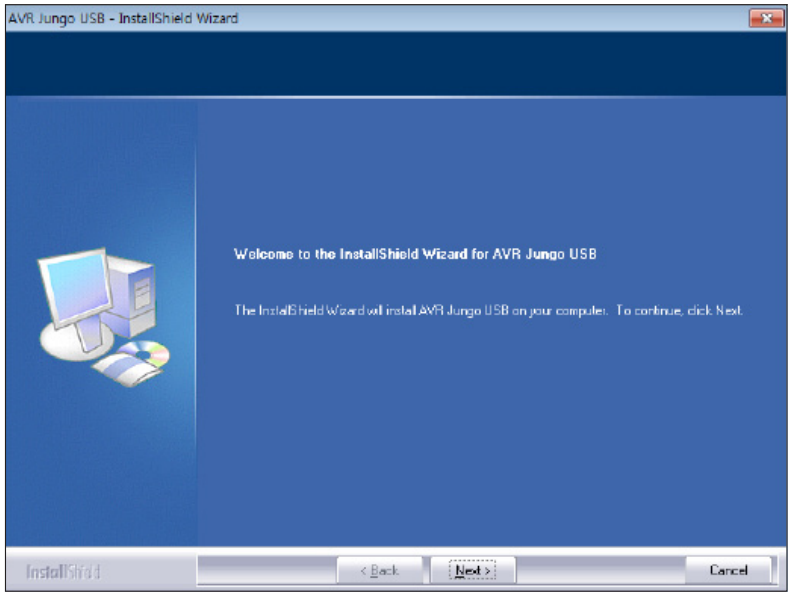
06 Start Installation

Click "Install" button.



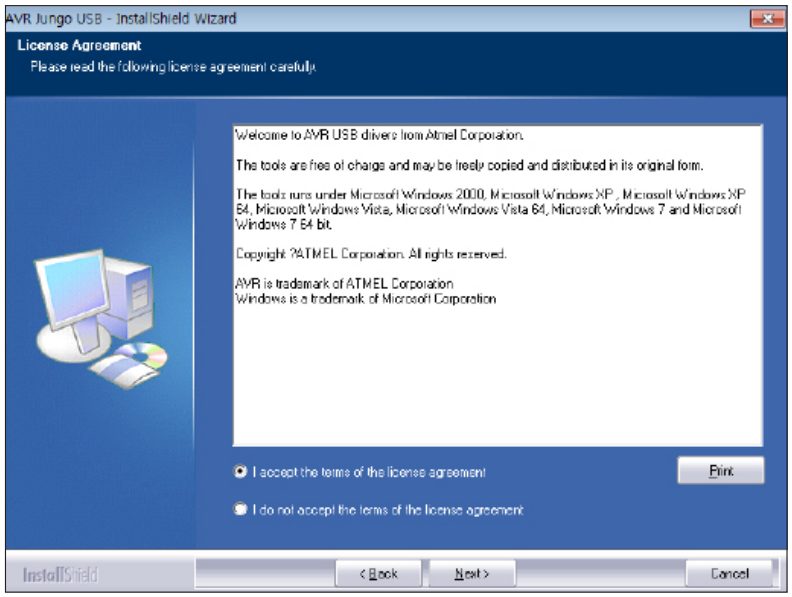
07 Program Installing

Program starts to install.



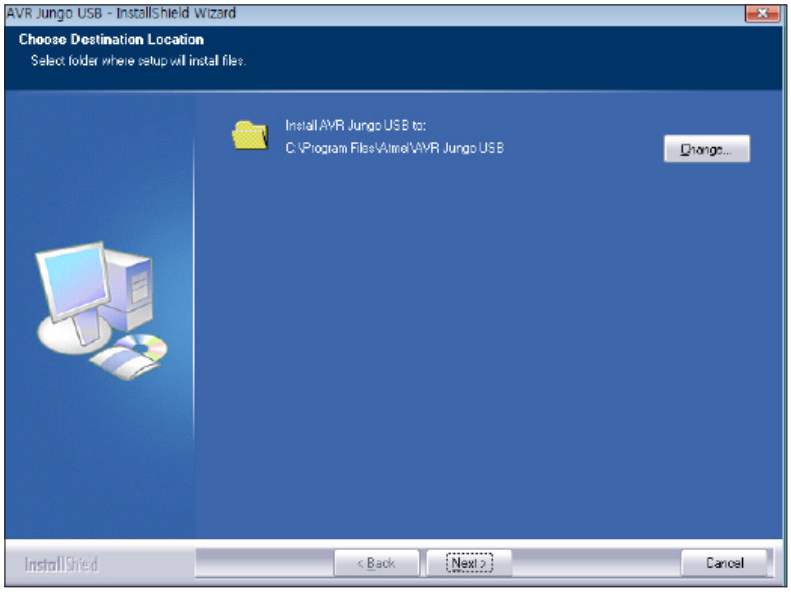
08 Confirm AVR Jungo USB installation

Click “NEXT”



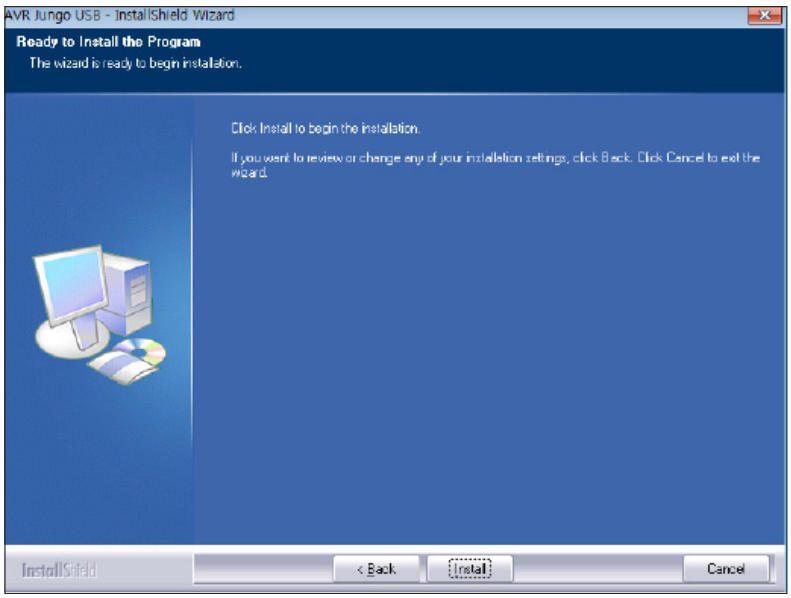
09 Accept License Agreement

Select “I accept” and click “Next” button



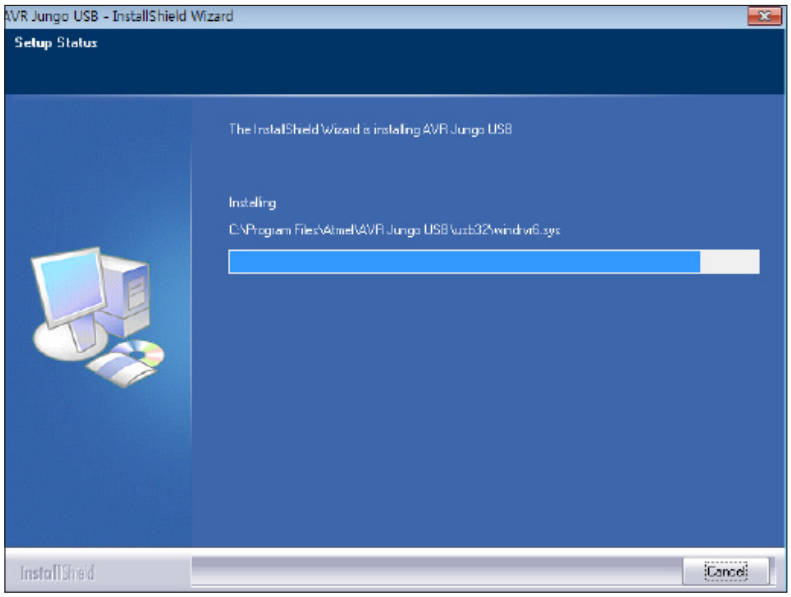
10 Select Destination Folder

Select prgram destination folder and click “Next”.



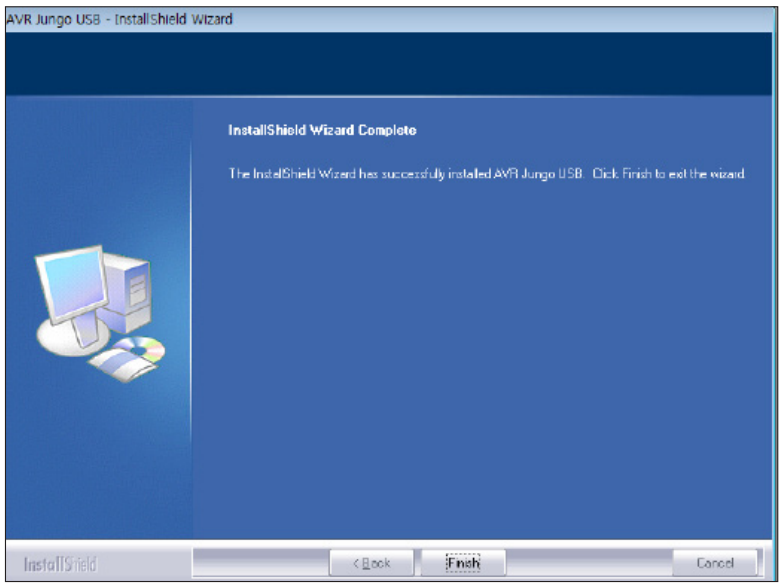
11 Start Installation

Click “Install” to start program installation.



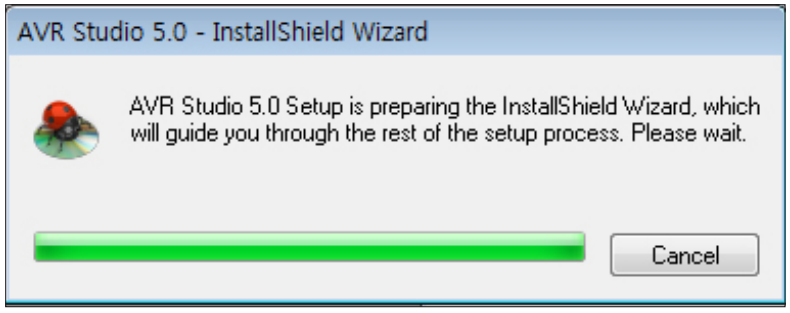
12 Program Installing

Program starts to install.



13 Complete Installation

Click “Finish” button to complete AVR Jungo USB installation.



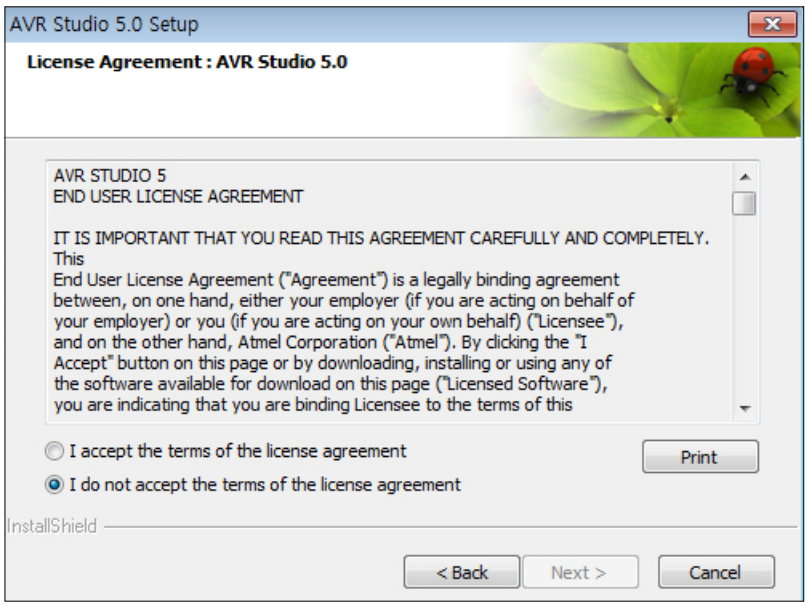
14 AVR Studio 5.0 Ready to Install

AVR Studio 5.0 ready to install.



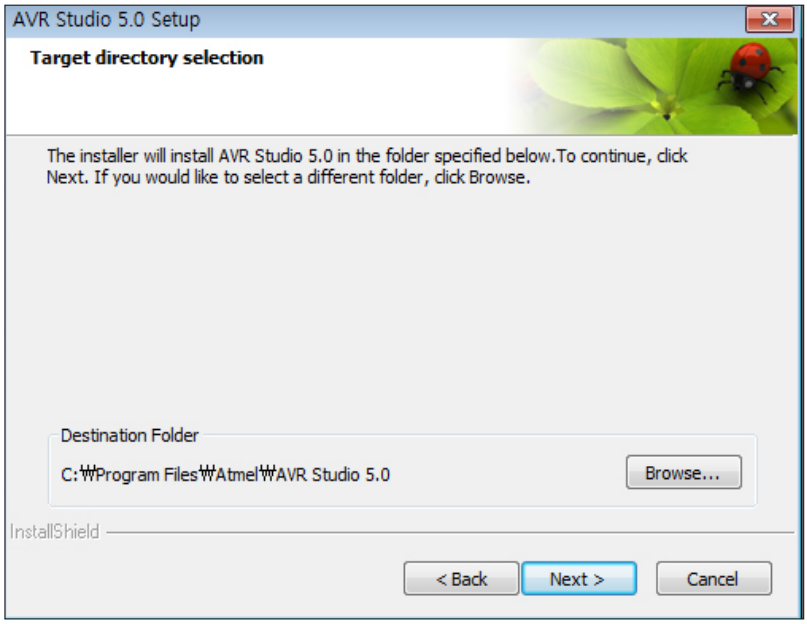
15 Start Setup Wizard

Click “Next”.



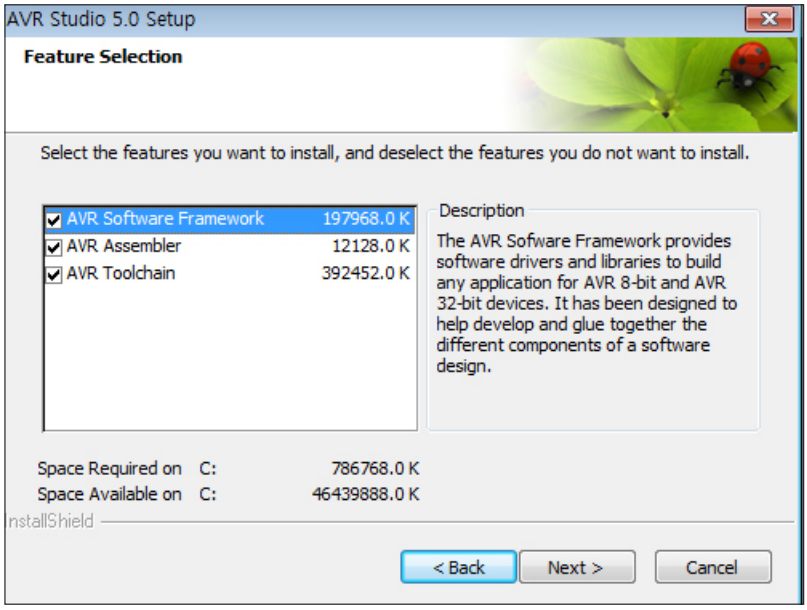
16 Accept License Agreement

Select "I accept the terms of the licese agreement" and click "Next".



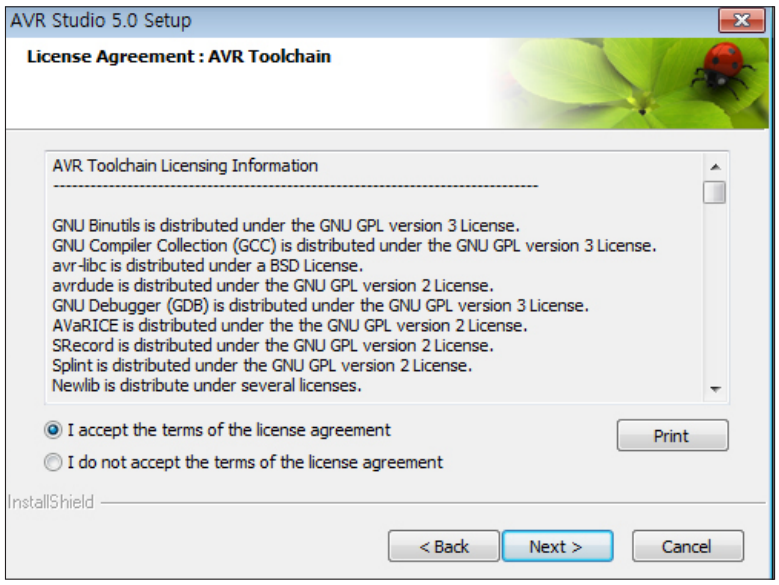
17 Confirm Destination Folder

Confirm destination folder and click "Next".



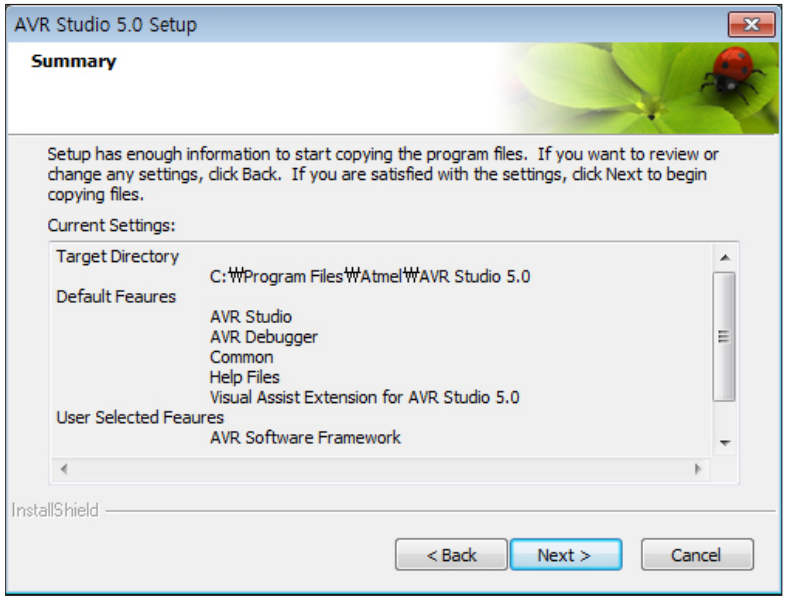
18 Feature Selection

Select features to install and click "Next".



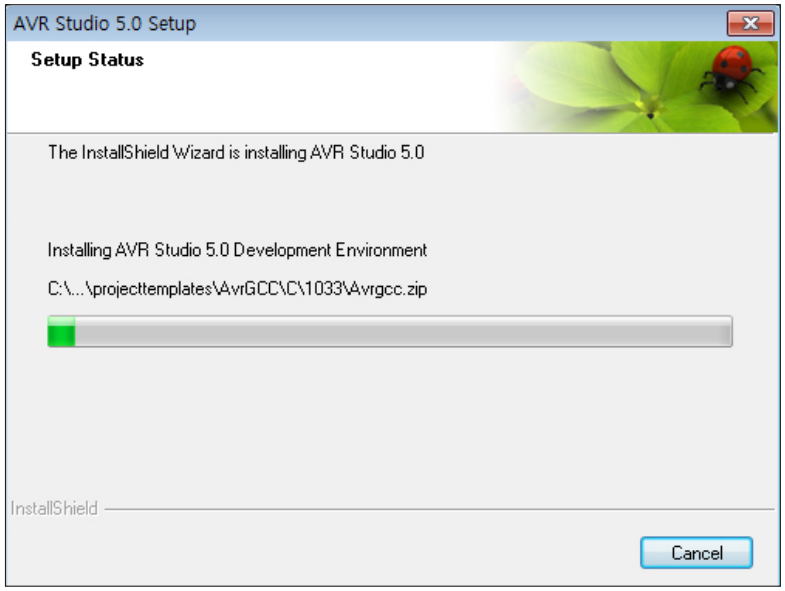
19 Acceipt License Agreement

Accept license agreement and click "Next".



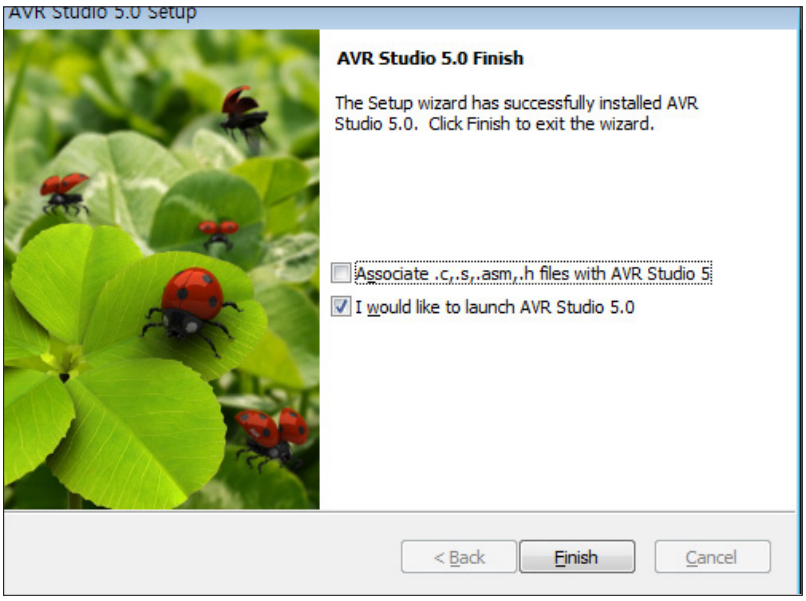
20 Check installation Details

Check installation details and click "Next".



21 Program Installing

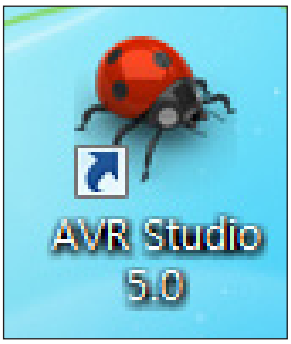
Program starts to install.



22 Installation Complete

Installation completed.

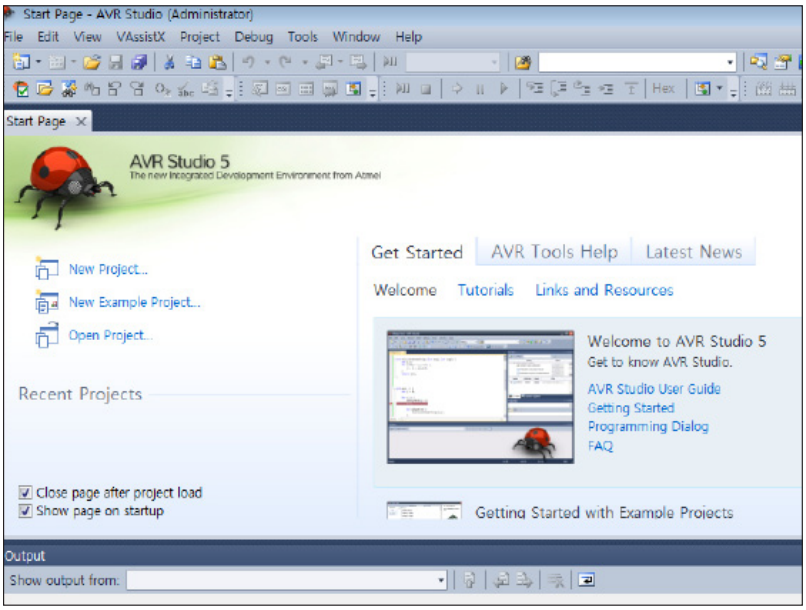
Check “I would like to launch AVR Studio 5.0” and click “Finish” to run the program.



New Project & Coding

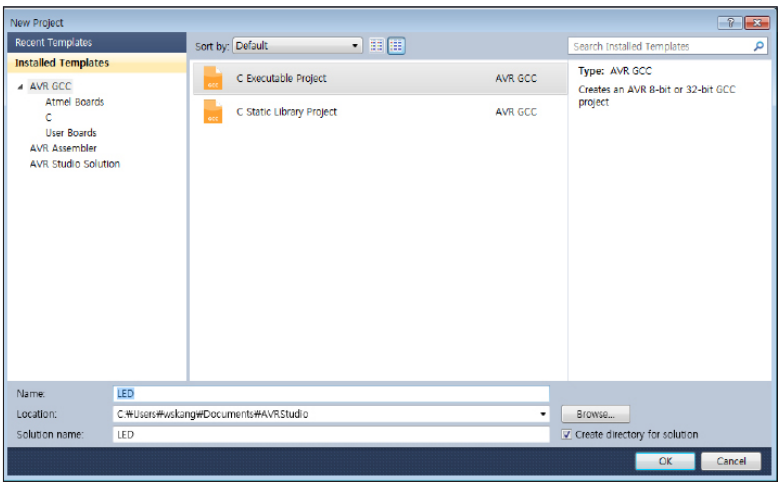
01 Run

Run AVR Stdio .



02 New Project

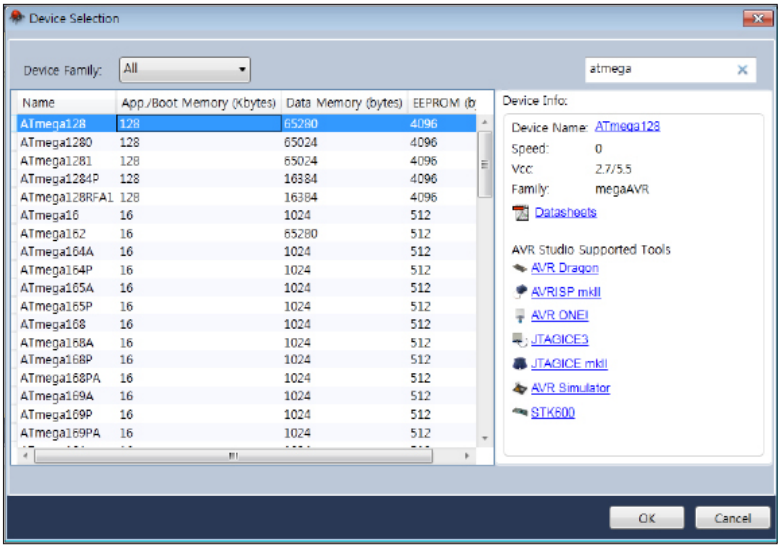
Click ‘New Project’.



03 Select GCC

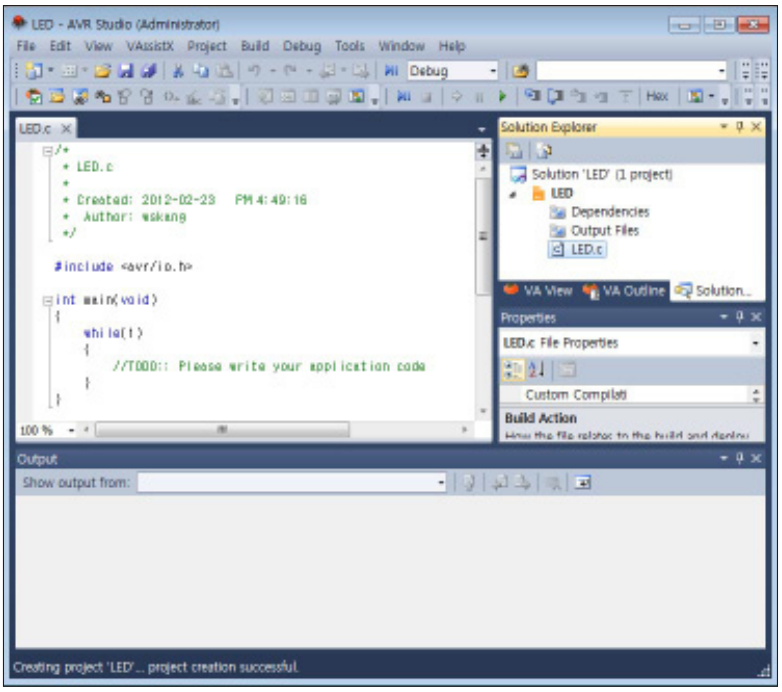
Select C Executable Project.

Enter “LED” as the name, Click ‘Create directory for solution’, Click ‘OK’ .



04 Select ATmega

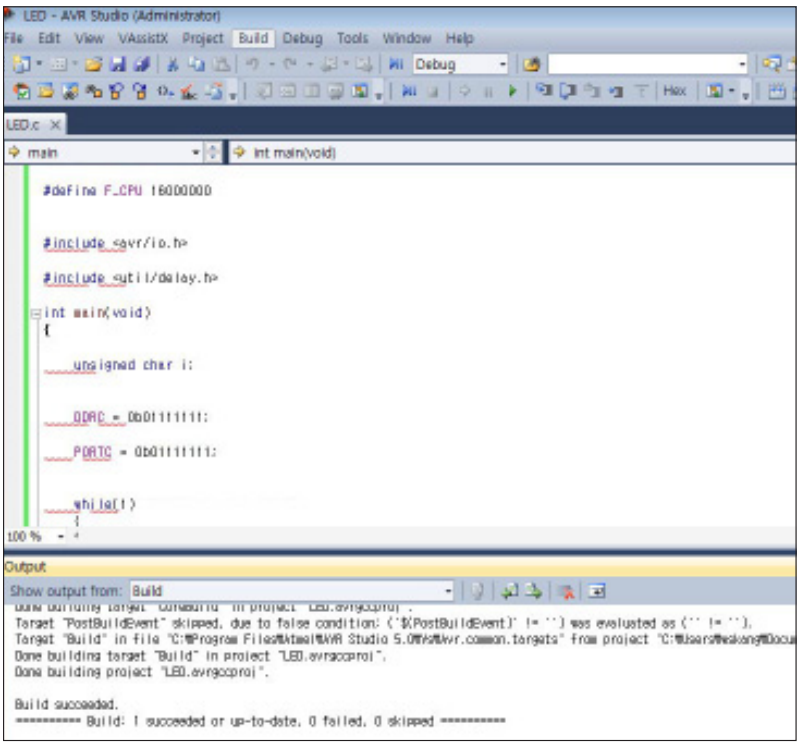
Search for atmega and select ATmega128 from the list.

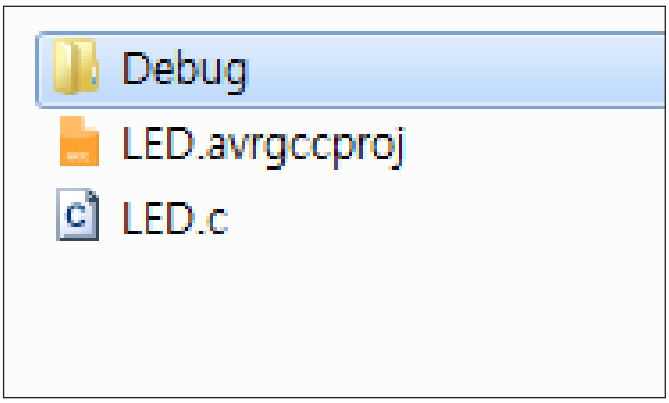


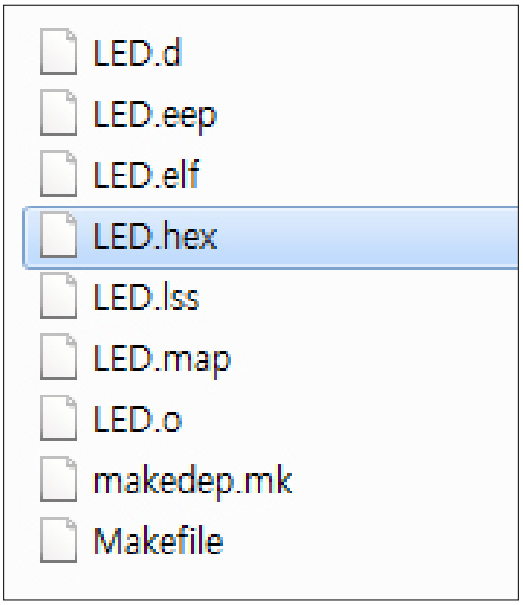
05 Editor

Start coding in the edit window.

Code example. LED example shown below.







Build & Run

01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by 1 succeded 0 failed comment.

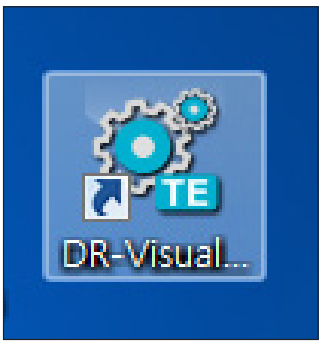
02 Debug Folder

After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



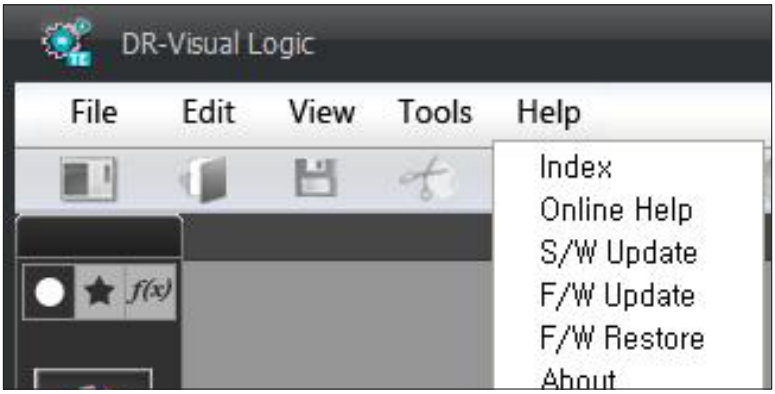
04 Visual Logic

Run DR-Visual Logic.



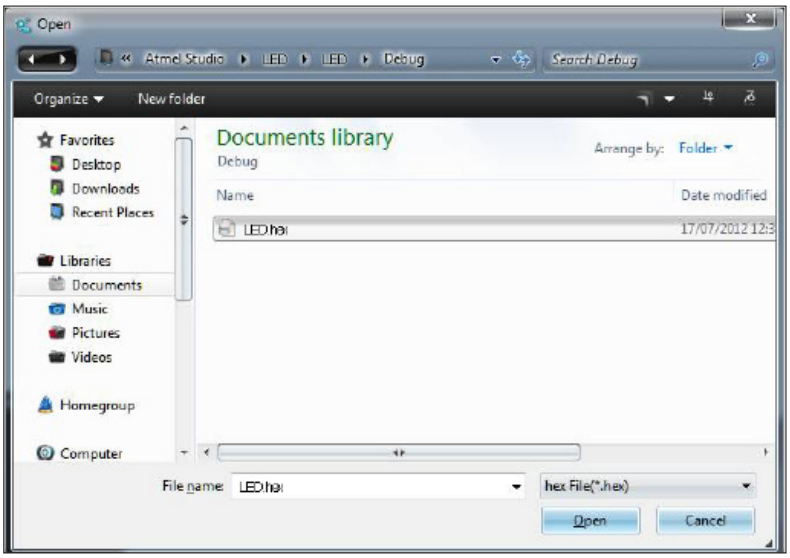
05 Connect

Select COM port and click Connect.



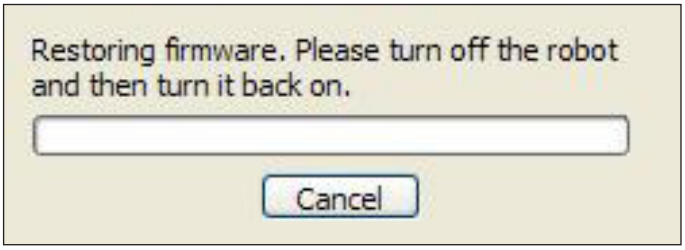
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



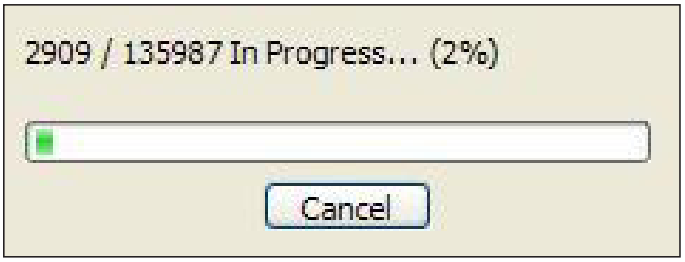
07 Load

Load Hex file from Debug folder



08 Firmware Restoration

Comment asking to restart the robot will appear. Turn off and on the controller power button.



09 Start Restoration

Firmware restoration started.



10 Run

LEDs will light up sequentially one by one from Servo LED to Power LED. After all 7 LEDs have been lit, LEDs will go off except for the Servo LED and sequence will start over again.

LED Control

Using ATmega128 digital output to control LED attached to DRC.

AVR Fundamentals

AVR has digital (I/O) ports installed which can be used to receive or send (HIGH & LOW) digital signal. As same pin can be configured to receive or send the signal, ports can be used for variety of purposes such as LED control, button input, and digital data transmission. There are total of 7 ports from A to G installed in the ATmega128, A–F ports have 8 pins and G port has 5 pins for total of 53 pins which can be freely used for digital I/O. In addition to regular I/O functhon, each of these 53 pins have specal functions which can be selected for use by register setting. Special functions will be discussed more in detail as the lesson progresses.

Registers related to digital I/O are as shown below. ATmega128 registers are composed of 8 bits (1byte). X in DDRX and PROTX refers to port name (A~G).

■ DDRX : PORT X data direction register

비트	7	6	5	4	3	2	1	0
	DDX7	DDX6	DDX5	DDX4	DDX3	DDX2	DDX1	DDX0
Read/ write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ PORTX : PORT X data register

비트	7	6	5	4	3	2	1	0
	PORTX7	PORTX6	PORTX5	PORTX4	PORTX3	PORTX2	PORTX1	PORTX0
Read/ write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

DDRX resgister determines the I/O direction of port X. If the register bit value is 00, port is set to input and if bit value is 1, port is set to output. PORTX register is used to save the logical value to be sent to Port X. When DDXn value is set to 1(output), value in PORTXn is sent to n pin of Port X.

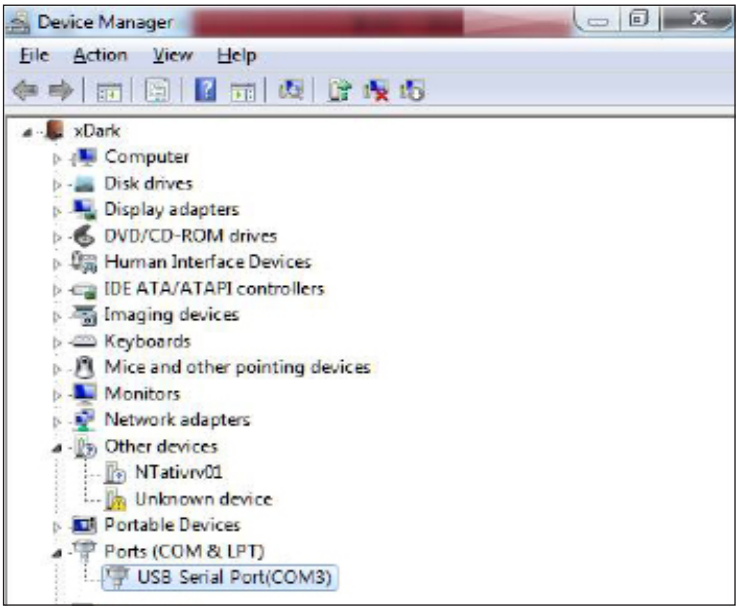
H/W Fundamentals

Diodes have two poles and have special characteristic that allow electric current to flow in forward direction while blocking current flow in opposite direction. LED is an abbreviated term for Light Emitting Diode, which lights up when electric current flows in forward direction(picture 00) and turns off when current flows in opposite direction(picture 00). ATmega128 circuit for LED control allows Active Low(photo 00) and Active High(picture 00) type of control. In Active Low type control, electric current flows in forward diirection and LED lights up when pin outpu is Low.In Active Hight type control, LED lights up when pin output is High. DRC uses pin 0–6 of PORT C to control the LED and the circuit is designed for Active Low type control (picture 00). LED corresponding to the pin number is shwon in the table below.

Pin Number	7	6	5	4	3	2	1	0
LED	–	Power	Spare	RX	TX	EXEC	Program	Servo

Preparation





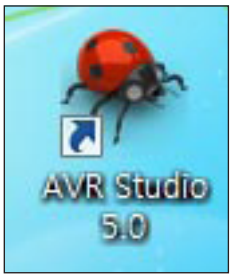
Hardware Preparation

01 Connection

Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.

02 COM port connection

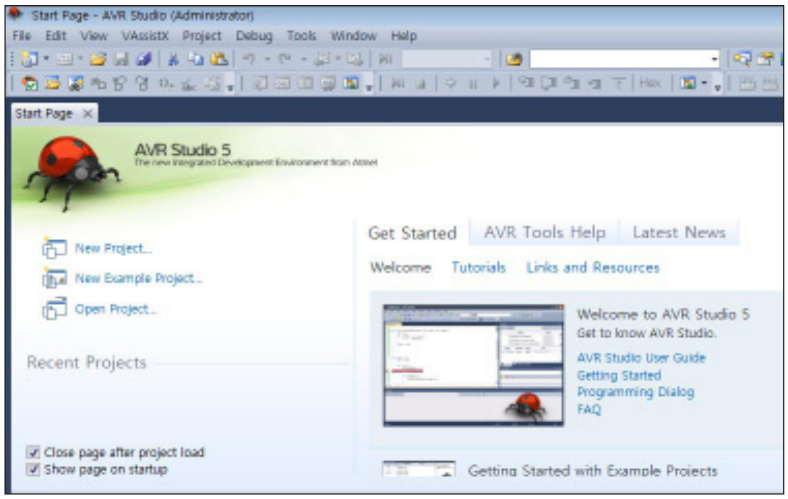
My Computer 〉Property 〉Device Manager. Check COM connection.



Running AVR Studio and Coding Preparation

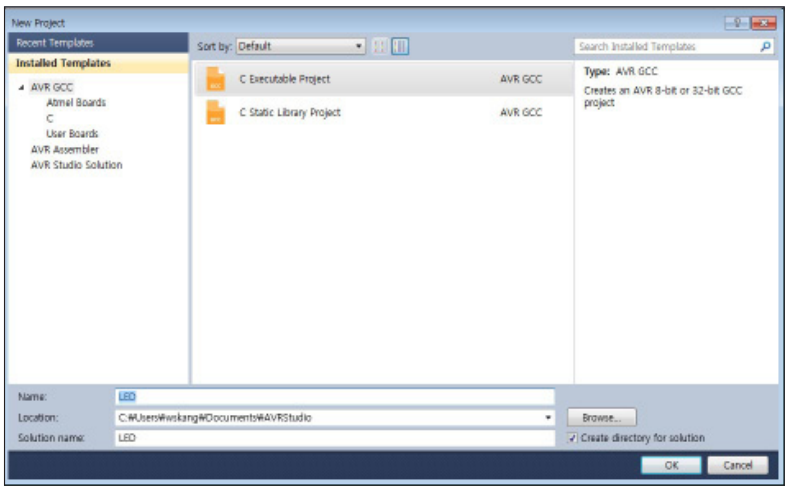
01 Run

Run AVR Stdio.



02 New Project

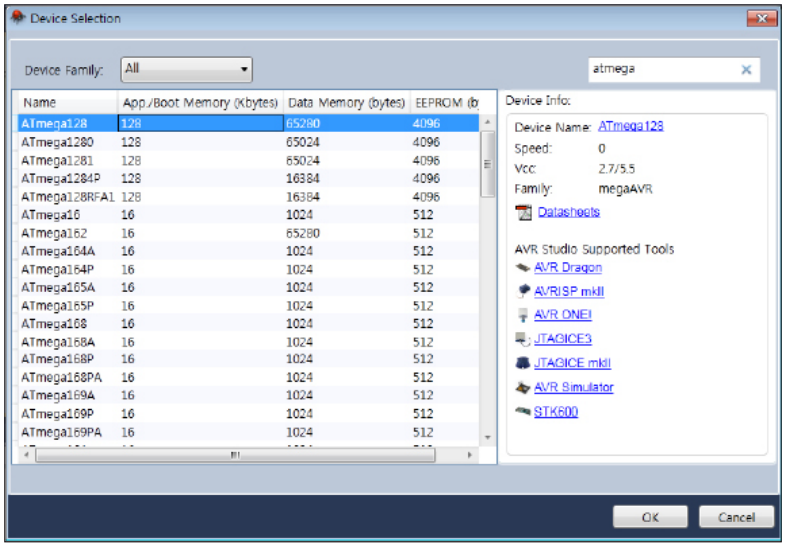
Click New Project.



03 Select GCC

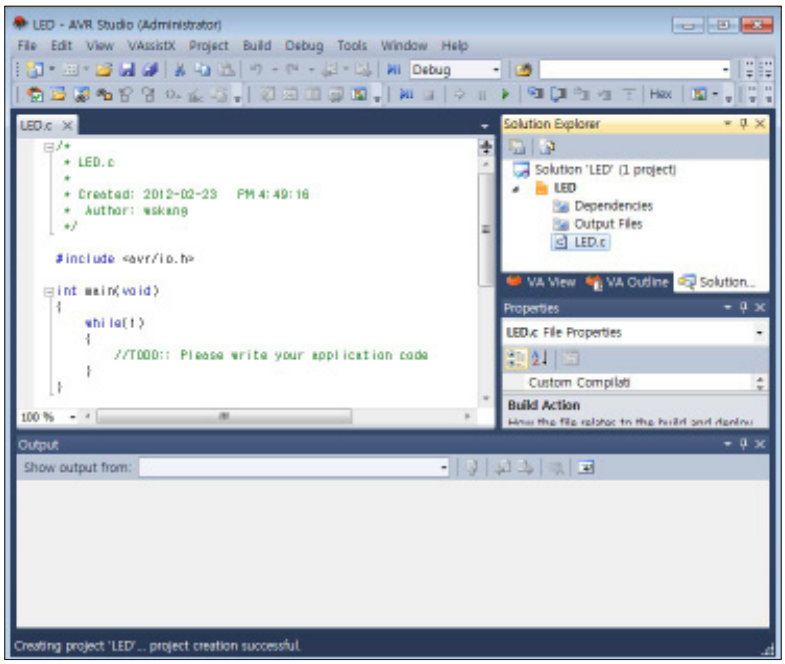
Select C Executable Project

Input 'LED' as Name, Click Create directory for solution, Press OK button.



04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



05 Editor

Start coding in the edit window.
Follow the example below.

Example

This example will repeatdly turn on the DRC LEDs in sequence from Servo(0 pin) to Power(6pin).

```
//CPU clock  setup
#define F_CPU 16000000

//ATmega128 register defined
#include <avr/io.h>
//_delay_ms delay function defined
#include <util/delay.h>

int main(void)
{
        //Declare 8bit  variable
        unsigned char i;

        //Set port C I/O direction
        DDRC = 0b01111111;
        //Set port C output value
        PORTC = 0b01111111;

        // Continuously repeat content of while statement
        while(1)
        {
                // Repeatdly turn on LED one by one
                for(i=0b00000001;i<=0b01111111;i=(i<<1)+1){
                        PORTC = ~i;             //Active Low, output reversed  value of  variable i
                        _delay_ms(200);                 // delay 200ms
                }
        }

        //Return integer value to fit format of the function
        return 1;
```

Example description

```
#define F_CPU 16000000
```

#define statement is one of the commands prior to coding. Command replaces any F_CPU value found with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on clock speed, this command sets the clock speed so that delay function and other functions execute properly.

```
#include <avr/io.h>
#include <util/delay.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. util/delay.h header file has _delay_ms() and other commonly used delay functions defined. Use #include statement to include the functions.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Set port C 0–6 pins to output, set output value to 1. This will turn off all LED. 0b01111111 shows binary format in C language.

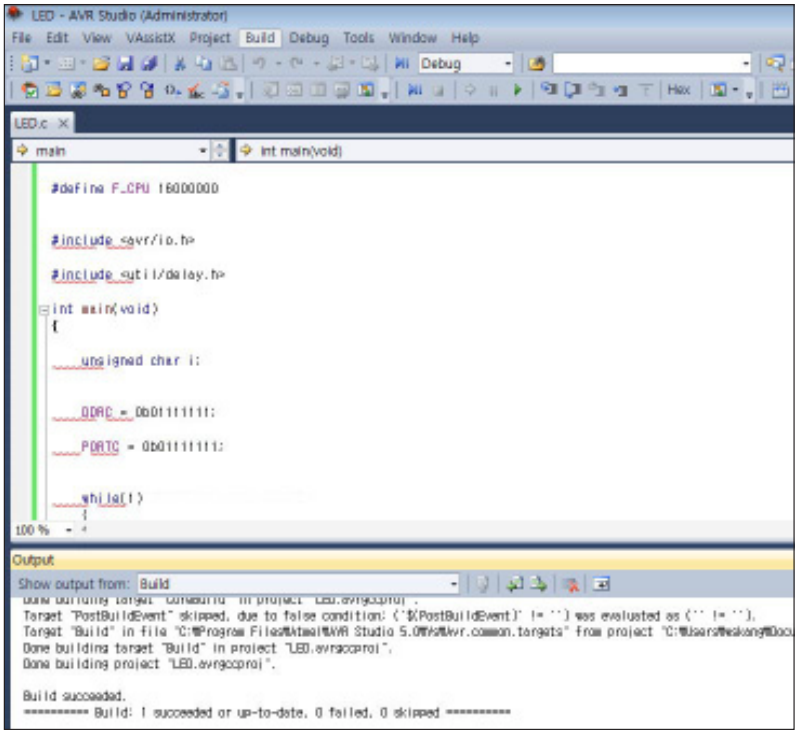
```
for(i=0b00000001;i<=0b01111111;i=(i<<1)+1){
        PORTC = ~i;
        _delay_ms(200);
```

i initializes to 0b00000001 within the for statement. i changes to 0b11111110 when ~(bitwise not) operation is applied. When 0b11111110 is entered as PCORTC value, LED with 0 bit number lights up. After waiting 200ms by _delay_ms() function,

i is updated to i=(i<<1)+1. This operations shifts i value to the left by 1 bit and adds 1 to the result. For example 0b00001111 becomes 0b00011110 when it is shifted to the left and then it becomes 0x00011111 when 1 is added. LED lights up sequentially starting from 0 bit numbered LED when FOR statement is executed repeatedly. FOR statement will continue to run until i<=0x01111111 condition is satisfied, FOR statement will end after all LEDs are lit As this FOR statement exists within the while(1) statement, FOR statement will start again from the beginning and LEDs will light up sequentially starting from the 0 bit numbered LED.

Program Execution

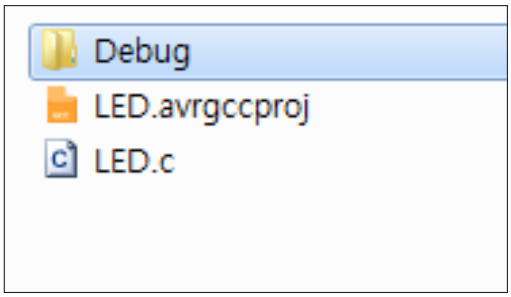
LEDs will light up sequentially one by one from Servo LED to Power LED when program is loaded and executed. After all 7 LEDs have been lit LEDs will go off except for the Servo LED and sequence will start over again.



01 Build

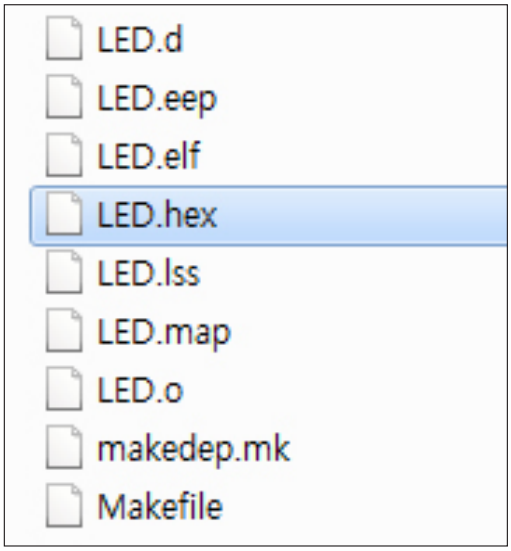
Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by 1 succeded 0 failed comment.



02 Debug Folder

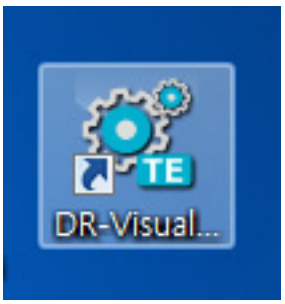
After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.



03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



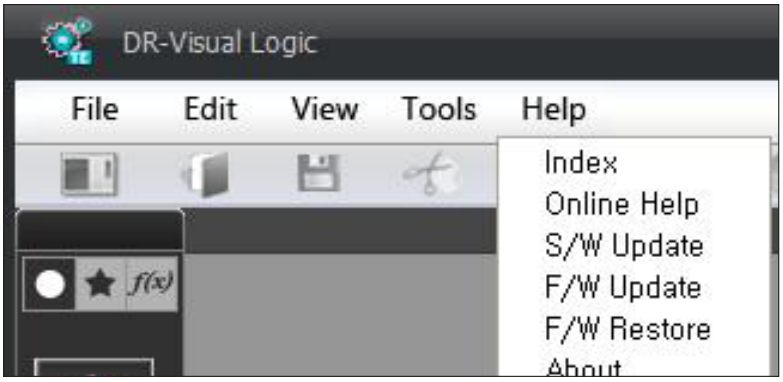
04 Visual Logic

Run DR-Visual Logic .



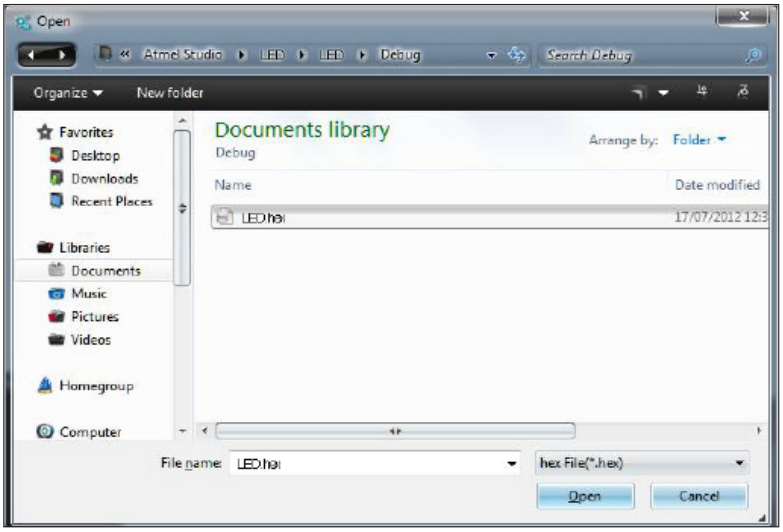
05 Connect

Select COM port and click Connect.



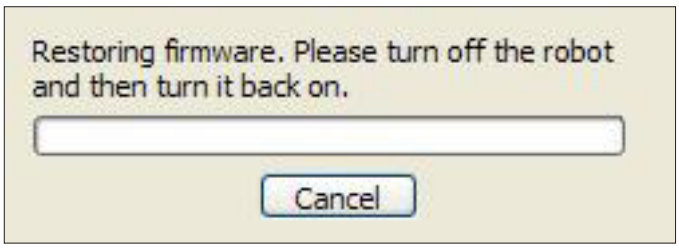
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



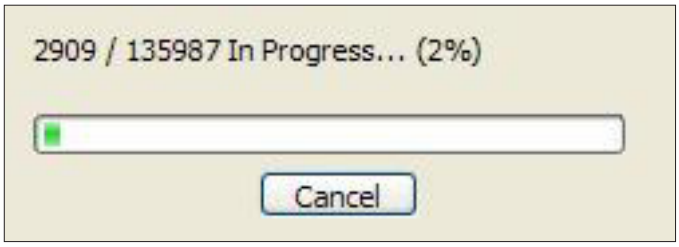
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.



10 Run

LEDs will light up sequentially one by one from Servo LED to Power LED. After all 7 LEDs have been lit, LEDs will go off except for the Servo LED and sequence will start over again.

LED Button Control

Using ATmega128 digital I/O function to receive signal from the DRC button to control the LED.

AVR Fundamentals

Chapter 1 showed using the output function of the digital I/O port to control the LED. This chapter will show how to control the LED using the results of an input from the DRC button. Registers related to digital I/O are as shown below. ATmega128 registers are composed of 8 bits (1byte). X in DDRX and PROTX refers to port names (A~G).

■ DDRX : PORT X data direction register

Bit	7	6	5	4	3	2	1	0
	DDX7	DDX6	DDX5	DDX4	DDX3	DDX2	DDX1	DDX0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ PORTX : PORT X data register

Bit	7	6	5	4	3	2	1	0
	PORTX7	PORTX6	PORTX5	PORTX4	PORTX3	PORTX2	PORTX1	PORTX0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ PINX : PORT X input pin address

Bit	7	6	5	4	3	2	1	0
	PINX7	PINX6	PINX5	PINX4	PINX3	PINX2	PINX1	PINX0
Read/ Write	R	R	R	R	R	R	R	R
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

As explained in Chapter 1, DDRX register determines the I/O direction of the Port X and PORTX register is used to save the logical value to be output to the Port X. PINX register is an address where logical value of the current port can be read. Logical value of the Port X can be read from PINX regardless of the I/O state. PINX is read only, values cannot be written to PINX register.

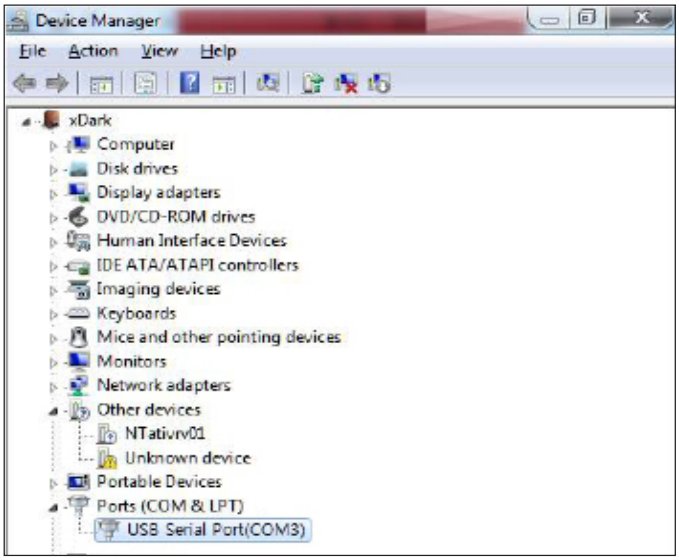
H/W Fundamentals

Tact switch is a type of push switch with the name derived from the word 'tactile' because of the tactile feedback it provides. Two tact switch pins are normally apart untill the switch is pressed to connect the pins together to let the current flow. Tact switches may have either 2 or 4 pins but there is no difference in function as 4 pins are connected in pairs internally. Circuit diagram below (Diagram 00) shows the tact switch connection to ATmega128. When the switch is not pressed, voltage (VCC) flows into the digital input pin as condenser C becomes charged through the Resistor R (Diagram 00). Therefore input value is HIGH when switch is not pressed. When the switch is pressed, digital input pin becomes connected directly to the GND and Condenser C becomes discharged (Diagram 00). Input value is LOW as GND voltage flows into the digital input pin. If Resistor R is absent from the circuit and switch is not pressed, input pin would be in floating state (neither HIGH or LOW) without connection to any voltage source. Resistor R connects VCC to input pin which pulls up the input pin voltage value to HIGH. Resistor R in this type of circuit is called Pull-Up Resistor. DRC Port A pin numbers 0-5 are used for button input. LED names corresponding to each pin number are shown below.

Pin Number	7	6	5	4	3	2	1	0
LED Name	–	–	Right	Left	Down	Up	OK	Mode

Preparation





Preparing the Hardware

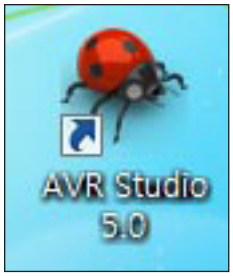
01 Connection

Plug in the DRC power adapter and connect the PC to DRC using USB to Serial Cable.

02 COM port connection

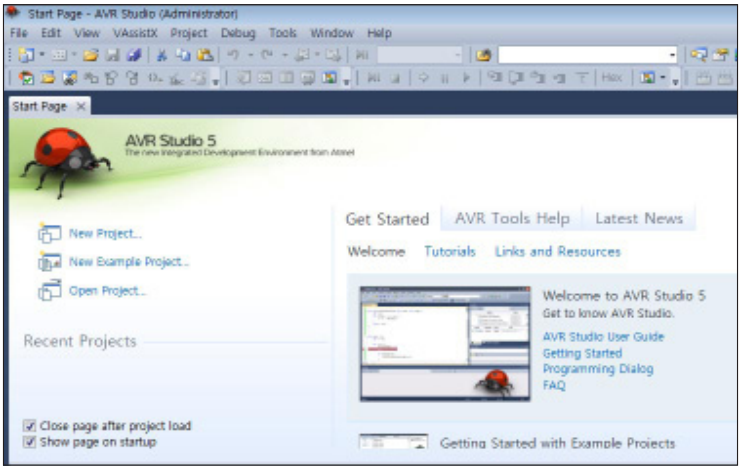
My Computer〉Property〉Device Manager. Check COM connection.

Running AVR Studio and Coding Preparation



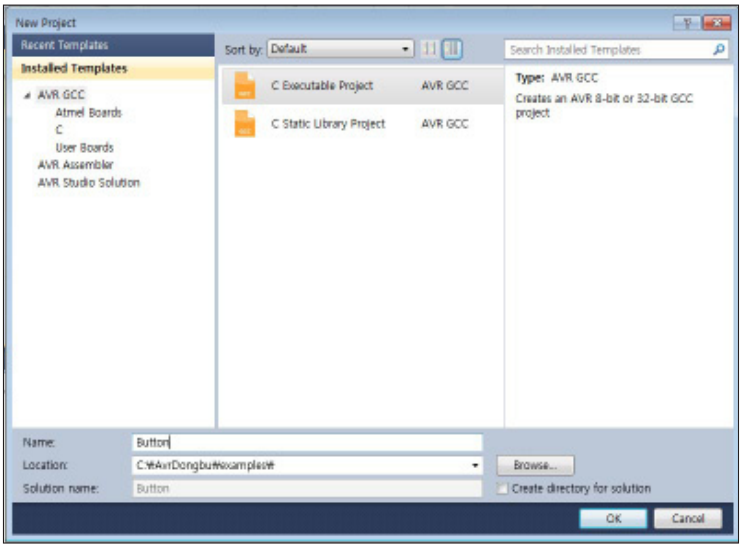
01 Run

Run AVR Stdio.



02 New Project

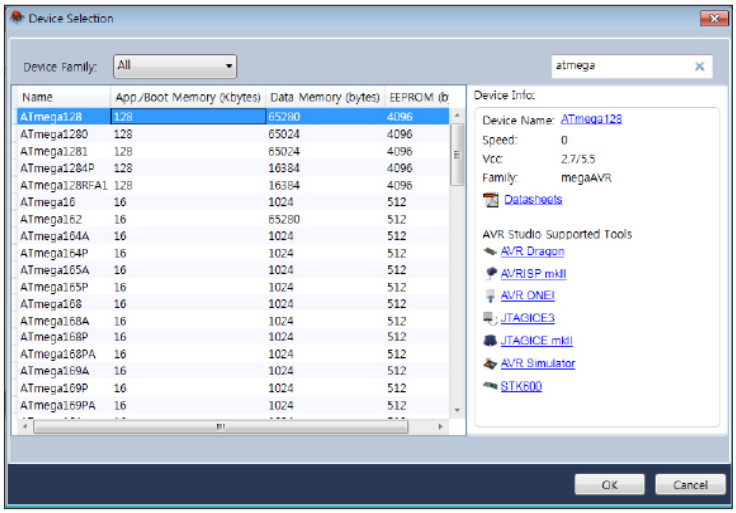
Click New Project.



03 Select GCC

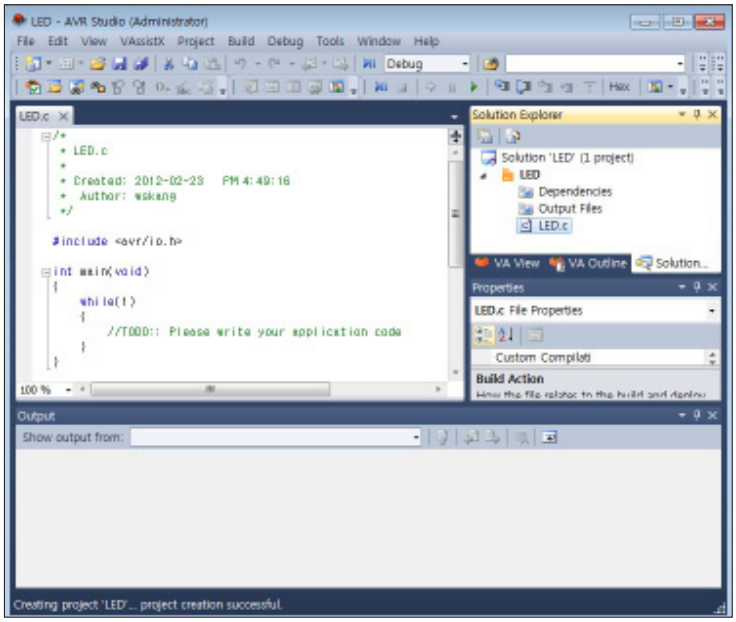
Select C Executable Project .

Input “Button” as the Name, Click Create directory for solution and then press OK button.



04 Select ATmega

Search for atmega. Select Atmega128 from the list.



05 Editor

Start coding in the edit window. Follow the example below.

Example

This example will turn on the correspoding LED each time DRC button is pressed. Table below shows the LED corresponding to each button.

Button	LED	Button	LED	Button	LED
Mode	Program	Up	TX	Left	Servo
OK	EXEC	Down	RX	Right	Power

```
//ATmega128 registers defined
#include <avr/io.h>

//Define Port C pins correspoding to LED names
#define LED_SERVO                0b00000001
#define LED_PROGRAM              0b00000010
#define LED_EXEC                 0b00000100
#define LED_TX                   0b00001000
#define LED_RX                   0b00010000
#define LED_SPARE                0b00100000
#define LED_POWER                0b01000000
#define LED_MASK                 0b01111111

//Define Port A prins corresponding to BTN names
#define BTN_MODE                 0b00000001
#define BTN_OK                   0b00000010
#define BTN_UP                   0b00000100
#define BTN_DOWN                 0b00001000
#define BTN_LEFT                 0b00010000
#define BTN_RIGHT                0b00100000
#define BTN_MASK                 0b00111111

int main(void)
{
```

```
//Define 8 bit variable
        unsigned char ucButton, ucLED;

        //Set Port A I/O direction
        DDRA = 0b00000000;

        //Set Port C C I/O direction
        DDRC = 0b01111111;
        //Set port C output value
        PORTC = 0b01111111;

        //Conitinuously repat content of while statement
        while(1)
        {
                //Read button value from Port A
                ucButton = (~PINA) & BTN_MASK;
                //Initialize ucLED to all LED off state
                ucLED = 0;

                //Turn on PROGRAM LED when MODE Button is pressed
                if(ucButton & BTN_MODE){
                        ucLED |= LED_PROGRAM;
                }
                //Turn on EXEC LED when OK Button is pressed
                if(ucButton & BTN_OK){
                        ucLED |= LED_EXEC;
                }
                //Turn on TX LED when UP button is pressed
                if(ucButton & BTN_UP){
                        ucLED |= LED_TX;
                }
                //Turn on RX LED when DOWN is pressed
                if(ucButton & BTN_DOWN){
                        ucLED |= LED_RX;
```

```
    }
                //Turn on SERVO LED when LEFT Button is pressed
                if(ucButton & BTN_LEFT){
                        ucLED |= LED_SERVO;
                }
                //Turn on POWER LED when RIGHT Button is pressed
                if(ucButton & BTN_RIGHT){
                        ucLED |= LED_POWER;
                }

                //Change  PORTC value to equal ucLED value
                PORTC = (~ucLED) & LED_MASK;
        }

    //Return integer value to fit format of the function
        return 1;
    }
```

Example Description

```
#include <avr/io.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file.Use #include statement to include the functions.

```
#define LED_SERVO                    0b00000001
...
#define BTN_MASK                     0b00111111
```

#define statement is one of the functions that is run prior to code compiling , #define LED_SERVO 0b00000001 will first replace LED_SERVO with 0b00000001 before the codes are compiled. #define statement is used to faciliate program-ming by providing the way to use rcognizable names such as LED_SERVO rather than using unfamiliar numbers like 0b00000001 every time Button and LED is used in programming.

```
unsigned char ucButton, ucLED;
```

Declare 8bit variables ucButton and ucLED. These two variables are used to show the current state of the button and LED. ubButton and ucLED values within the while statement are reverse of the actual port input or output. ucButton value is reverse of PINA with pressed button value being 1 and released value being 0. ucLED has reversed value of PORTC value as well with LED on value being 1 and off value being 0.

```
DDRA = 0b00000000;
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Set Port A pin numbers 0~5 as input. Set Port C pin numbers 0~6 as output. Set output value as 1 and all LEDs will be turned off.

```
ucButton = (~PINA) & BTN_MASK;
ucLED = 0;
```

While statement starts by reading the button value into ucButton. Button value read from PINA is 0 if button is pressed and 1 if the button is released, ~ operation is applied to revese the read value. Apply &(bitwise AND) operation to BTN_MASK(0b00111111) to change the 6th and 7th bit values to 0 (6th & 7th bits are not related to button). ucButton will have value of 0b00XXXXXX wth Xs containing state of the buttons corresponding to each bit. ucLED will be initialized to 0 with all LEDs off.

```
if(ucButton & BTN_MODE){
        ucLED |= LED_PROGRAM;
}
```

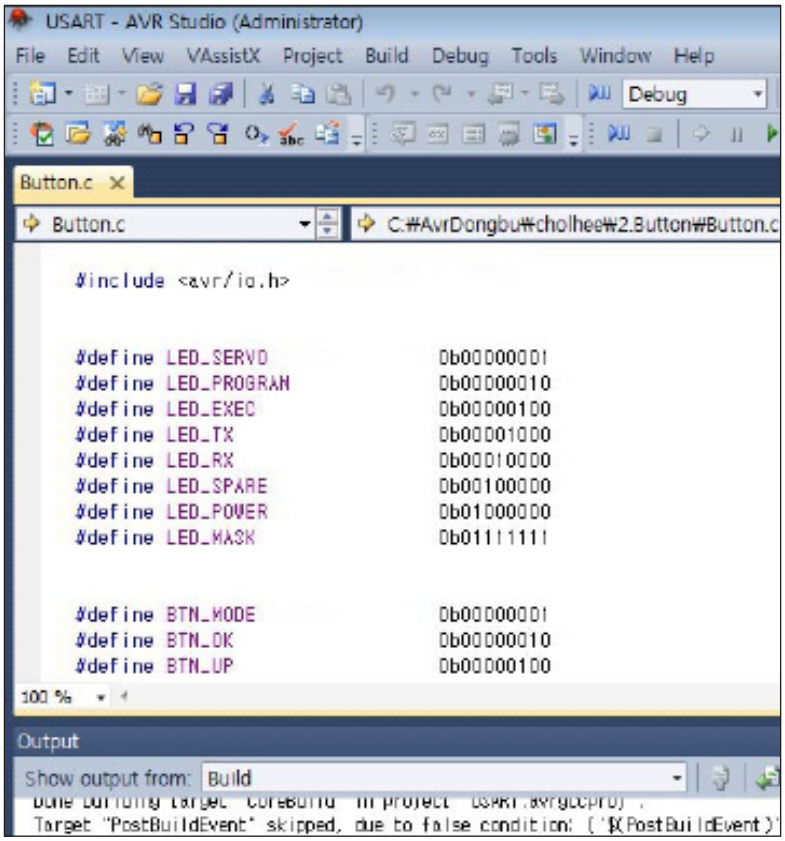
if(ucButton & BTN_MODE)coditional statement applies bitwise operation ucButton & BTN_MODE(0b00000001). Resulting value will be 0b00000001 if 0bit value of ucButton is 1 (Mode Button Pressed) , and 0 if 0 bit value is 0. If MODE Button is pressed, condition in the bracket of the if statement becomes true and content is executed. ucLED |= LED_PROGRAM in the if statement is short form of ucLED = ucLED | LED_PROGRAM. |(bitwise OR) operation applied to the value of ucLED and LED_PROGRAM(0b00000010) will change the ucLED value to the result. The result of this operation will change the 1st bit value of ucLED corresponding to Program LED to 1. Next if statements bring about similar results. Just as ucLED bit value corresponding to Program LED becomes 1 when Mode button is pressed, bit value corresponding to EXEC LED becomes 1 when OK button is pressed, TX LED bit value when UP button is pressed, RX LED when DOWN button is pressed, and etc.

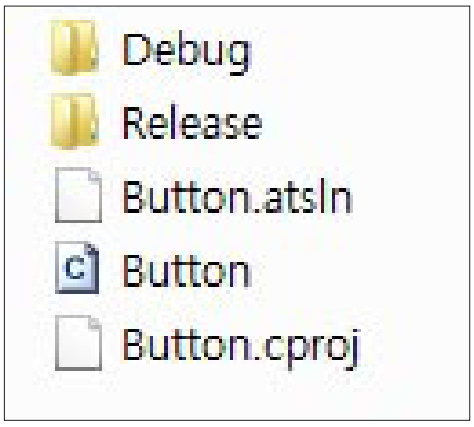
```
PORTC = (~ucLED) & LED_MASK;
```

Last section of the while statement applies changed ucLED value to PORTC. First, '~' operation is applied to ucLED since PORTC uses reversed value to turn on the applicable LED. Next, &(bitwise AND) operation is applied to LED_MASK(0b01111111) to convert 7th bit to 0 just as it was done to the button. Finally, resulting value is applied to PORTC to control the LED. Contents of the whle statement will run in infinite loop, reading the button input and controlling the LED continuously.

Program Execution

Download and run the program. Applicable LED will turn on when button is pressed.







01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be followed by 1 succeded 0 failed comment.

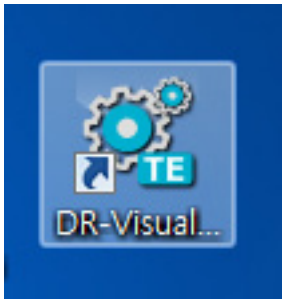
02 Debug Folder

After bulid, debug folder will be created within the source folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



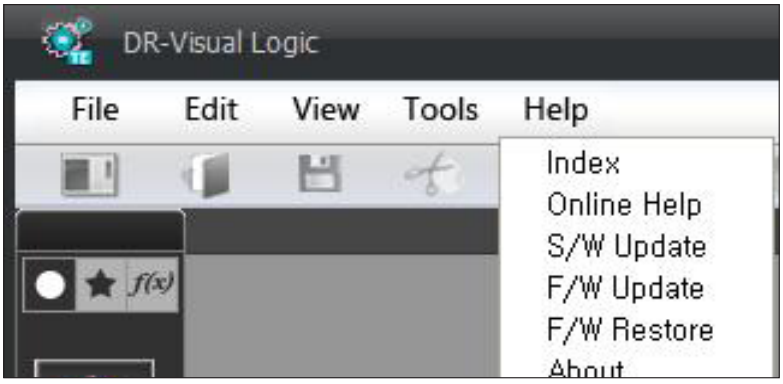
04 Visual Logic

Run DR-Visual Logic.



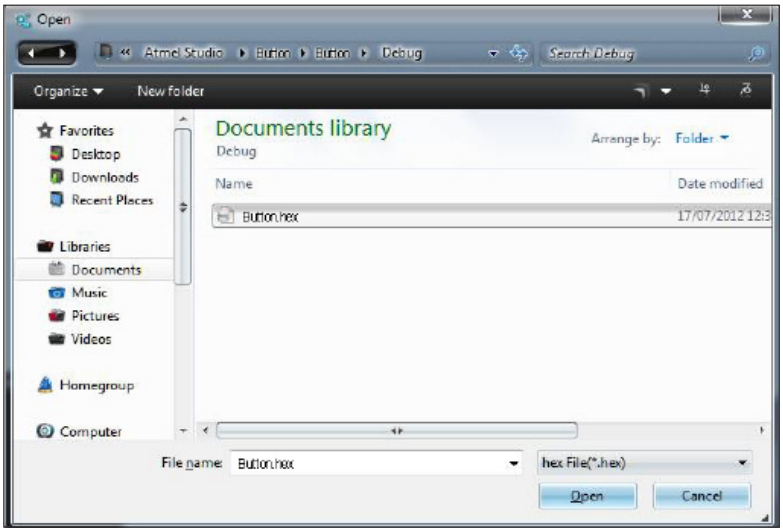
05 Connect

Select COM port number and click Connect.



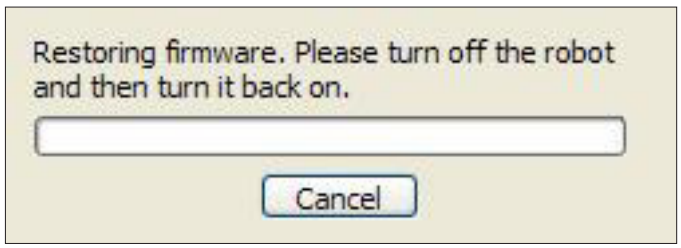
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



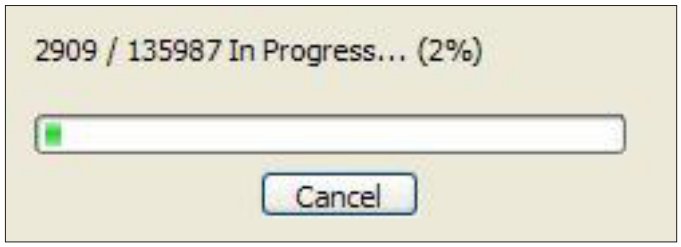
07 Load

Load Hex file from the Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restorateion in progress.



10 Run

Press the button to turn on the corresponding LED.

Emitting Sound From The Buzzer

Using ATmega128 digital I/O function to emit beep sound from the buzzer installed in the DRC.

AVR Fundamentals

Chapter 2 showed using the input from the buttons to control the LED. This chapter will show how to control the buzzer through the output pin to emit sound. Registers related to digital I/O are DDRX, PORTX, and PINX. Refer to Chapter 2 for details on registers..

H/W Fundamentals

Buzzer installed in DRC incorporates coils. Strength of the magnetic field around the coil changes when the current flowing in the buzzer fluctuates periodically. Sound is emitted when the metal blade vibrates due to the changing magnetic field strength. Circuit diagram below (Diagram 00) shows the buzzer connection to ATmega128. Buzzer is connected to the ATmega128 (Port 8 7th pin PB7) through the transistor. As buzzer requires more current than ATmega128 is able to provide, connecting the buzzzer directly to ATmega128 will cause error due to low current or damage the pin. Use of transistor between the pin and the buzzer increases the curent flowing into the buzzer even when current output from PB7 is relatively small. This enables the buzzer to operate even with small current output from PB7. As explained previously, buzzer emits sound when metal blade vibrates caused by the changing magnetic field when current strength fluctuates. Increasing the speed of current fluctuation (higher frequency) increases the pitch of the sound. This example will use the delay function to change the PB7 output at set intervals to emit different pitched sound.

Preparation



Preparing The Hardware

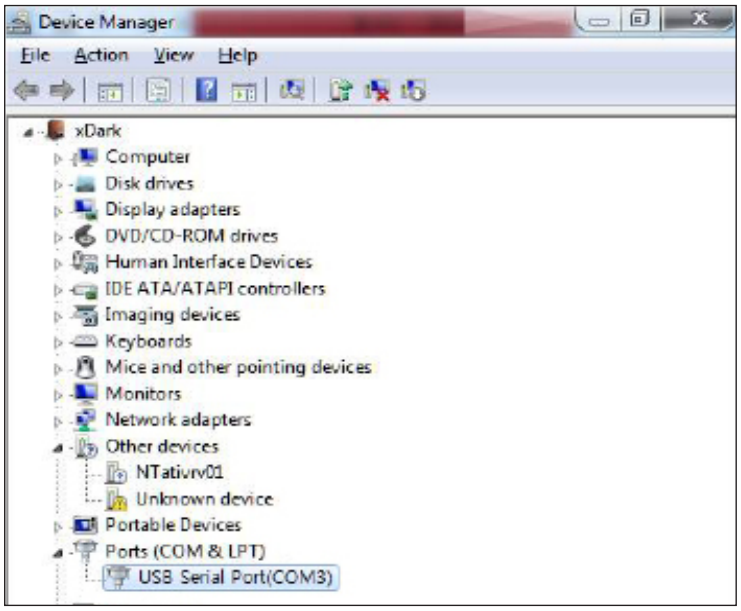
01 Connection

Plug in the DRC power adapter and connect the PC to DRC using USB to Serial Cable.



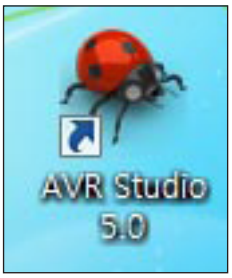
01 Connection

Plug in the DRC power adapter and connect the PC to DRC using USB to Serial Cable.



02 COM port Connection

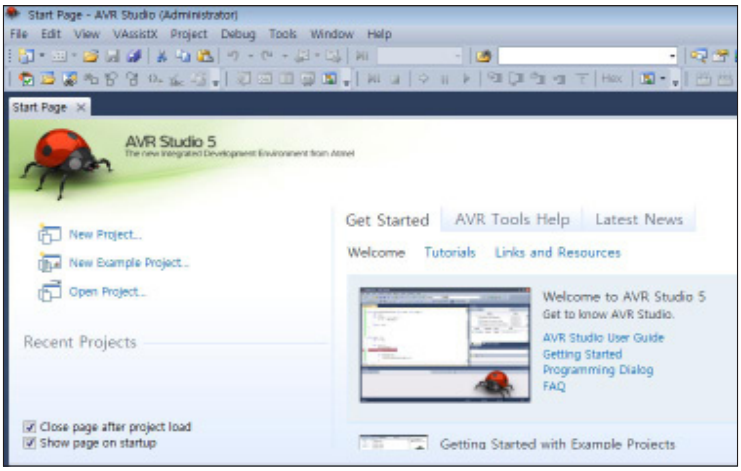
My Computer〉Property〉Device Manager. Check COM connection.



Running AVR Studio & Coding Preparation

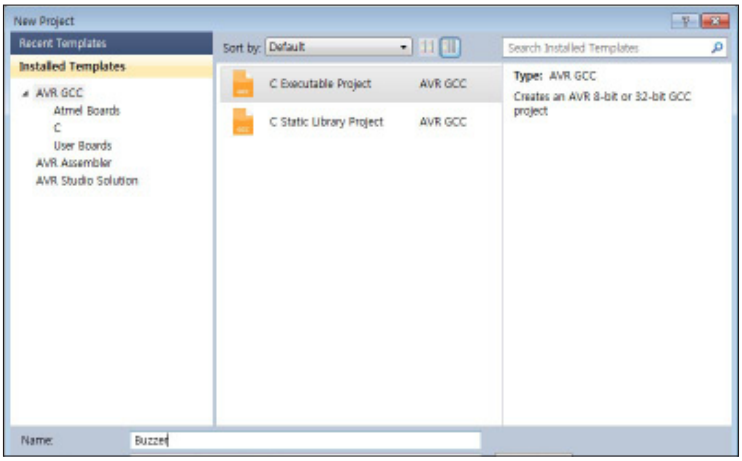
01 Run

Run AVR Stdio.



02 New Project

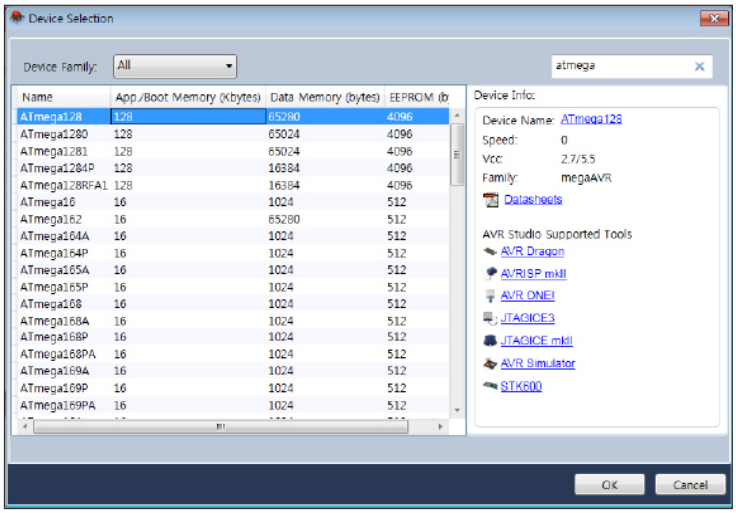
Click New Project.



03 Select GCC

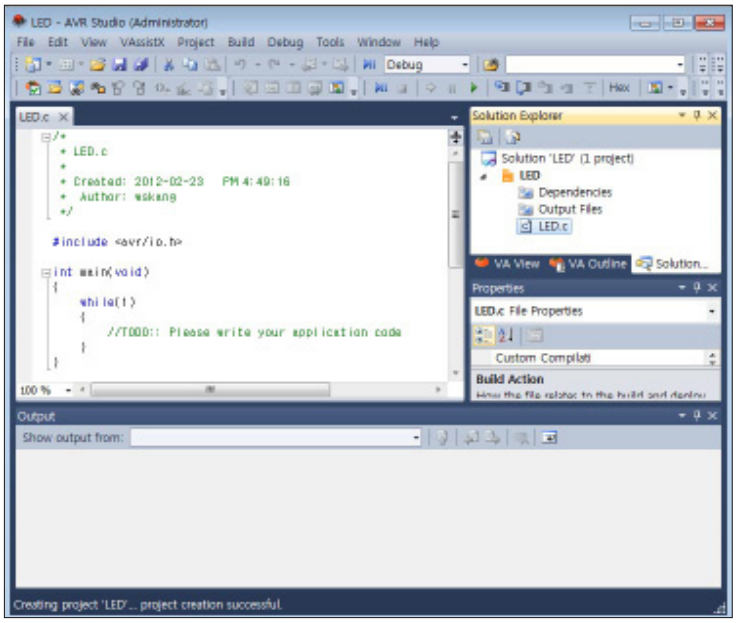
Select C Executable Project.

Input “Buzzer” as the Name, Click Create directory for solution and then press OK button.



04 Select ATmega

Search for atmega.
Select Atmega128 from the list.



05 Editor

Start coding in the edit window.

Follow the example below.

Example

In this example, buzzer will a emit sound OK button on DRC is pressed.

```
//CPU clock setup
#define F_CPU 16000000

//ATmega128 registers defined
#include <avr/io.h>
//_delay_ms delay function defined
#include <util/delay.h>

// Define PORT A pins corresponding to BTN names
#define BTN_MODE                0b00000001
#define BTN_OK                  0b00000010
#define BTN_UP                  0b00000100
#define BTN_DOWN                0b00001000
#define BTN_LEFT                0b00010000
#define BTN_RIGHT               0b00100000
#define BTN_MASK                0b00111111

int main(void)
{
        //Declare 8 variable
        unsigned char ucButton;

        //Set PORT A I/O direction
        DDRA = 0b00000000;

        //Set Buzzer Pin (PB7) I/O direction
        DDRB |= 0b10000000;
        //Set  Buzzer PIn (PB7) output value
        PORTB &= ~0b10000000;
```

```
//Set PORT C I/O direction
        DDRC = 0b01111111;
        //Set PORT C output value
        PORTC = 0b01111111;

        //Continuously repeat content of while statement
        while(1)
        {
                //Read button value from PORT A
                ucButton = (~PINA) & BTN_MASK;

                //Reverse the value of PB7 when OK button pressed
and wait 1ms
                if(ucButton & BTN_OK){
                        PORTB ^= 0b10000000;
                        _delay_ms(1);
                }
        }

        //Return integer value to fit format of the function
        return 1;
}
```

Example Description

```
#define F_CPU 16000000
```

#define statement is one of the commands prior to coding. Command replaces any F_CPU value found with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on clock speed, this command sets the clock speed so that delay function and other functions execute properly.

```
#include <avr/io.h>
#include <util/delay.h>
```

DDRX, PORTX, and other registers required for AVR control are defined in the avr/io.h header file. Frequently used functions such as _delay_ms() delaly function are defined in the util/delay.h header file. Use #include statement to include the functions.

```
#define BTN_MODE                0b00000001
...
#define BTN_MASK                0b00111111
```

#define BTN_MODE 0b00000001 will first replace BTN_MODE with 0b00000001 before the codes are compiled. #define statement is used to faciliate programming by providing the way to use recognizable names for each DRC button bits.

```
unsigned char ucButton;
```

Declare 8bit variable ucButton. ucButton is used to show the current state of the button. ucButton value within the while statement is reverse of the actual port input or output. ucButton value is reverse of PINA with pressed button value being 1 and released value being 0.

```
DDRA = 0b00000000;
DDRB |= 0b10000000;
PORTB &= ~0b10000000;
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Set Port A pin numbers 0~5 as input. Only the pins corresponding to the buttons are set as input. Set Port B pin number 7 as output and the output value as 0. Set Port C pin numbers 0~6 as output and outuput value as 1. All LEDs will be turned off.

Unlike other registers, DDRB and PORTB uses |=, &= instead of =. When |(bitwise OR) operation is applied to DDRB and 0b10000000, 0 bits will remain same as the DDRB value while 1 bits are changed to 1. This will change the 7th bit of DDRB to 1 and leave the rest unchanged. In reverse of previous case, when &(bitwise AND) operation is applie to PORTB and 0b01111111(result of ~0b10000000). 1 bits will remain same the PORTB value while 0 bits are changed to 0. This will change the 7th bit of PORTB to 0 and leave the rest unchanged.

```
ucButton = (~PINA) & BTN_MASK;
```

While statement starts by reading the button value into ucButton. Button value read from PINA is 0 if button is pressed and 1 if the button is released, ~ operation is applied to revese the read value. Apply &(bitwise AND) operation to BTN_MASK(0b00111111) to change the 6th and 7th bit values to 0 (6th & 7th bits are not related to button). ucButton will have value of 0b00XXXXXX wth Xs containing state of the buttons corresponding to each bit.

```
if(ucButton & BTN_OK){
        PORTB ^= 0b10000000;
        _delay_ms(1);
}
```

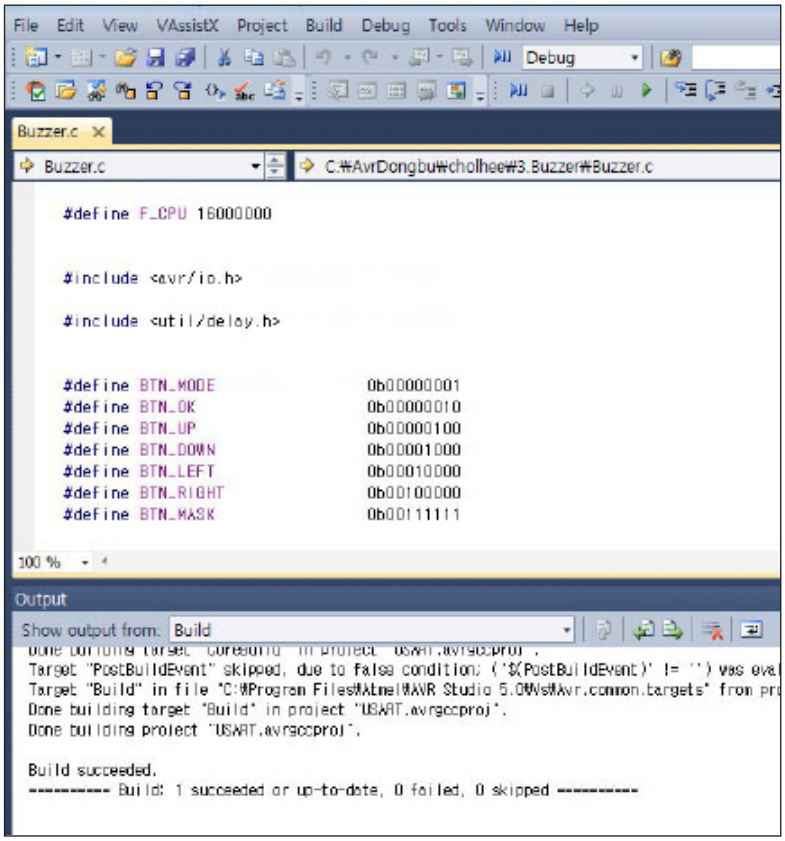
Result of if(ucButton & BTN_OK) becomes true if one of the bits in ucButton corresponding to OK button is 1, false if the bit is 0. Content of the bracket will be executed if the OK button is pressed.

PORTB ^= 0b10000000 in the if statement is short form of PORTB = PORTB ^ 0b10000000;. ^(bitwise XOR) applied to value of PORTB and 0b10000000 will change the PORTB value to the result . XOR(Exclusive OR) operation results in 1 if two bits are different and 0 if two bits are same. Therefore, PORTB ^= 0b10000000 will change the 7th bit of PORTB to 1 if it is originally 0 and to 0 if it is originally 1.

_delay_ms function is used to delay the operation for 1ms after output value of PB7 is changed. Since while statement continues to run in loop, output value of PB7 will cahnge every 1ms as long as the OK button is pressed. Therefore, current will flow into the buzzer approximately ever 2ms and the the buzzer will emit sound according to the frequency.

Run

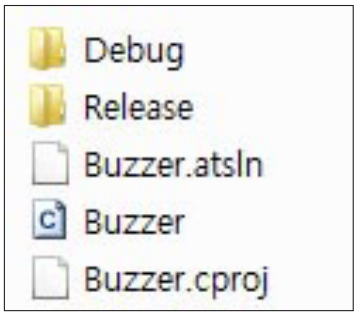
Download and run the program. Buzzer will emit sound with different pitch every 2ms as long as the OK button is pressed.

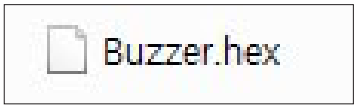


01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.





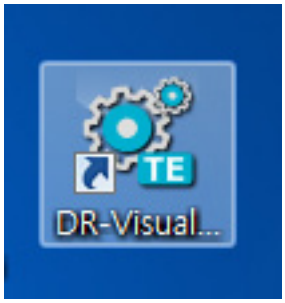
02 Debug Folder

After bulid, debug folder will be created within the source folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



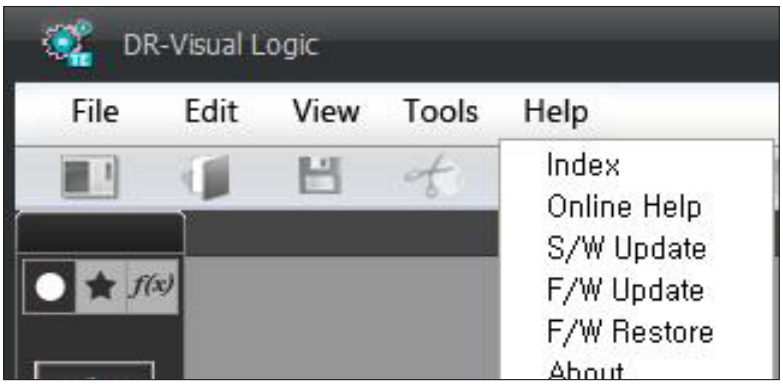
04 Visual Logic

Run DR-Visual Logic.



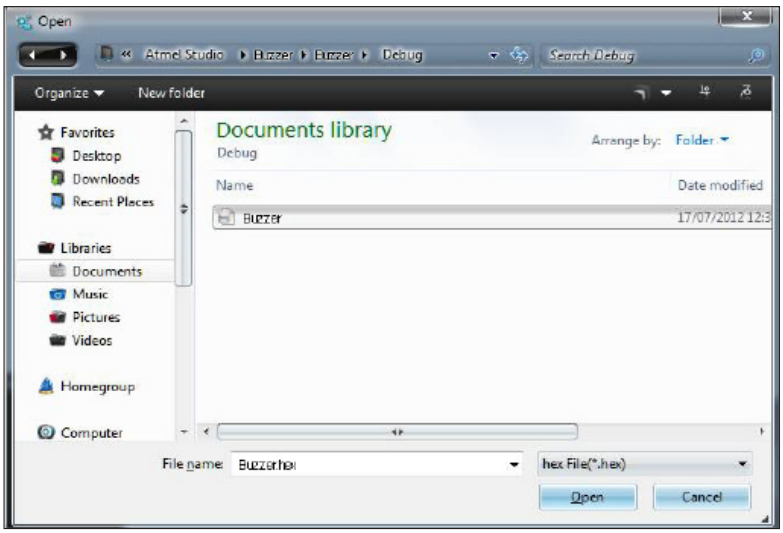
05 Connect

Select COM port number and click Connect.



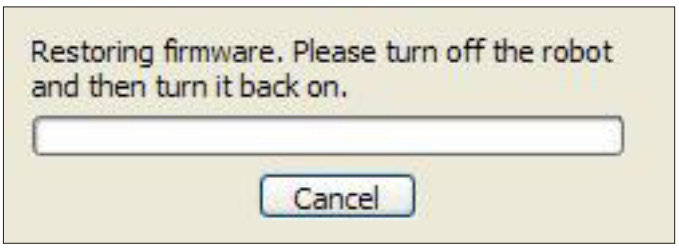
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



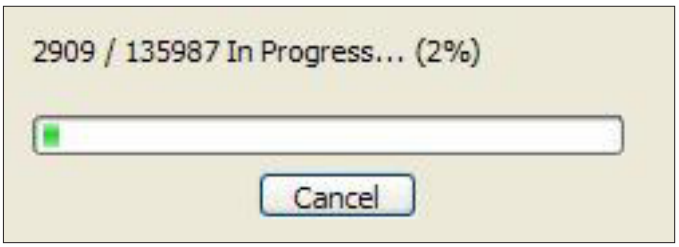
07 Load

Load Hex file from the Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restorateion in progress



10 Run

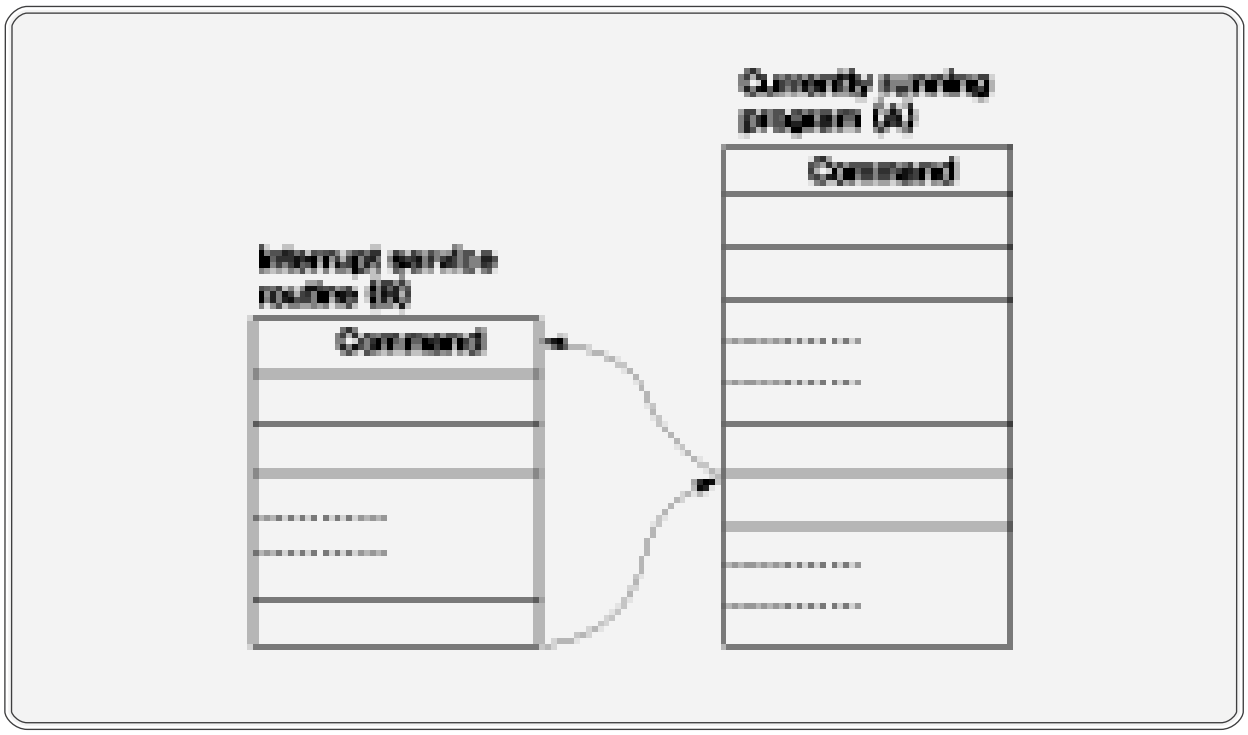
Press OK button to emit buzzer sound.

Receiving MIC Input From External Interrupt

Using ATmega128 external interrupt to receive input throught the MIC installed on DRC.

AVR Fundamentals

Using ATmega128 external interrupt function. Interrupt temporarily halts the current program and begins execution of another program. When interrupt condition occurs, current process is halted and saved to a spcific address and execution of interrrupt handler begins. Saved process resumes once the interrrupt process is completed (diagram 00). Interrrupt handler is also called Interrupt Service Routine (ISR). Using the interrupt enables the program to be processed more efficiently as well as ability to required action at has the advantage of,



As shown below, there are many interrupt sources that can be used by ATmega128. Interrupt sources that are higher in the list have the higher priority. This chapter will show how to use external interrupts in programming and the rest of the remaining chapters will show how to use the timer/counter, overlow, and USART reception complete interrupts.

Vector Number	Vector Address	Interrupt Source	Interrupt Description
1	$0000	RESET	Reset
2	$0002	INT0	External Interrupt 0
3	$0004	INT1	External Interrupt1
4	$0006	INT2	External Interrupt 2
5	$0008	INT3	External Interrupt 3
6	$000A	INT4	External Interrupt 4
7	$000C	INT5	External Interrupt 5
8	$000E	INT6	External Interrupt 6
9	$0010	INT7	External Interrupt 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Input Capture
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter 0 Compare match
17	$0020	TIMER0 OVF	Timer/Counter 0 Overflow
18	$0022	SPI, STC	SPI Serial Reception Complete
19	$0024	USART0, RX	USART0 Reception Complete
20	$0026	USART0, UDRE	USART0 Emptly Data Register
21	$0028	USART0, TX	USART0 Trasmission Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Compare
25	$0030	TIMER1 COMPC	Timer/Counter1 Compare match C
26	$0032	TIMER3 CAPT	Timer/Counter3 Input Capture
27	$0034	TIMER3 COMPA	Timer/Counter3 Compare match A
28	$0036	TIMER3 COMPB	Timer/Counter3 Compare match B
29	$0038	TIMER3 COMPC	Timer/Counter3 Compare match C
30	$003A	TIMER3 OVF	Timer/Counter3 Overflow
31	$003C	USART1 RX	USART1 Reception Complete
32	$003E	USART1 UDRE	USART1 Empty Data Register
33	$0040	USART1 TX	USART1 Transmission Complete
34	$0042	TWI	TWI(I2C) Direct Interface
35	$0044	SPM READY	프로그램 메모리 저장 준비 완료

ATmega128 uses PORT E for external interrupts. Each of the 8 pins in PORT E respond to the interrupt source INTn. Registers related to the external interrupt are shown below.

■ SREG : AVR Status Register

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

SREG register shows the overall status of the AVR. 7th bit of the register is related to interrupt.

– 7th bit I : Enable all interrupts

I bit has to be set to 1 to enable particular interrupt in ATmega128. Even though enabling each interrupt is done by relevant register bit setup, interrupt will not occur if I bit is set to 0.

■ EICRA : External Interrupt status register A

Bit	7	6	5	4	3	2	1	0
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Cal	0	0	0	0	0	0	0	0

EICRA register is used to select action mode of external interrupts 0–3. 2 bits are assigned to each interrupt source. Table below shows the action modes dependent on bit selection.

ISCn1	ISCn0	Description
0	0	Interrupt occurs when LOW input is received by INTn
0	1	Not used
1	0	Request asynchronous interrupt when falling edge (HIGH to LOW) occurs at INTn.
1	1	Request asynchronous interrupt when rising edge (LOW to HIGH) occurs at INTn

EICRB : External Interrupt Status Register B

Bit	7	6	5	4	3	2	1	0
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EICRB register is used to select action mode of external interrupts 0–3. 2 bits are assigned to each interrupt source. Table below shows the action modes dependent on bit selection.

ISCn1	ISCn0	Description
0	0	Interrupt occurs when LOW input is received by INTn
0	1	Interrupt occurs when falling edge or rising edge occurs at INTn
1	0	Request interrupt when falling edge (HIGH to LOW) occurst at INTn
1	1	Request interrupt when rising edge (LOW to HIGH) occurs at INTn

EIMSK : External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EIMSK register is used to determine whether to allow external interrupts. External interrupt is allowed when bit is set to 1 and disallowed when bit is set to 0. INTFn bit of EIFR becomes 1 and Interrupt subroutine is executed when SREG I bit and INTn is set to 1 and interrupt meeting the condition set in ISCn1:0 bit occurs.

EIFR : External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EIFR register shows status of each individual external interrupt requests. When change in status of PEn pin activates interrupt request, corresponding bit INTFn is set to 1 and later initialized to 0 by the hardware when interrupt service routine starts to execute. To change INTFn bit to 0 manually by using the software, 0 is written to INTFn. External interrupt n executes when I bit of SREG, INTn, and INTFn are all 1.

H/W Fundamentals

2 microphones (left & right) installed in DRC are connected to the external interrupt pin through the circuit. Signal entering the interrupt pin changes from LOW to HIGH when vibration greater than certain intensity is detected. Signal changes back to LOW when vibration disappears. Sound of a single clap changes the signal entering the interrrupt pin from LOW–〉HIGH–〉LOW and causes rising edge as signal changes from LOW to HIGH. DRC uses 2 external interrupts, left mic uses INT5 and right mic uses INT6 to receive the input. If the sound is from the left side, rising edge will occur from the left mic and INT5 interrupt will be activated first and vice versa. This example will activate interrupt when rising edge occurs in INT5 or INT6 and create interrupt service routine.

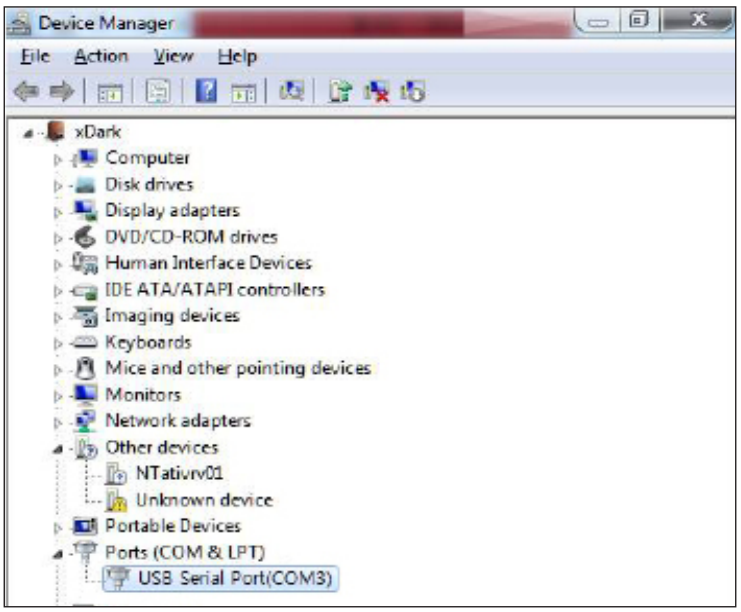
Preparation



Hardware Preparation

01 Connection

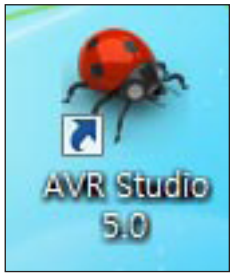
Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.



02 COM Port Connection

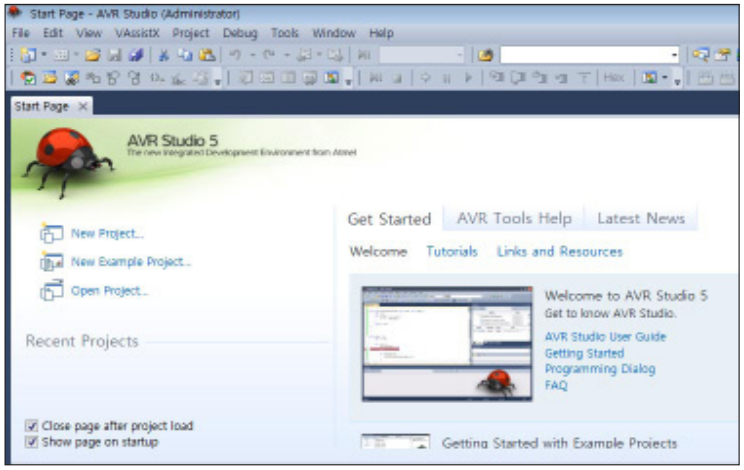
My Computer〉Property〉Device Manager. Check COM connection.

Running AVR Studio @ Coding Preparation



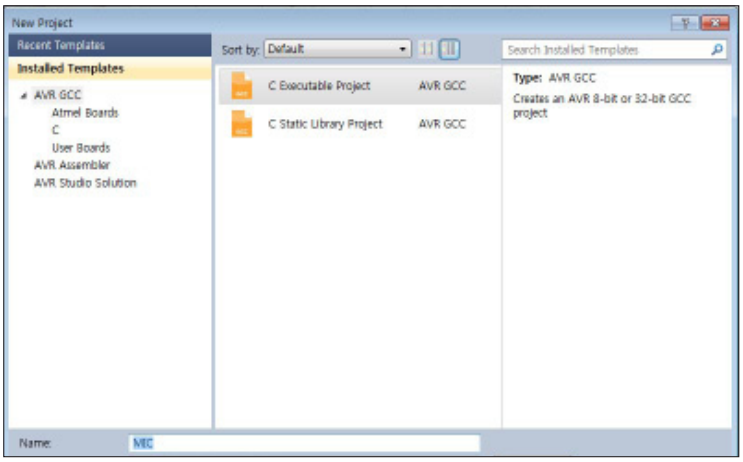
01 Run

Run AVR Stdio.



02 New Project

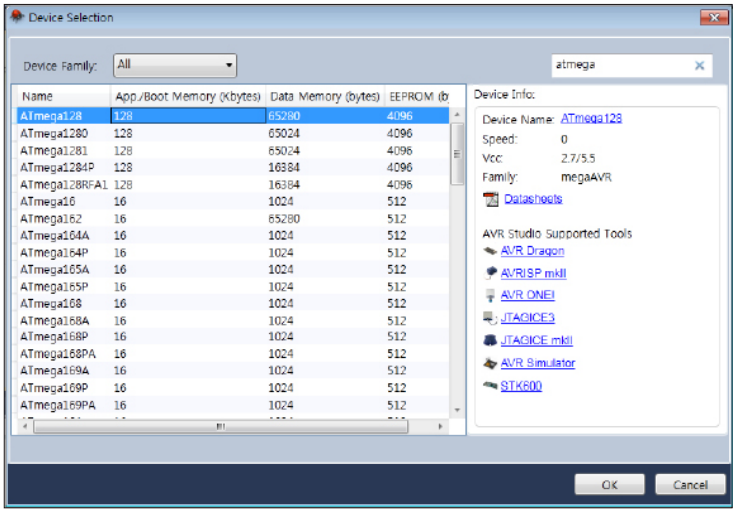
Click New Project.



03 Select GCC

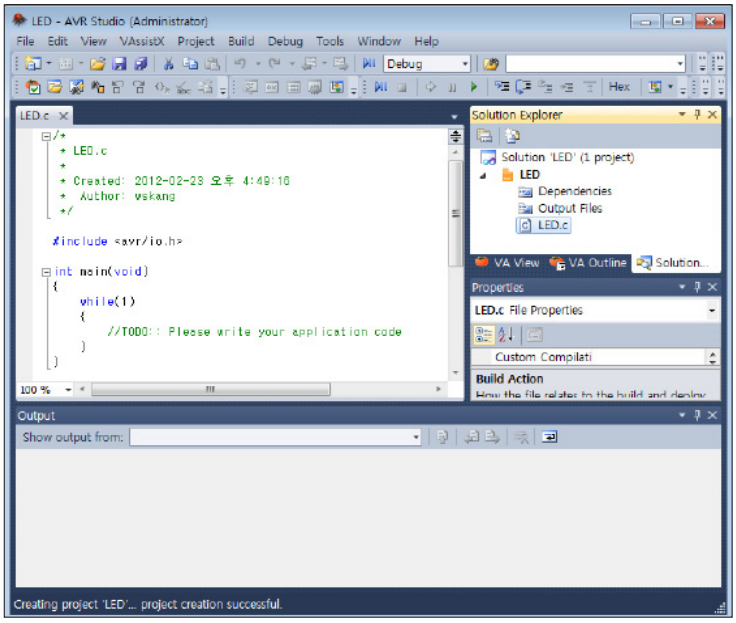
Select C Executable Project.

Input 'MIC' as Name, Click Create directory for solution, Press OK button.



04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



05 Editor

Start coding in the edit window.
Follow the example below.

Example

This example will turn on the Sevo LED for 0.5s when sound is detected from the left DRC MIC and turn on the POWER LED for 0.5s if sound is detected from the right MIC.

```
//Set CPU clock speed
#define F_CPU 16000000

//ATmega128 registers defined
#include <avr/io.h>
//information required for interrupt defined
#include <avr/interrupt.h>
//_delay_ms and other functions defined
#include <util/delay.h>

//Define Port C pins corresponding to LED names
#define LED_SERVO                    0b00000001
#define LED_PROGRAM                  0b00000010
#define LED_EXEC                     0b00000100
#define LED_TX                       0b00001000
#define LED_RX                       0b00010000
#define LED_SPARE                    0b00100000
#define LED_POWER                    0b01000000
#define LED_MASK                     0b01111111

//External interrupt number 5  ISR. turn on Servo LED for 0.5s.
ISR(INT5_vect){
        PORTC &= ~LED_SERVO;
        _delay_ms(500);
        PORTC |= LED_SERVO;
}

//External interrupt number 6 ISR. turn of Power LED for 0.5s
ISR(INT6_vect){
        PORTC &= ~LED_POWER;
        _delay_ms(500);
        PORTC |= LED_POWER;
}
```

```
int main(void)
{
        //Set PORT C I/O direction
        DDRC = 0b01111111;
        //Set PORT C output value
        PORTC = 0b01111111;

        //Disable all interrupts
        cli();

        //set INT5, INT6 to rising edge mode
        EICRB |= 0b00111100;
        //Enable INT5, INT6
        EIMSK |= 0b01100000;
        //Initialize INT5, INT6 interrupt flag to 0
        EIFR |= 0b01100000;

        //Enable all interrupts
        sei();

        //Continuously repeat content of while statement
        while(1)
        {

        }

        //Return integer value to fit format of the function
        return 1;
}
```

Example Description

```
#define F_CPU 16000000
```

#define statement is one of the commands prior to coding. Command replaces any F_CPU value found with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on clock speed, this command sets the clock speed so that delay function and other functions execute properly.

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

#define BTN_MODE 0b00000001 will replace BTN_MODE with 0b00000001 before compiling. #define statement is used to faciliate programming by replacing DRC button bits with easily recognizable names.

```
#include <util/delay.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. Functions for using interrupt are defined in the avr/interrupt.h header file. Frequently used functions such as _delay_ms() delaly function are defined in the util/delay.h header file. Use #include statement to include the functions.

```
#define LED_SERVO                    0b00000001
...
#define LED_MASK                     0b01111111
```

#define LED_SERVO 0b00000001 will replace LED_SERVO with 0b00000001before compiling. #dfines statement is used to facilitate programming by replacing DRC LED bits with easily recognizable names.

```
ISR(INT5_vect){
        PORTC &= ~LED_SERVO;
        _delay_ms(500);
        PORTC |= LED_SERVO;
}
ISR(INT6_vect){
        PORTC &= ~LED_POWER;
        _delay_ms(500);
        PORTC |= LED_POWER;
```

Interrupt service routine for external interrupt number 5 & 6. When relavant bit in EIFR becomes 1 due to an interrupt, current process will be halted and interrupt subroutine will be executed. EIFR register bit will be initialized back to 0 as soon as subroutine starts. When rising edge occurs in INT5 due sound detected from the left MIC, INT5 interrupt subroutine will change the PORTC bit corrresponding to Servo LED to 0 and turn on the Servo LED for 500ms and then change the bit back to 1 to turn off the Servo LED. When rising edge occours in INT6 due to the sound detected from the right MIC, INT6 interrupt subroutine will turn on the Power LED for 500ms before turning it off again.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Set 0~6 pins of PORTC to output and set output value to 1 to turn off all LED.

```
cli();

EICRB |= 0b00111100;
EIMSK |= 0b01100000;
EIFR |= 0b01100000;

sei();
```

cli() and sei() are commands for disabling and enabling all interrupts. ATmega128 contains AVR status register named SREG. 7th bit of SREG register I is used for allowing the jump to interrupt subrountine when interrupt request is received. I bit has to be set to 1 for interrupt to function even if all interrupts are enabled. cli() command wil set I bit to 0 and sei() command will set I bit to 1. First, diable all interrupts using the cli() command to prevent interrupt request interfering wtih the interrupt setting change process. Change the external interrupt settings by using appropriate values in the registers assoicated with the interrupt. To use rising edge mode in external interrupt numbers 5 & 6, change ISC61, ISC60, ISC51, and ISC50 in EICRB to 1. Apply |(bitwise OR) opeartion to EICRB and 0b00111100 to change the applicable bits to 1. Also, change the 5th and 6th bit of EIMSK to 1 to allow use of external interrupts. Apply |(bitwise OR) operation to EIMSK and 0b01100000 to change 5th and 6th bit to 1. Initialize 5th and 6th bit of EIFR to 0. As explained previously,
1 has to be written manually to change EIFR bit to 0. Write 0 to 5th and 6th bit of EIFR. Use sei() command to enable all interrupts after completing the interrupt setup.

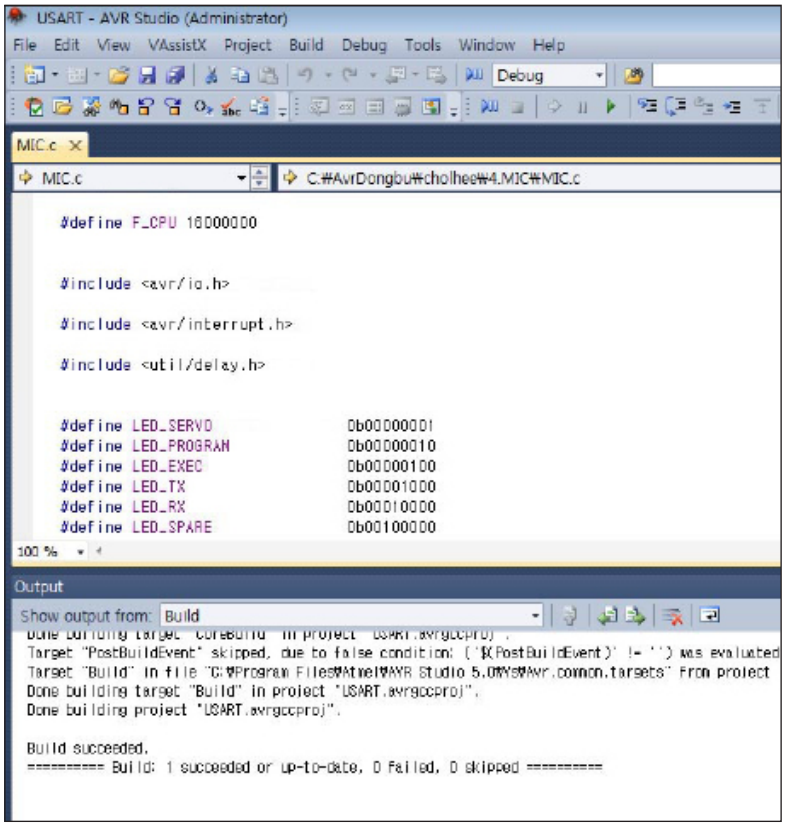
```
while(1)
{

}
```

This program contains empty while statement which will run in continuous loop unless it is halted temporarily by an interrupt request and jump is made to interrupt subroutine. Empty while statement will start to run again from the spot where it was temporarily halted once the subroutine is completed.

Run

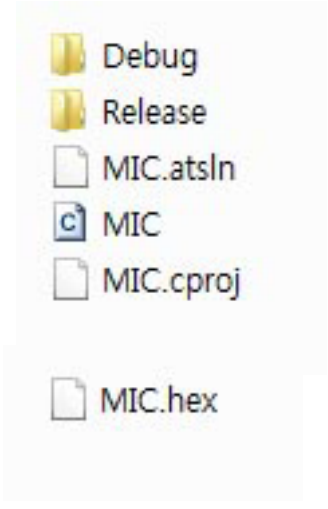
Download and run the program. Servo LED and Power LED will come on for 0.5s when sound is detected. Left clap will turn on the Servo LED first and then the Power LED. Right clap will turn on the Power LED first and then the Servo LED. Slight delay in 2nd LED is due to the difference in time it takes for the clapping sound to reach the microphones resulting in slight difference in timing of the interrrupt. If INT6 occurs while INT5 interrupt routine is being executed, 6th bit of EIFT will remain as 1 until INT5 interrupt service routine is completed and INT6 interrupt service routine starts to execute, at which point 6th bit will be initialized to 0.



01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.



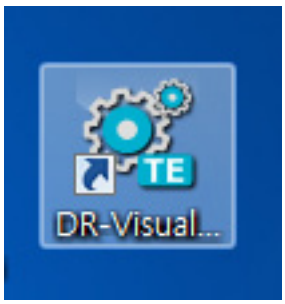
02 Debug

After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



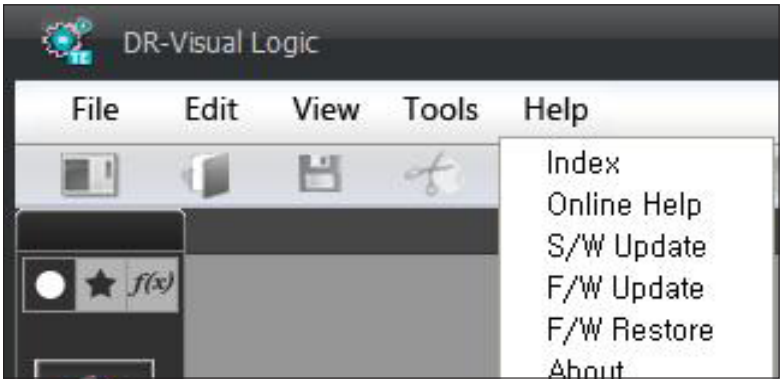
04 Visual Logic

Run DR-Visual Logic.



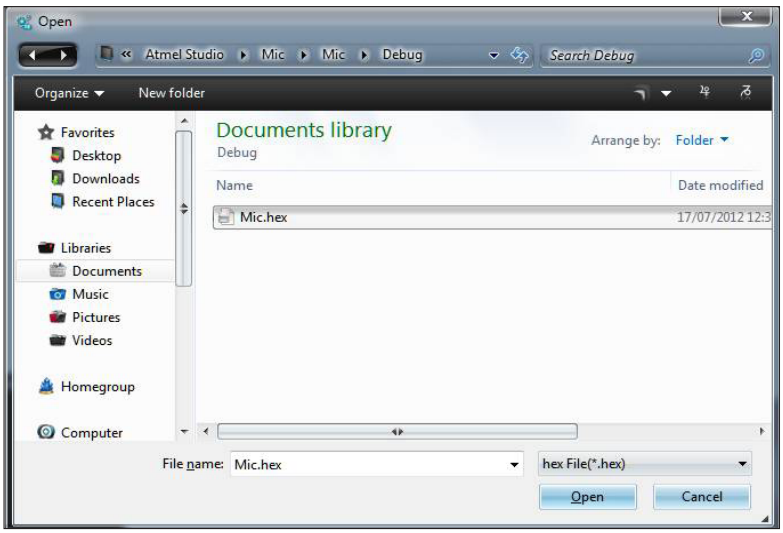
05 Connect

Select COM port and click Connect.



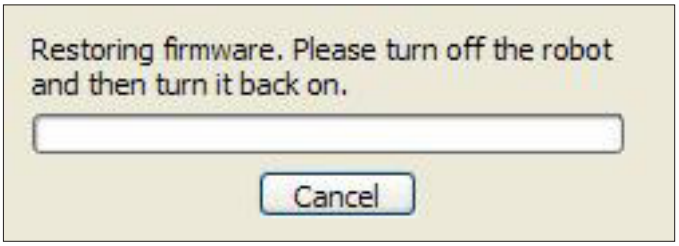
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



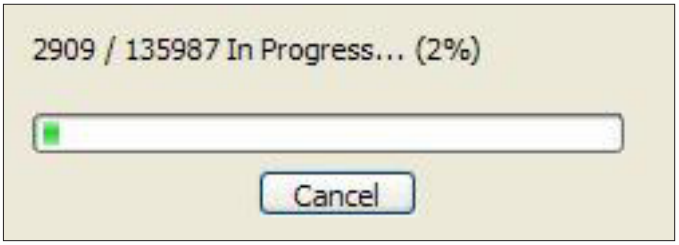
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.

10 Run

Left clap will turn on the Servo LED first and then the Power LED. Right clap will turn on the Power LED first and then the Servo LED.

Using Timer/Counter in Delay Function

Using ATmega128 Timer/Counter to implement delay. Delay will be used in LED example from Chaper 1.

AVR Fundamentals

ATmega128 has total of 4 Timer/Counters consisting of 8 bit Timer/Counters Timer/Counter 0, Timer/Counter 2, and 16bit Timer/Counters Timer/Counter 1, and Timer/Counter 3. Timer/Counter is used to measure the time between specific signals, count number of signals, generate pwm pulse, and etc. Summary of Timer/Counter functions are as follows.

– Standard Timer Function

Based on the selected clock, Timer/Counter will increment the counter register value periodically. When overflow occurs, overflow interrupt flag will be set to 1 and counter register will be initialized to 0. Overflow interrupt can be made to occur at desired intervals if counter register is initialized to a value other than 0 whenever overflow occurs.

– PWM Pulse Generation

Overflow occurs periodically when Timer/Counter is in PWM pulse mode. PWM pulse can be generated by setting reference value between initial value 0 and and the maximum value, and using the time interval from 0 to the reference value or from the reference value to the overflow to generate either High–LOW or LOW–HIGH wave. PWM can be controlled by changing the reference value to adjust the duty cycle.

※ What is PWM?

Pulse Width Modulation (PWM) or Pulse Duration Modulation (PDM). Input voltage can be varied to control the speed of the motor or brightness of an LED when analog voltage is used to control the motor or LED. However, control by digital signal is limited as digital signal can only have High or Low value. PWM is used to overcome this limitation by sending out fast periodic digital pulses and controlling the duration of HIGH pulses to achieve results similar to using analog voltage. For example, if HIGH has 5V voltage and LOW has 0V voltage, sending out fast periodic digital pulses from the digital output pin wih HIGH to LOW signal length ratio of 1:1 would have same effect as sending out 2.5V output. If the signal length ratio was 4:1, result would be similar to 4V output. The ratio of HIGH signal length to total signal length is called Duty Cycle. If HIGH:LOW ratio was 1:1, duty cycle would be 50% and duty cycle would be 80% if the rato was 4:1.

– Toggle

If the value of the counter register is equal to the reference value, output pin state can be toggled (HIGH to LOW or vice versa). Frequency of the digital pulse from the output pin can be controlled by changing the reference value.

– Pulse Width Measurement

Input capture function can be used to detect the changes in the signal coming into the pin and capture the time of the change. Resulting value can be used to to compare the length or measure the time interval between the signals.

– Counter

Changes in the input singal can be received by the clock and the number of signal changes counted by letting the counter register increase everytime there is a change in the input signal.

This example will use the Timer/Counter0 as standard timer and use the Timer/Couter0 to replace the _delay_ms function in the LED example from Chapter 1.

Registers related to Timer/Counter0 are as shwon below.

TCCR0 : Timer/Counter0 control register

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/ Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

– Bit 2:0 CS02:0 : Clock Selection

Select clock for Timer/Counter0. Divide the ATmega128 clock frequency by prescalar (integer number) and decide which to use for TCNT0. DRC has standard clock frequency of 16Mhz. If 0b101 is selected for CS02:0, Timer/Counter frequency becomes 16Mhz/128 = 125khz and TCNT0 will increase by 1 every 8us.

CS02	CS01	CS00	Description
0	0	0	NA
0	0	1	Clock/1
0	1	0	Clock/8
0	1	1	Clock/32
1	0	0	Clock/64
1	0	1	Clock/128
1	1	0	Clock/256
1	1	1	Clock/1024

– 6th Bit, 3rd Bit WGM01:0 : Pulse Generation Mode

Selecting the Timer/Counter mode. Direction of the counter register, counter maximum value, and generated pulse type will change with the selected mode. Timer/Counter has 4 selectable modes; standard, CTC Mode, and two types of PWM mode. Refer to below for detailed information on each type of mode.

ISCn1	ISCn0	Mode	Maximum Value	OCR0 Register Update Time	Time TOV0 Flag becomes 0
0	0	Standard	255(0xFF)	Immediately after selection	TCNT0=255
0	1	Phase Correct PWM	255(0xFF)	TCNT0=Maxium Value	TCNT0=0
1	0	CTC	OCR0	Immediately after selection	TCNT0=255
1	1	High speed PWM	255(0xFF)	TCNT0=0	TCNT0=255

– Bit 5:4 COM01:0 : Output Match & Comparison Mode

These 2 bits are used to select the function or behavior of OC0 terminal which is located at 4th pin (PB4) of PORT 8. If either of COM01:0 has value of 1, PB4 will will function as Timer/Counter0 output comparison pin instead of standard I/O port. Even with same COM01:0 values, OC0 pin in different modes will have different functions. Comparison & match mentioned in the table above refers to the the exact moment when the vaule of OCR0(Output comaprison register) becomes equal to the value of TCNT0(Timer/Counter0 register). To generate output from OC0 pin, 4th bit of DDRB has to be set as ouptput pin even if COM01:0 is set to use OC0 pin.

Output Comparison Mode In Standard & CTC Mode (Not PWM)

COM01	COM00	OC0 Pin Function
0	0	Standard I/O Port
0	1	Toggle OC0 ouput when comparison match occurs
1	0	Set OC0 output to LOW when comparison match occurs
1	1	Set OC0 output to HIGH when comparison match occurs

Comparison Output Mode In High Speed PWM Mode

COM01	COM00	OC0 Pin Function
0	0	Standard I/O Port
0	1	NA
1	0	Set OC0 output to LOW when comparison match occurs Set OC0 output to HIGH when TCNT0=0
1	1	Set OC0 output to HIGH when comparison match occurs Set OC0 output to LOW when TCNT0=0

■ Output Comparison Mode In Phase Correct PWM Mode

COM01	COM00	OC0 Pin Function
0	0	Standard I/O Port
0	1	NA
1	0	Set OC0 output to LOW when up count comparison match occurs Set OC0 otupt to HIGH when down count comparison match occurs
1	1	Set OC0 output to HIGH when up count comparison match occurs Set OC0 output to LOW when down count comaprison match occurs

– 7th Bit FOC0 : Forced Ouput Comparison

FOC0 only works when it is not in PWM mode. When value of 1 is written to the 7th bit, comparison match is forced regardless of the timer value.Output value of OC0 is changed according to COM01:0 setting. Forced comparison match by this bit does not generate interrupt nor does it initialzie the timer in CTC MODE. 7th bit can only be written to and it is always read as 0.

■ TCNT0 : Timer/Counter0 Register

Bit	7	6	5	4	3	2	1	0
	TCNT0[7:0]							
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Values	0	0	0	0	0	0	0	0

TCNT0 holds the current Timer/Counter0 value. As determined by CS02:0 bit, Timer/Counter0 will be incremented by 1 at each cycle. While value can be written and read directly from this register, care must be taken when using this register as it is possible to miss the required comparison match since comparison match does not occur immediately after value is written to TCNT0.

■ OCR0 : Timer/Counter0 Output Comparison Register

Bit	7	6	5	4	3	2	1	0
	OCR0[7:0]							
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Value in OCR0 register is compared with TCNT0 each time TCNT0 value changes with comparison match occuring when values in OCR0 and TCNT0 become equal. Output from OC0 terminal will vary depending on the funtion mode and COM02:0 bit values. Depending on the function mode, change in OCR0 value may not be reflected immediately and delayed until TCNT0 reaches certain value. Refer to WGM01:0 bit table for more details.

■ TIMSK : Timer/Counter0 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIEB	TOIE1	OCIE0	TOIE0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

– 1st Bit OCIE0 : Timer/Counter0 Output Comarison Match Interrupt Activation

When this particular bit is set to 1, Timer/Counter0 comparison match interrupt becomes active. OCF0 bit of TIFR register becomes 1 and corresponding interrupt routine is executed when comparison match occurs with OCIE0 and SREG register I bit set to 1.

– 0th Bit TOIE0 : Timer/Counter0 Overflow Interrupt Activation

When this particular bit set to 1, Timer/Counter0 overflow interrupt becomes active. TOV0 bit of TIFR register becomes 1 and corresponding interrupt service routine is executed when overflow occurs with TOIE0 bit and SREG register I bit is set to 1.

■ TIFR : Timer/Counter0 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	OCF2	TOF2	ICF1	OCF1A	OCFB	TOV1	OCIE0	TOIE0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

– 1st Bit OCF0 : Timer/Counter0 Output Comparison Flag

OCF0 becomes 1 when comparison match occurs between TCNT0 and OCR0. OCF0 is initialized to 0 by the hardware when corresponding interrupt subroutine starts to execute. To change OCF0 bit to 0 manually by using the software, 1 is written to OCF0. Timer/Counter0 comparison mtach interrupt is activated when OCF0, SREG I bit, and OCIE0 all have value of 1.

– 0th Bit TOV0 : Timer/Counter0 Overflow Flag

TOV0 becomes 1 when overflow occurs at Timer/Counter0 and then it is initialized to 0 by the hardware when corresponding interrupt subroutine starts to execute. To change TOV0 bit to 0 manually by using the software, 1 is written to TOV0. Timer/Counter0 overflow interrupt is activated when TOV0, SREG I bit, and TOIE0 all have value of 1. Unlike in other modes, in Phase Correct PWM mode, TOV0 bit is set to 1 when direction of TCNT0 increment changes from downcount to upcount (TCNT0=0).

ASSR, SFIOR, and other registers are also related to Counter/Timer0 operation in addition to the registers mentioned peviously. Refer to ATmega128 data sheet or other reference books for more detailed information.

As mentioned previously during TCCR0 description, Timer/Counter0 has 4 function modes.

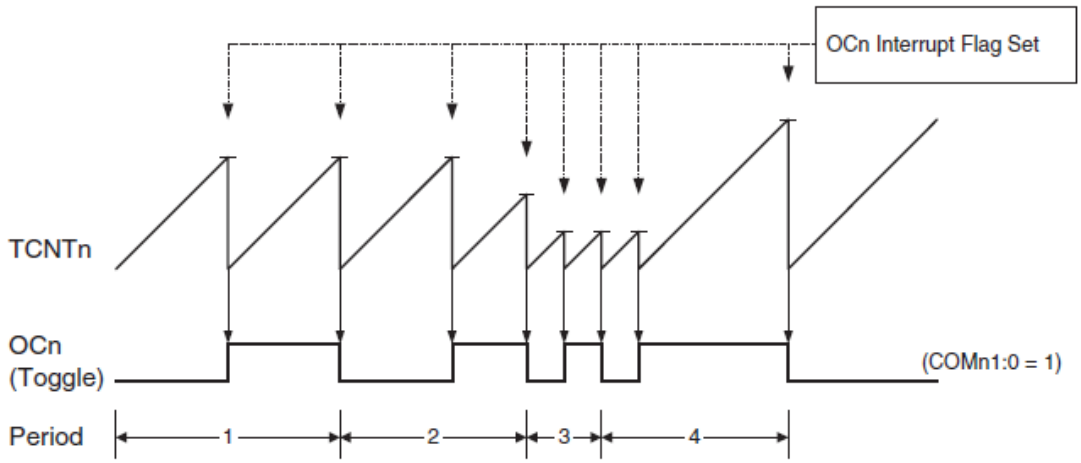
1. Standard Mode(WGM01:0 = 0b00)

This is the simplest of the fuction modes with TCNT0 always operating as upcounter (Increment counter), TOV0 becomes 1 when the value changes from maximum 255 to minimum 0. When both TOIE0 and SREG I bit is 1, Timer/Counter0 overflow interrupt service routine is executed and TVO0 is initialized back to 0. Stanard Mode is mostly used for Timer/Counter0 overflow interrupt and counting external clock signals. Comparison match interrupt can also be used as well but it is not recommended due to high CPU usage time.

2. CTC Mode(WGM01:0 = 0b10)

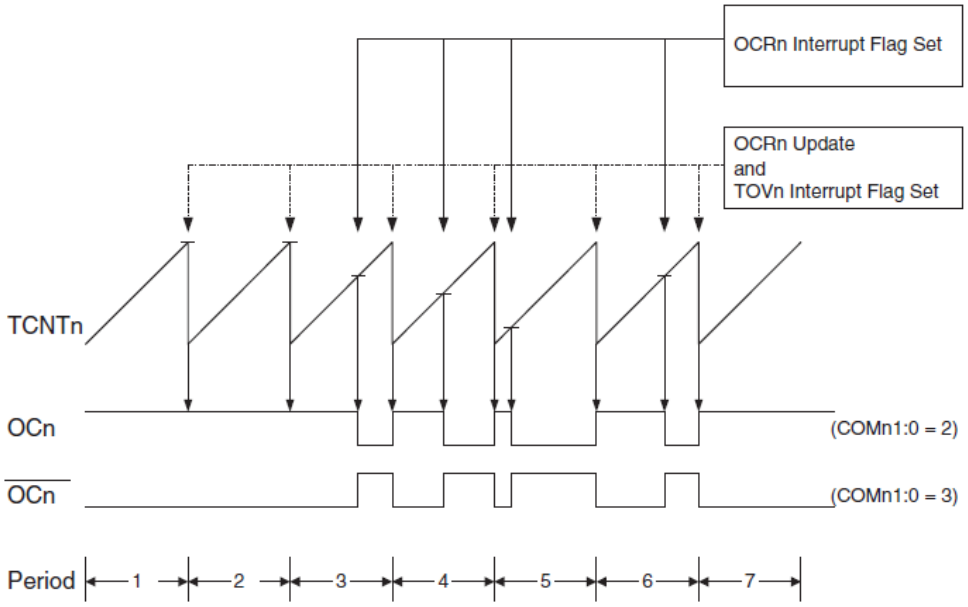
In CTC mode (Clear Timer on Compare mode) OCR0 register is used to determine the timer initialization period. TCNT0 register is initialized to 0 when TCNT0 value becomes equal (Comparison match) to OCR0 value. In other words, OCR0 register value determines the maximum counter value. When comparison match occurs, OCF0 bit changes to 1, Timer/Counter0 overflow interrut service routine is executed and TCNT0 is initialized to 0.

OC0 pin can be made to toggle each time comparison match occurs by setting COM01:0 value to 0b01. This feature can be used by OC0 pin to generate periodic pulses and pulse frequency controlled by changing the value of OCR0



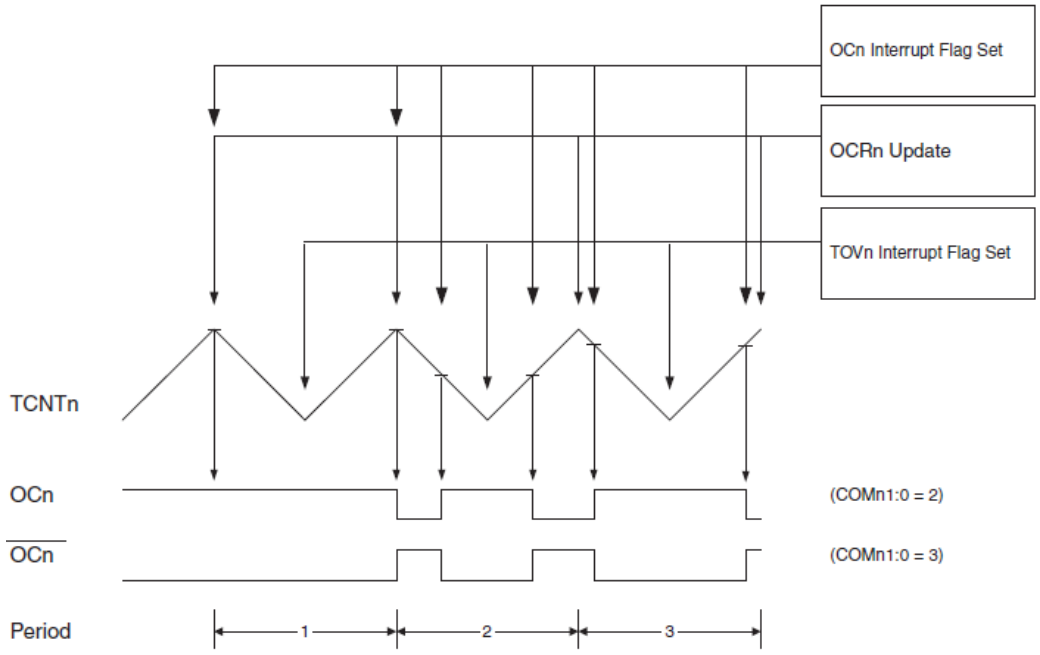
3. High Speed PWM Mode(WGM01:0 = 0b11)

In High Speed PWM Mode, TCNT0 value increases from 0 to 255 and then goes back to 0 showing saw blade like pattern and generates PWM output with duty cycle that changes accroding to the value of OCR0. When COM01:0 is set to 0b10, OC0 changes to LOW when comparison match occurs and changes to HIGH when TCNT0 is 0. Therefore, OC0 functions in non-inverting output comparison mode with 0~OCR0 generating HIGH and OCR0~255 generating LOW pulse. When COM01:0 is set to 0b11, OC0 changes to HIGH when comparison comparison match occurs and changes to LOW when TCNT0 is 0.Therefore, OC0 functions in inverting output comarison mode with 0~OCR0 generating LOW pulse and OCR0~255 generating HIGH pulse.



4. Phase Correct PWM Mode(WGM01:0=0b01)

Phase Correct PWM mode is similar to High Speed PWM mode but instead of TCNT0 always increasing from 0 to 255, TCNT0 value will increase from 0 to 255 and then reverse direction from 255 to 0. Phase Correct PWM mode has a period or cycle that is twice the length of High Speed PWM mode with each period lasting from 0–255 and 255–0 but it is often used for motor control due to its symmetrical quality with TCNT0=0 as the center. When COM01:0 is set to 0b10, OC0 pin changes to LOW when comparison match occurs during the upcount and changes to HIGH when comparison match occurs during the downcount (Non–Inverting Output Comparison Mode). When COM01:0 is set to 0b11, OC0 pin changes to HIGH when comparison match occurs during the upcount and changes to LOW when comparison match occurs during the downcount (Inverting Output Comparison Mode).



H/W Fundamentals

This example will use the Timer/Couter0 in ATmega128 to control the LED. Refer to Chapter 1 for information concerning LED hardware.

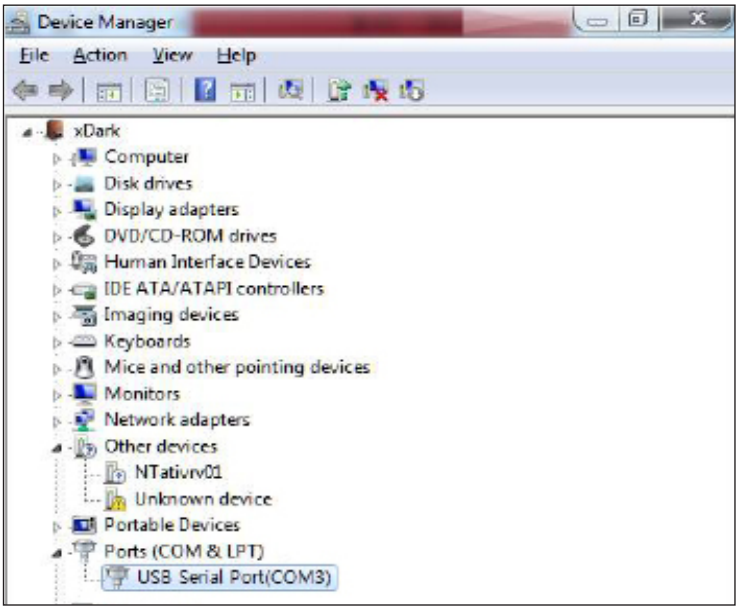
Preparation

Hardware Preparation



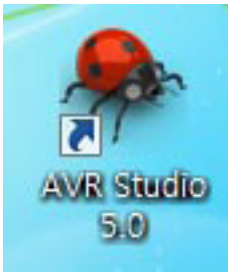
01 Connection

Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.



02 COM Port Connection

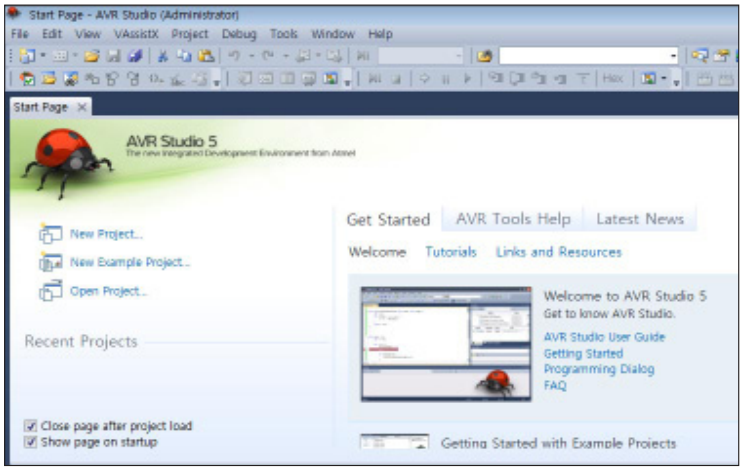
My Computer〉Property〉Device Manager. Check COM connection.



Running AVR Studio & Coding Preparation

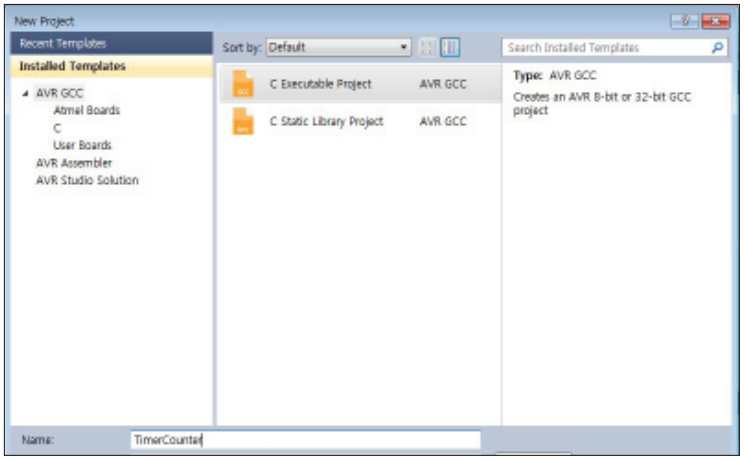
01 Run

Run AVR Stdio.



02 New Project

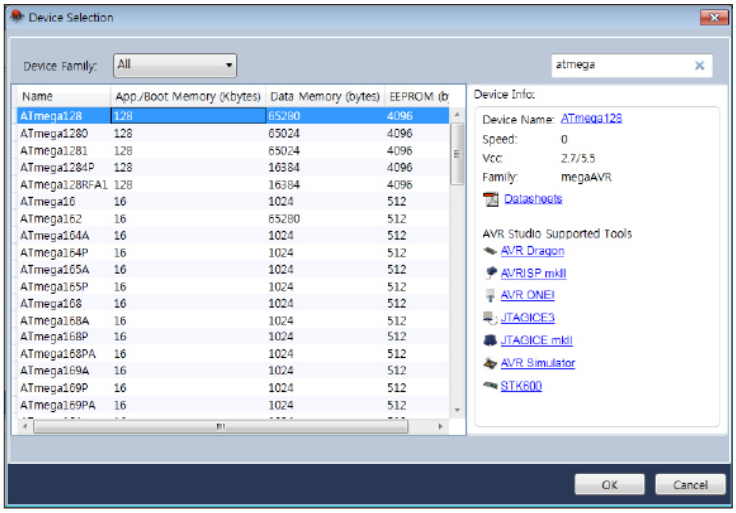
Click New Project.



03 Select GCC

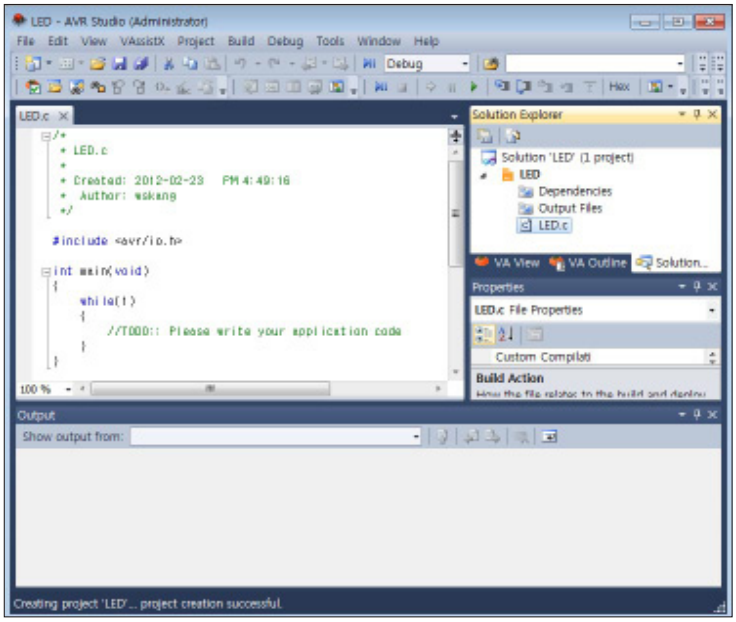
Select C Executable Project.

Input 'TimerCounter' as Name, Click Create directory for solution, Press OK button.



04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



05 Editor

Start coding in the edit window.
Follow the example below.

Example

This example will repeatdly turn on the DRC LEDs in sequence from Servo(0 pin) to Power(6pin).

Delay function will be provided by TImer/Counter0.

```
//ATmega128 registers defined
#include <avr/io.h>
//Information required for interrupt defined
#include <avr/interrupt.h>

//Timer/Counter0 clock speed and TCNT0 initial value
#define TIMER0_DIVIDER
0b00000101
#define TIMER0_TCNT0_PRELOAD                    6

//Declare global variable for measuring time in TImer/Counter0
volatile unsigned char gucTimerTick;

//Timer/Counter0 overflow interrupt service routine
ISR(TIMER0_OVF_vect){
        //Reset TCNT0 to initial value
        TCNT0 = TIMER0_TCNT0_PRELOAD;

        //Decrease global variable by 1 if not 0
        if(gucTimerTick != 0){
                gucTimerTick--;
        }
}

int main(void)
{
        //Declare 8bit variable
        unsigned char i;

        //Set Port C I/O direction
        DDRC = 0b01111111;
        //Set Port C output value
        PORTC = 0b01111111;
```

```
//Disable all interrupts
        cli();

        //Set TCNT0 initial value
        TCNT0 = TIMER0_TCNT0_PRELOAD;
        //Enable Timer/Counter0 overflow interrupt
        TIMSK |= (1<<TOIE0);
        //Set Timer/Counter0 overflow flag to 0
        TIFR |= (1<<TOV0);
        //Standard Mode, start Timer/Counter0 with clock speed set
without OC0 output
        TCCR0 = TIMER0_DIVIDER;

        //Enable all interrupts
        sei();

        //Continuously repeat content of while statement
        while(1)
        {
                //Repeatedly tun on LED one by one
                for(i=0b00000001;i<=0b01111111;i=(i<<1)+1){
                        PORTC = ~i;             //Active Low, output
reversed value of variable i

                        gucTimerTick = 100;     //Delay 200ms
                        while(gucTimerTick);
                }
        }

        //Return integer value to fit format of the function

        return 1;

}
```

Example Description

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. avr/interrupt.h header file has defined functions required for using the interrupt. Use #include statement to include the functions.

```
#define TIMER0_DIVIDER                  0b00000101
#define TIMER0_TCNT0_PRELOAD            6
```

TIMER0_DIVIDER & TIMER0_TCNT0_PRELOAD are used for setting Timer/Counter0 overflow interrupt activation cycle. #define is used to facilitate programming by using easily recognaizable names for the above items. Detailed explananation of TIMER0_DIVIDER & TIMER0_TCNT0_PRELOAD will follow later.

```
volatile unsigned char gucTimerTick;
```

Define global variable. Variable defined within the main function cannot be used in interrupt subroutine. Defining global variable will allow the variable to be used in main function and interrupt subroutine.

```
ISR(TIMER0_OVF_vect){
        TCNT0 = TIMER0_TCNT0_PRELOAD;

        if(gucTimerTick != 0){
                gucTimerTick--;
        }
}
```

Timer/Counter0 overflow interrupt subroutine. TCNT0 value is initialized to 0 when interrupt occurs. CNT0 = TIMER0_TCNT0_PRELOAD changes 0 to predefined value. If global variable gucTimerTIck has a value other than 0, subroutine is terminated after being reduced by 1.

```
unsigned char i;

DDRC = 0b01111111;
PORTC = 0b01111111;
```

Define 8bit variable i that will hold LED value, Set PORT C pins 0~6 for output, Set output value to 1. All LEDs will be turned off.

```
cli();

TCNT0 = TIMER0_TCNT0_PRELOAD;
TIMSK |= (1<<TOIE0);
TIFR |= (1<<TOV0);
TCCR0 = TIMER0_DIVIDER;

sei();
```

cli() is used to change SREG I bit to 0 to disable alll interrupts. Interrupts are disabled to prevent error by an interrupt while settings are being changed.

TCNT0 value is replaced by TIMER_TCNT0_PRELOAD value. As shown previously in the interrupt subroutine, TCNT0 value will always be initialized to TIMER_TCNT0_PRELOAD value. Timer/Counter0 overflow interrupt occurs when TCNT0 value increases from 0 to 255, replacing the TCNT0 value will change the Timer/Counter0 overflow interrupt period by changing the starting value of TCNT0 from 0 to TIMER_TCNT0_PRELOAD. Original overflow period of 256 tmer clock will change to (256–TIMER0_TCNT0_PRELOAD). In this example, overflow period will be 250. TIMSK changes TOIE0 value to 1. Above method can be used to change the register value instead of writing the value such as 0b00000001 directly into the register. avr/io.h header file contains information on which bit number each individual bit corresponds to. TOIE0 is defined as 0 since it is the 0 numbered bit of TIMSK. Therefore, TOIE0 bit of TIMSK can be set to 1 by using TIMSK |= (1〈〈TOIE0). TOV0 bit of TIFR can also be initialized to 0 by writing 1 to TOV0 bit. As OC0 output will not be used, WGM01:0, COM01:0 of TCCR0 becomes 0. Therefore, simply writing appropriate CS02:0 value to TCCR0 will activate the Timer/Counter in standard mode. TIMER0_DIVIDER is the CS02:0 value mentioned previously. In our example, 0b101 will be used to divide the default clock speed 16Mhz by 128. This will change the clock frequency to 125khz and TCNT0 will increase by 1 every 8us. With TCNT0 increasing every 8us, overflow interrupt occuring after 250, overflow interrupt period becomes 2ms. Overflow interrupt period can be changed as desired by changing the TIMER0_TCNT0_PRELOADand TIMER0_DIVIDER values. Use sei() command to enable all interrupts once the timer becomes active by writing value to the CS02:0 of TCCR0. Timer/Counter overflow interrupt will occur every 2ms.

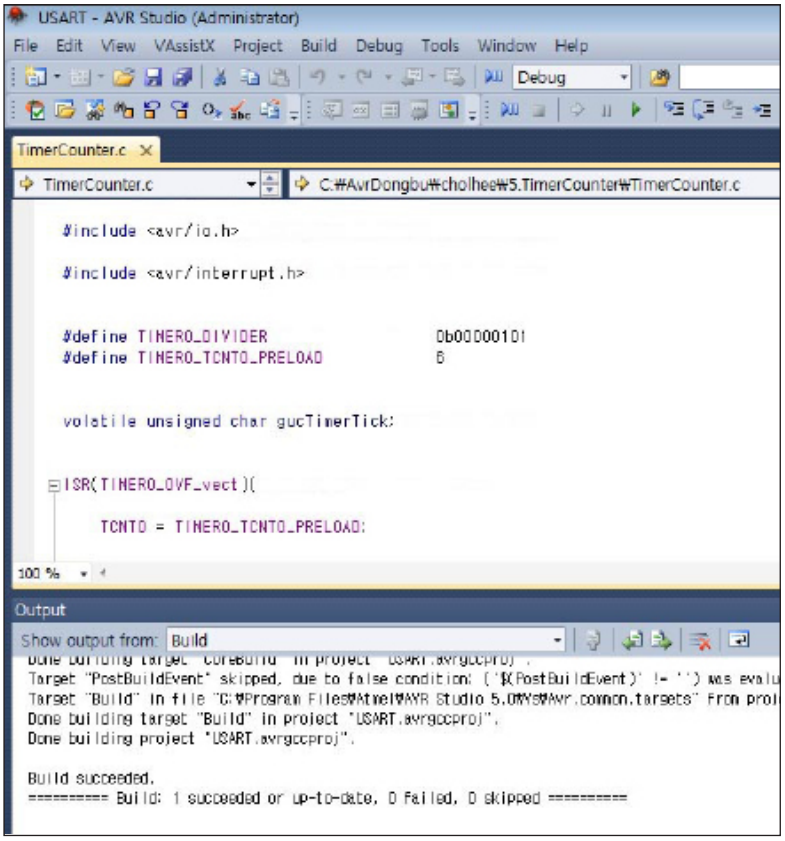
```
for(i=0b00000001;i<=0b01111111;i=(i<<1)+1){
        PORTC = ~i;

        gucTimerTick = 100;
        while(gucTimerTick);
}
```

Content of the for statement is same as in Chapter 1 except for one difference. In this for statement, _delay_ms(200); is replaced by gucTimerTick=100; and while(gucTimerTick);.When gucTimerTick is changed to 100, gucTimerTick will decrease by 1 every 2ms due to overflow interrupt service routine. Therefore, it will take 100 interrupt service routine for gucTimerTick to become 0. While(gucTimerTick); repeats the content of the while statement until gucTimerTick becomes 0 at which time program goes back to the beginning of the for statement. 200ms delay by _delay_ms function in the example in Chapter 1 has been replaced by the Timer/Counter0 overflow interrupt.

Run

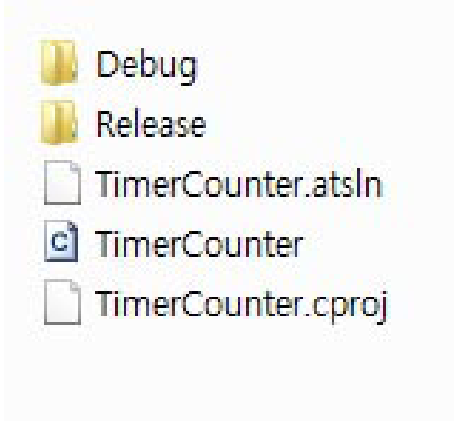
As in Chapter 1, LEDs will light up sequentially one by one from Servo LED to Power LED when program is loaded and executed. After all 7 LEDs have been lit LEDs will go off except for the Servo LED and sequence will start over again.



01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by 1 succeded 0 failed comment.





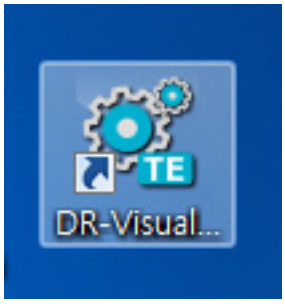
02 Debug Folder

After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.

03 Check Hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



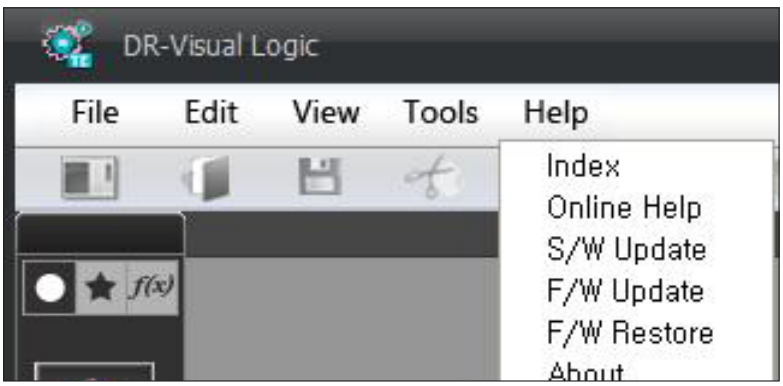
04 Visual Logic

Run DR-Visual Logic.



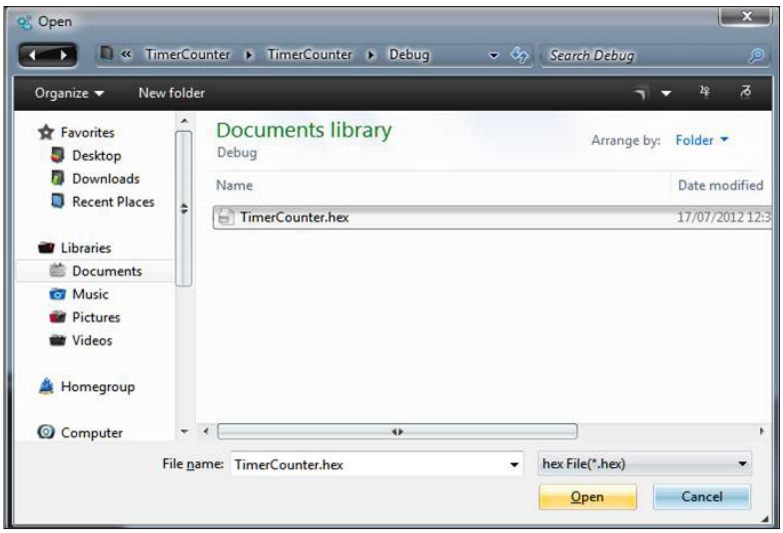
05 Connect

Select COM port and click Connect.



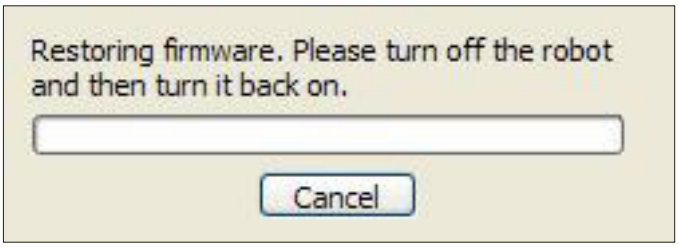
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



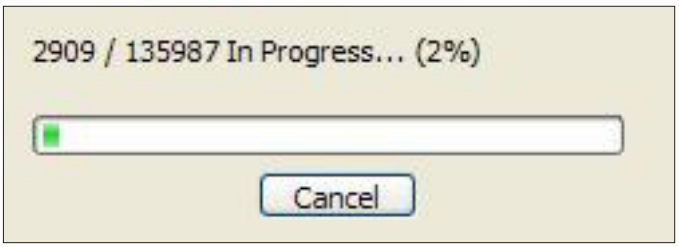
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.



10 Run

LEDs will light up sequentially one by one from Servo LED to Power LED. After all 7 LEDs have been lit, LEDs will go off except for the Servo LED and sequence will start over again.

Using ADC To Read PSD Sensor Value

Use ATmega128 ADC (Analog-to-Digital Converter) to read the analog PSD sensor output value.

AVR Fundamentals

ATmega128 contains 8 channel (ADC0~ADC7) 10 bit resolution successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port F. Sample/Hold circuit connected to ADC maintains the input analog signal constant during conversion. Atmega128 ADC supports 16 differential voltage input combinations including 7 differential analog input channels sharing common negative terminal (ADC1) and diffrential input mode with two channels (ADC1, ADC0 and ADC3, ADC2) that are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20dB (10x), or 46dB (200x) on the differential input voltage before the A/D conversion.

To reduce noise, ADC receives power from separate analog supply voltage pin, AVCC. AVCC must not differ more than±0.3V.

Registers related to ADC are as follows

ADMUX : ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Read/ Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- Bit 7:6 REFS1:0 : Reference Selection Bits

As ADC compares the intput analog voltage with the reference voltage VREF and converts the relative size to 10bit digital value, refence voltage VREF needs to be known to find the analog voltage from the digital value. Normally, analog voltage on the AREF pin becomes the refeence voltage VREF. Optinally, AVCC voltage or internal 2.56V maybe used as the reference voltage as well.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF Input Voltage
0	1	AVCC Input Voltage
1	0	NA
1	1	Internal 2.56V Voltage

– Bit 5 ADLAR : ADC Left Adjust Result

10 bit data converted by the ADC is saved in the ADC Data Register ADCH and ADCL. When ADLAR is 0, 8 low 8 bits are saved in ADCL and 2 high bits are saved in 1:0 bit of ADCH. When ADLAR is 1, 8 high bits are saved in ADCH and 2 low bits are saved in 7:6 bit of ADCL.

– Bit 4:0 MUX4:0 : Analog Channel and Gain Selection Bit

value of these 5 bits are used to select the type of input and gain to amplify the voltage for conversion. The table below shows the type of input according to the MUX4:0 value.

MUX4:0	SIngle Ended Input	Differential Input		
		Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	–		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	–	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	
11110	1.23V	–		
11111	0V			

■ ADCSR : ADC Control and Status Register

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

– Bit 7 ADEN : ADC Enable

This bit eanables or disables ADC. 1 to eanble ADC. 0 to disable ADC even during conversion.

– Bit 6 ADSC : ADC Start Conversion

This bit starts the ADC conversion. Set to 1 to start ADC conversion. When the value is set to 1 in single conversion mode, ADSC value wil remain as 1 during conversion and then change to 0 when the conversion ends. When the value is set to 1 in free running mode, conversion will start and continue performing successive conversions. ADC coversion normally requires 13 ADC clock cycle but the first conversion after the ADC is switched
on takes 25 ADC clock cycles in order to initialize the analog circuitry. ADC clock will be described more in detail in ADPS2:0 bit.

– Bit 5 ADFR : ADC Free Running Select

Free Running mode is selected by writing 1 to ADFR bit. When the bit is set to 1, ADC will continue the conversion and update the data register as long as ADSC bit is 1. When ADFR bit is set to 0, free running mode will be terminated and revert to single conversion mode.

– Bit 4 ADIF : ADC Interrupt Flag

ADIF bit is set to 1 when ADC conversion ends and the data register is updated. When SREG I bit and ADIE bit is set to 1, ADC conversion complete interrupt is activated. ADIF will automatically be initialized to 0 when corresponding interrupt subroutine starts to run. Write 1 to ADIF bit to change the ADIF bit to 0 manually through the software.

– Bit 3 ADIE : ADC Interrupt Enable

ADIE bit is used to enable the interrupt after the ADC conversion. ADC conversion complte interrupt is activated after the conversion when ADIE and SREG I bit is set to 1.

– Bit 2:0 ADPS2:0 : ADC Prescalar Select Bits

These bits determine the division factor between the ATmega128 clock frequency and the input clock to the ADC. ADC circuit requires input frequency of 50~200kHz to get maximum resoultion of 10 bits. Increase input frequency to achieve higer sampling rate if lower resoultion is required. ATmega128 ADC contains prescaler, which generates an acceptable ADC clock frequency of 50–200kHz from any clock frequency above 100kHz. User is able to select the ADC clock frequrency by setting the ADPS2:0 bits. For example, if ADC clock frequency of 125kHz is required, Atmega128 clock frequency of 16Mhz would be divided by 128.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

■ ADCL, ADCH : ADC Data Register

ADLAR = 0

Bit	15	14	13	12	11	10	9	8
ADCH	–	–	–	–	–	–	ADC9	ADC8
Read/ Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0
Bit	7	6	5	4	3	2	1	0
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Read/ Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

ADLAR = 0

Bit	15	14	13	12	11	10	9	8
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
Read/ Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0
Bit	7	6	5	4	3	2	1	0
ADCL	ADC1	ADC0	–	–	–	–	–	–
Read/ Write	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

– ADC9:0 : ADC Conversion Result

Result of the ADC converion can be read from the 2 registers ADCH and ADCL. In single eded conversion mode, saved data values 0∼1023 are positive integer, in differential mode, saved data values –512∼+511 are two's complement form. 10 bit value saved in ADCH and ADCL is called ADC. Refer to below for ADC formula accoring to the conversion mode.

Single Conversion Mode :

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

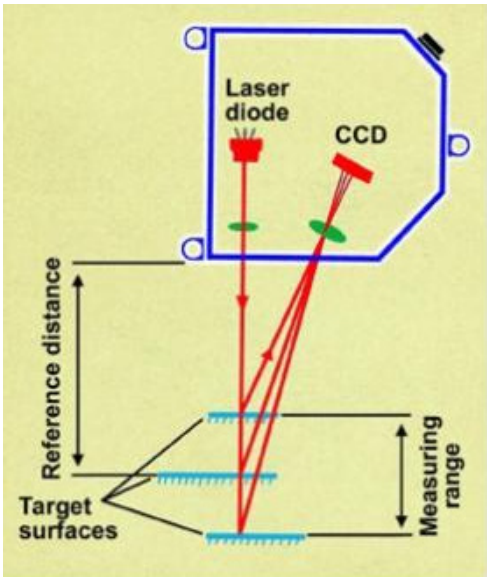
Differential Mode :

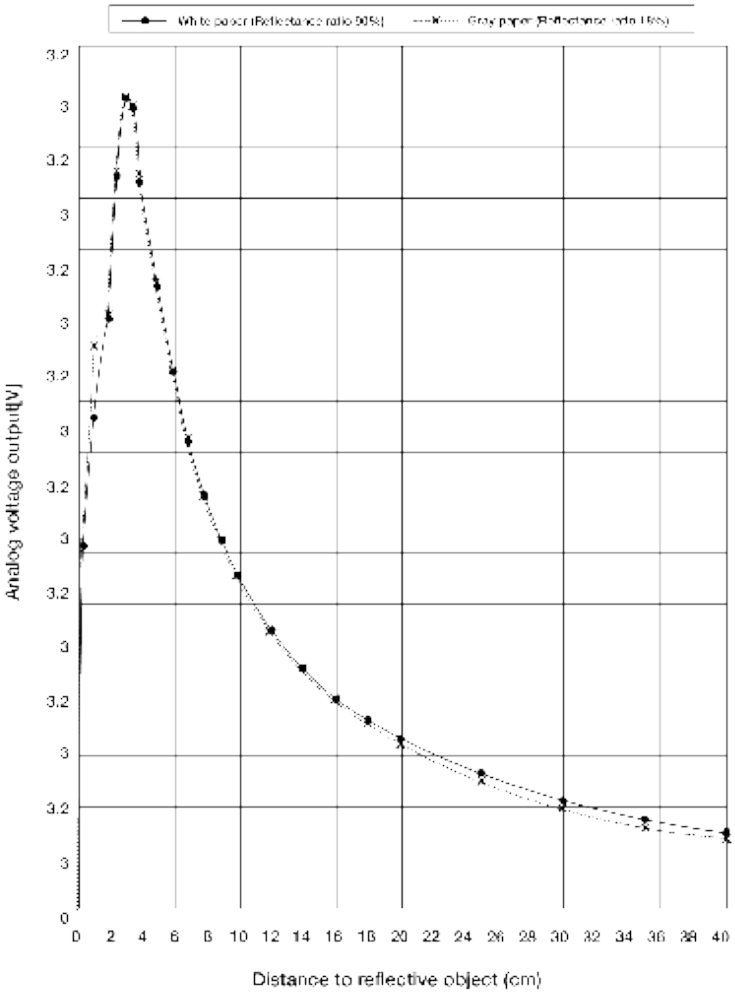
$$ADC = \frac{(V_{POS} - V_{NEG}) \times GAIN \times 512}{V_{REF}}$$

Set ADLAR to 1 and read and use only ADCH regsiter data when 8bit resolution rather than the 10bit resolution is enough to satisfy the requirement. However, if data from both registers are required, always read ADCL register first prior to reading ADCH register.

H/W Fundamentals

This example will conntect the analog PSD sensor module to the DRC and read the sensor outpupt value through ADC coversion. PSD sensor uses the IR light to measure the distance to the obstacle located at the front. Two round windows on the PSD sensor are for transmitting and receiving IR light. When transmitted IR light hits an object, it is reflected back to the receiver. As shown in the diagram below, incident angle of the reflected beam is determined by the distance from the sensor to the reflecting object. PSD sensor varies the output voltage used to measure the distance to an object according the angle of the reflected light. PSD sensors are divided between the analog and digital type with varying degree of range and widely used for measuring distance and detecting obstacles. The PSD sensor we will be using is an analog type with range of approximately 40cm. Greaph below (Graph 00) shows the output voltage related to the distance. Left DRC sensor port correspnds to ADC6(PF6) and the right sensor port corresponds to ADC7(PF7). Sensor values are read from ADC6 if analog PSD sensor is connected to the left sensor port.





Hardware Preparation



01 Connection

Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.



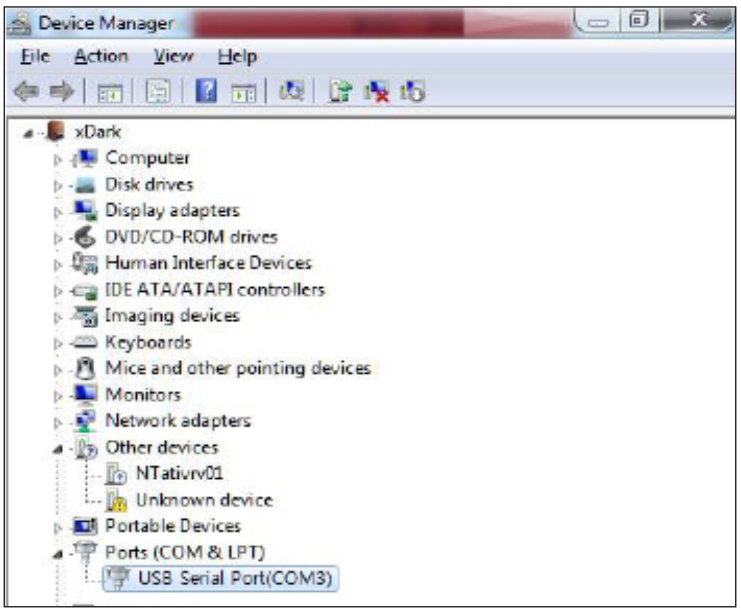
02 PSD Sensor Preparation

Prepare PSD sensor.



03 PSD Sensor Connection

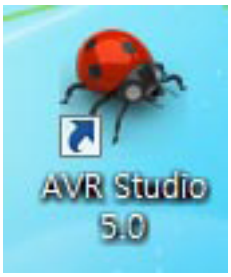
Connect the PSD sensor the the left side port on DRC



04 COM port Connection

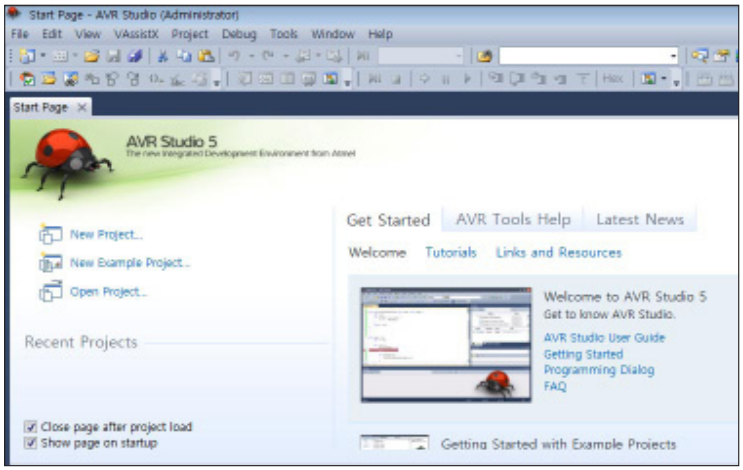
My Computer〉Property〉Device Manager. Check COM connection.

Running AVR Studio & Coding Preparation



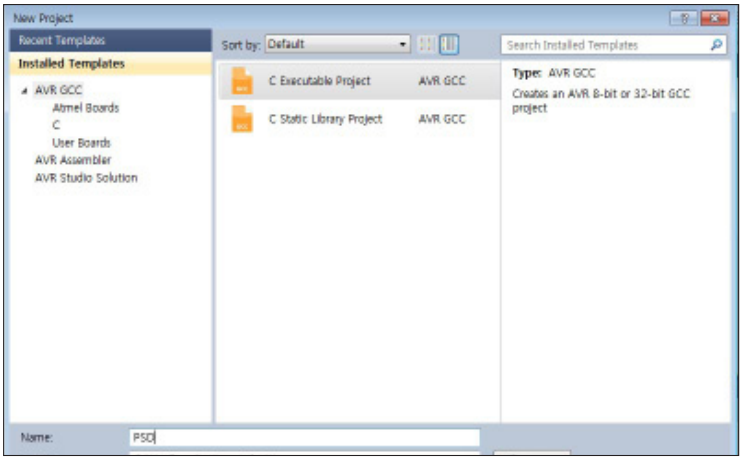
01 Run

Run AVR Stdio .



02 New Project

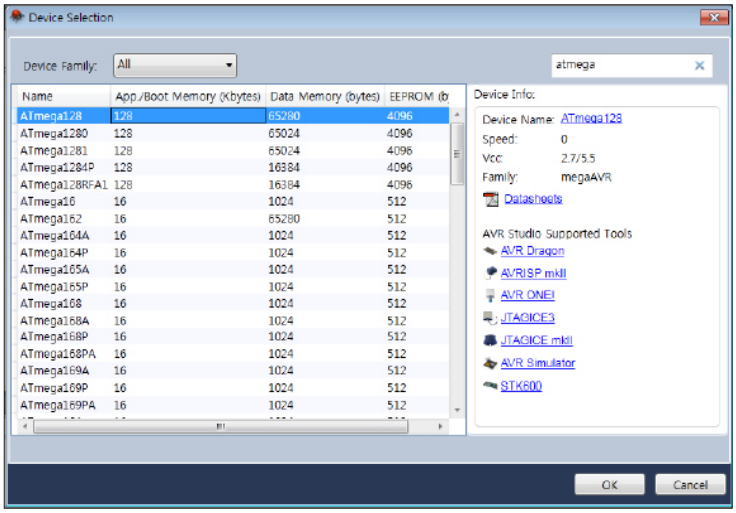
Click New Project .



03 Select GCC

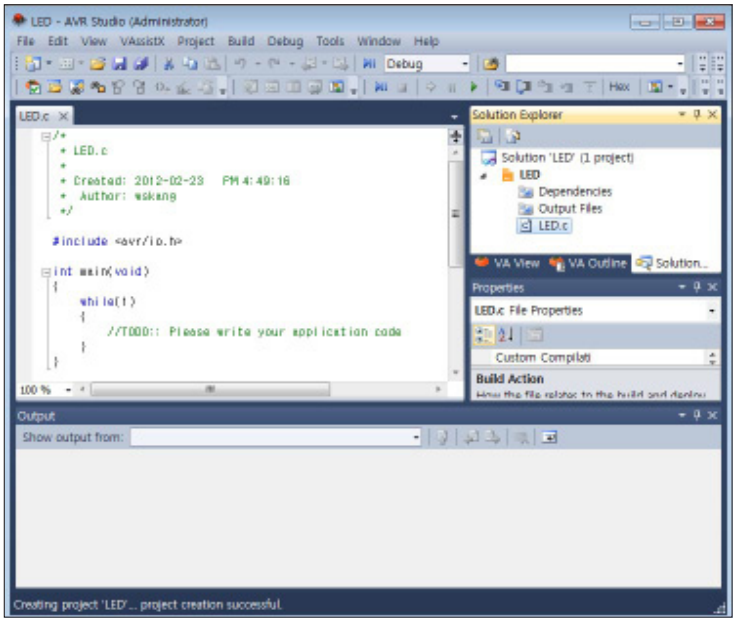
Select C Executable Project

Input 'PSD' as Name, Click Create directory for solution, Press OK button.



04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



05 Editor

Start coding in the edit window.
Follow the example below.

Example Description

Use ADC to convert analog PSD sensor output values from PSD connected to the left sensor port of DRC. LED will light up if the value is less than 10cm.

PSD.c

```
//ATmega128 registers defined
#include <avr/io.h>

int main(void)
{
        //Declare 16bit variable
        int iADC;

        //Set PORT C I/O direction
        DDRC = 0b01111111;
        //Set PORT C output value
        PORTC = 0b01111111;

        //VREF=AREF input voltage, turn off left adjust, ADC 6 single
ended input
        ADMUX=0b00000110;
        //ADC enable, Initialize ADIF , Clock division factor  128
        ADCSRA=0b10010111;

        //Continuously repeat conten of the while statement
        while(1)
        {
                //Start ADC conversion
                ADCSRA |= (1<<ADSC);
                //Wait for ADC conversion to finish
                while(!(ADCSRA & (1<<ADIF)));
                //Read and save ADCL & ADCH value
                iADC = ADCL | (ADCH<<8);
                //Initialize ADIF
                ADCSRA |= (1<<ADIF);

                // 1.3/5*1024 = 266.24
                if(iADC > 266)
                        PORTC = 0b00000000;
                else
                        PORTC = 0b01111111;
        }

        //Return integer value to fit format of the function
        return 1;
}
```

Example Description

```
#include <avr/io.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. Use #include statement to include the functions.

```
int iADC;

DDRC = 0b01111111;
PORTC = 0b01111111;
```

Declare 16bit variable iADC which will hold ADC value, Set PORT C 0~6 pins for output, set output value as 1. All LEDs will go off.

```
ADMUX=0b00000110;
ADCSRA=0b10010111;
```

Set registers to read analog input fromADC6 port in single ended ADC mode. Looking at ADMUX first, REFS1:0 bit is set to 0b00 as both the reference voltage and ATmega128 5V input power are connected to the AREF pin. Set ADLAR to 0 since left adjust will not be used. Set MUX4:0 bit to 0b00110 as ADC6 port will be read in single ended mode. In ADCSRA, set ADEN bit to 1 to enable ADC. Set ADSC and ADFR to 0 as coversion will not start yet and free running mode will not be used. Initialize ADIF to 0 by writing 1. Interrupt will not be used, set ADIE to 0. Set ADPS2:0 bit to 0b111 to to divide ATmega128 16MHz clock speed by 128, ADC clock speed will become 125kHz.

```
ADCSRA |= (1<<ADSC);
while(!(ADCSRA & (1<<ADIF)));
iADC = ADCL | (ADCH<<8);
ADCSRA |= (1<<ADIF);
```

ADC conversion will repeat continuously within the while statement. Set ADSCRA ADSC bit to 1 to start the conversion. Program will wait as long as ADIF bit is 0, satifying the !(ADCSRA & (1<<ADIF)) condition. But once ADIF bit changes to 1, program will exit the while statement and continue to the next step.

ADCL and ADCH values are read and saved to iADC once the ADC conversion ends and ADCL and ADCH values are updated and ADIF bit changes to 1. As ADCH corrsesponds to upper 1byte of ADC value, 〈〈(shift left) operation is peformed to left adjust 8bits and then ADCL |(bitwise OR) operation is peformed to fit 10bit data in iADC. ADCL was written first to read the ADCL register data first since ADCL has to be read prior to ADCH as explained previously. ADIF bit will not change to 0 automatically since ADC coversion complete interrupt is not being used. To initialize interrupt flag to 0 after reading the ADC value, 1 has to be written to ADCSRA ADIF bit.

```
if(iADC > 266)
        PORTC = 0b00000000;
else
        PORTC = 0b01111111;
```

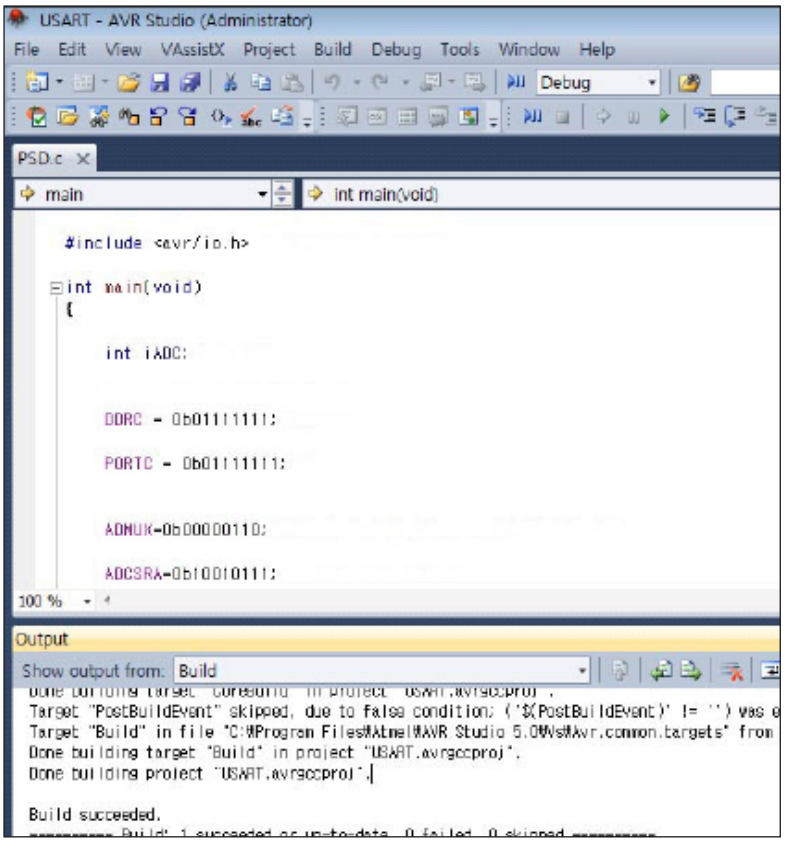
LED will light up according to the iADC value. According to the graph (diagram 00) showing the voltage in relation to PSD sensor distance, sensor output volatage is approximately 1.3V if the detected distanc to an object is 10cm. ADC output value of 1.3V is $ADC = \frac{V_N \times 1024}{V_{REF}}$.

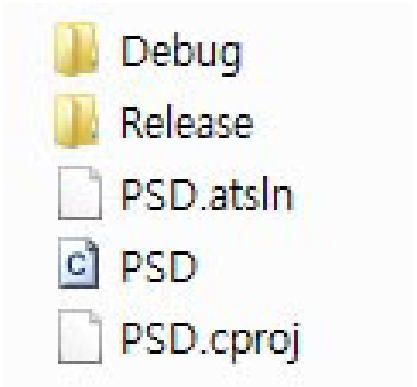
1.3×1024÷5 = 266.24. When input voltage is 1.3V ADC value will be 266.

When iADC value is larger tha 266, distance to an object is less than 10cm and vice versa. if–else statement will turn on all LEDs if an object is less than 10cm away and turn off all LEDs if it is farther than 10cm.

Run

All LEDs will light up when object (hand) is closer than 10cm from the PSD sensor. LEDs will go off if the object (hand) is farther tha 10cm away.







01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.

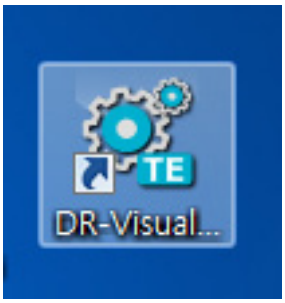
02 Debug Folder

After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



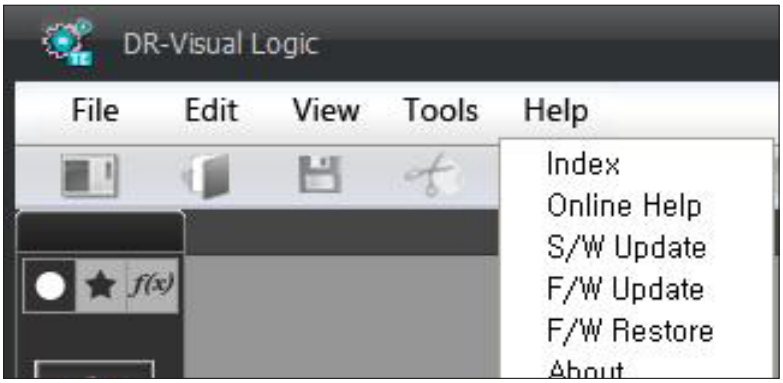
04 Visual Logic

Run DR-Visual Logic



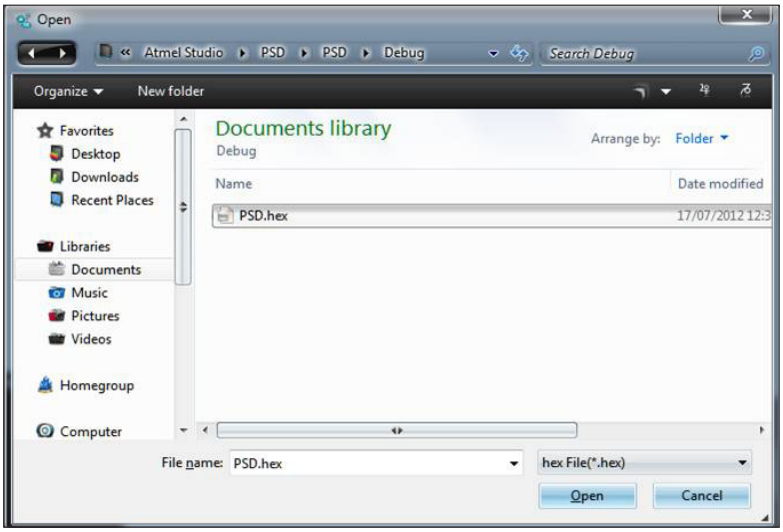
05 Connect

Select COM port and click Connect.



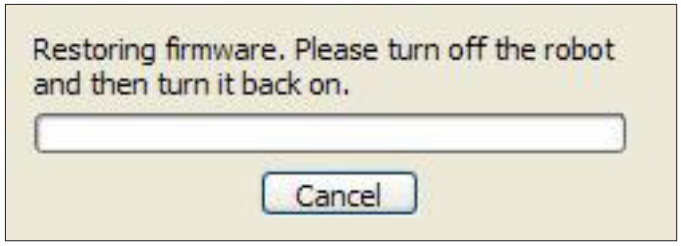
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



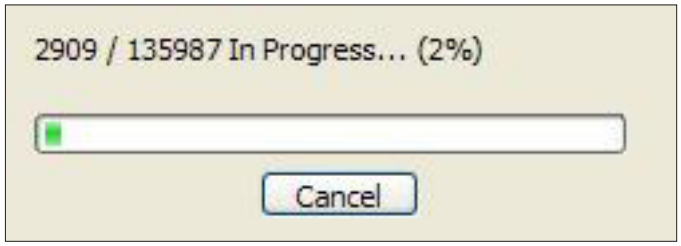
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.

(All LED OFF if no object is detectec withn 10cm)



(All LED on if object(hand) detected within 10cm)



10 Run

All LEDs will light up when object (hand) is closer than 10cm from the PSD sensor. LEDs will go off if the object (hand) is farther tha 10cm away.

Serial Communication Using USART

Serial communication between PC and the DRC using ATmega128 USART(Universal Synchronous and Asynchronous serial Receiver and Transmitter).

AVR Fundamentals

ATmega128 USART actually has two USART's USART0 and USART1. USART supports four modes of operation: Normal Asynchronous, Double Speed Asynchronous, Master Synchronous, and Slave Synchronous mode. Also as the name suggests, both synchronous and Asynchronous communciation is supported. The operational mode we will be using for this example will be Double Speed Asychronous communication. ATmega128 USART is largely composed of Clock Generator Block, Trasmitter Block, Receiver Block, and Register Block. Clock Generator Block generates clock required for USART communication. Internal clock is used for Asynchronous communication and Master Synchronous communication. Slave synchronous communication receives and synchronizs clock signal from the master for communication. Transmitter Block is used to serially transmit 8bit parallel data saved in the register. Receiver Block converts the received serial data into parallel data and saves the converted data in the reigster. Register block is used for various USART control bits, setup bits, and data bits.

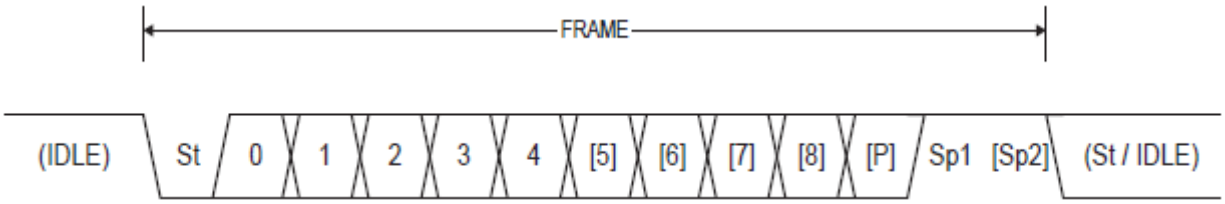


Diagram 00 shows the USART serial frame. Serial frame is composed of data bit, synchronization bits (start bit and stop bit), and optional parity bit. USART accepts all 30 combinations of the following as valid frame formats:

– 1 start bit

– 5, 6, 7, 8 or 9 data bit

– no, even or odd parity bit

– 1 or 2 stop bits

The most often used frame format conisists of 8 data bit, no parity bit, 1 stop bit. This format is is used by DRC and HekuleX as well.

Communication line is held HIGH during the IDLE state. Start bit (St) is always LOW. Data bit (0~[8]) is transmitted after the start bit. Depending on the format, data bit can contain 5–9 bits. Parity bit ([P]) may follow the data bit if parity bit is enabled. Stop bit (Sp1~[Sp2]) is sent last. Completed frame maybe followed by another frame or communication line may go into IDLE state.

The most commonly used format has 10 bits total with 1 start bit + 8 data bit + 1 stop bit.
USART related registers are shown below. n in the register name can be either 0 or 1 depending on whether the register is related to USART0 or USART1.

UDRn : USART I/O Data Register

Bit	7	6	5	4	3	2	1	0
Read	RXBn[7:0]							
Write	TXBn[7:0]							
Read/ Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

USARTn transmit data buffer register and USARTn receve data buffer register share the same data register named UDRn. Data written to UDRn will be saved in transmit data buffer register and the data read from UDRn will be from receive data buffer register. Data can be written to transmit data buffer register only when UDREn bit in UCSRAn is set to 1. When data is written with the transmitter enabled, data will be sent to transmit shift register and transmitted serially one bit at a time. Receive data register is composed of 2 level FIFO which changes to the next data value every time it is accessed. To read data from the UDRn or to access paticular bit from RXBn[7:0] it is safer to save the read data from UDRn into another variable first.

UCSRnA : USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Read/ Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

– Bit 7 RXCn : USART Receive Complete

RXCn is used to show the status of receive data register. Bit is set to 1 when data is received in the received data register and set to 0 when the register is empty. RXCn can also be used to activate USARTn receive complete interrupt.

– Bit 6 TXCn : USART Transmit Complete

TXCn is used to show the status of transmit data register. TXCn is set to 1 when all data in transmit data register is transmitted. TXCn can also be used to activate USARTn transmit complete interrupt. Bit is initalized to 0 when interrupt subroutine starts to execute. Bit can be manually initialized to 0 by writing 1.

– Bit 5 UDREn : USART Data Register Empty

UDREn bit is used to show that UDRn is ready to receive new data. Data written to UDRn will be transmitted when UDREn is 1. UDREn can also be used to activate USARTn data register empty interrupt.

– Bit 4 FEn : Frame Error

This bit is used to show if the next character in the receive buffer had a Frame Error when received. First stop bit of the data frame should always be 1. FEn bit will be set to 1 if the first stop bit of the frame is 0.

– Bit 3 DORn : Data OverRun

DORn bit is used to show over run error while receiving data . Bit is set when new start bit is detected in the receiver pin when 2 level FIFO in receive buffer already holds 2 characters.

– Bit 2 UPEn : Parity Error

UPEn is used to show parity error during data transmission. This error only occurs when parity is enabled by setting the UPMn1:0 bit in UCSRnC register.

– Bit U2Xn : Double USART Transmission Speed

U2Xn bit only works for the asynchronous operation. Set U2Xn to 0 when using synchronous communication.
Writing 1 to U2Xn will reduce the clock divider from 16 to 8 which will double the USART communication speed even with same UBRRL and UBRRH value. Clock divider will remain at 16 if U2Xn bit is set to 0.

– Bit 0 MPCMn : Multi Processor Communication Mode

MPCMn bit is used to activate the multi processor communications mode. When MPCMn is set to 1, any data received received by USARTn without address information will be ignored. This mode is used for controlling multiple processors with single communications line.

■ UCSRnB : USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
Reat/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

– Bit 7 RXCIEn : Enable Receive Complete Interrupt

Set RXCIEn bit to 1 to enable interrupt on the RXCn. USARTn receive complete interrupt will only be activated if SREG I bit and RXCIEn bit is set to 1 and RXCn flag on UCSRnA is 1.

– Bit 6 TXCIEn : Enable Transmission Complete Interrupt

Set TCXIEn bit to 1 to enable interrupt on the TXCn. USARTn transmission complete interrupt will only be activated if SREG I bit and TXCIEn bit is set to 1 and TXCn flag on UCSRnA is 1.

– Bit 5 UDRIEn : Enable Data Register Empty Interrupt

Set UDRIEn bit to 1 to enable interrupt on UDREn. Data register empty interrupt will only be activated if SREG I bit and UDRIEn bit is is set to 1 and UDREn flat on UCSRnA is 1.

– Bit 4 RXENn : Enable Receiver

Set RXENn bit to 1 to enable USARTn receiver block. When RXENn bit is set to 1, RxDn pin on ATmega128 will operate as serial data receiver pin regardless of the I/O port setting.

– Bit 3 TXENn : Enable Transmitter

Set TXENn bit to 1 to enable USARTn transmitter block. When TXENn bit is set to 1, TxDn pin on ATmega128 will operate as serial data transmitter pin regardless of the I/O port setting. When 0 is written to the TXENn during data transmission, tranmission pin will become disabled after completing the current transmission.

– Bit 2 UCSZn2 : Character Size

UCSZn2 bit and UCSZn1:0 bit in UCSRnC are used to set the data size. Refer to UCSZn1:0 bit section for more details.

– Bit 1 RXB8n : Receive Data Bit 8

RXB8n is the 9th bit of the received data when operating with serial frames with 9 data bits. RXB8n must be read before reading the lower RXBn[7:0] bits from UDRn.

– Bit 0 TXB8n : Transmit Data Bit 8

TXB8n is the 9th bit of the data to be transmitted when operating with serial frames with 9 data bits. TXB8n must be written prior to writing lower RXBn[7:0] bits to UDRn.

UCSRnC : USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	1	1	0

– Bit 6 UMSELn : USART Mode Select

UMSELn bit is used to select between Asynchronous and Synchronous mode of operation. USART is set to Asynchronous mode when bit is 0 and Synchronous mode when bit is 1.

– Bit 5:4 UPMn1:0 : Parity Mode

UPMn1:0 bits are used to set parity generatlon and parity check. When parity is enabled, transmitter block will automatically generate and send the parity bit behind the data bit. Reciver block will check the parity of the received data. When partity error is detected, UPEn flag in UCSRnA register is set to 1.

UPMn1	UPMn0	Parity Mode
0	0	No Parity Bit
0	1	(Not Used)
1	0	Even Parity Bit
1	1	Odd Parity Bit

– Bit 3 USBSn : Select Stop Bit

USBSn bit is used to select the number of stop bits. 1 bit when USBSn bit is 0, 2 bits when USBSn is 1.

– Bit 2:1 UCSZn1:0 : Character Size

UCSZn1:0 bit together with UCSZn2 bit in UCSRnB are used to set the frame data bit size.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5 Bit
0	0	1	6 Bit
0	1	0	7 Bit
0	1	1	8 Bit
1	0	0	(Not Used)
1	0	1	(Not Used)
1	1	0	(Not Used)
1	1	1	9 Bit

– Bit 0 UCPOLn : Clock Polarity

UCPOLn bit is used only in synchronous mode. Set bit to 0 in asynchronous mode. UCPOLn bit is used to set the relationship between data output change and data input sample, and the synchronous clock (XCKn).

UCPOLn	Outpu Data Changing Time	Received Data Sampling Time
0	When Rising Edge	When Falling Edge
1	When Falling Edge	When Rising Edge

UBRRnL, UBRRnH : USART Baud Rate Registgers

Bit	15	14	13	12	11	10	9	8
UBRRnH	–	–	–	–	UBRRn[11:8]			
Read/ Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0
Bit	7	6	5	4	3	2	1	0
UBRRnL	UBRRn[7:0]							
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

– 11:0 UBRRn11:0 : USARTn Baud Rate Register

This 12 bit register is used to set the baud rate or communication speed of the USARTn. USART communication speed is shown by the baud rate bps(bit per second). Commonly used baud rate are 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800, 921600 bps. DRC and HerkuleX uses 115200 bps as the standard baud rate. Baud rate and UBRR formula for each communication mode and U2Xn bit value are shown below.

Operation Mode	Baud Rate Formula	UBRR Formula
Standard Asynchronous Mode(U2Xn = 0)	$BAUD = \frac{f_{osc}}{16\ (UBRR+1)}$	$UBRR = \frac{f_{osc}}{16\ BAUD} - 1$
2X Speed Asynchronous Mode(U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Master Synchronous Mode	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

BAUD refers to Baud Rate, fOSC refers to system clock frequency, and UBRR refers to UBRR register value.

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1 Mbps		2 Mbps	

Table above shows what UBRR value to set to generate commonly used baud rate with DRC system clock frequency of 16MHz. To generate DRC & HerkuleX standard baud rate of 115200bps, set UBRR to 8 in standard mode and to 16 in 2x speed mode. DRC & HerkuleX uses 16 in 2x speed mode which has smaller error.

H/W Fundamentals

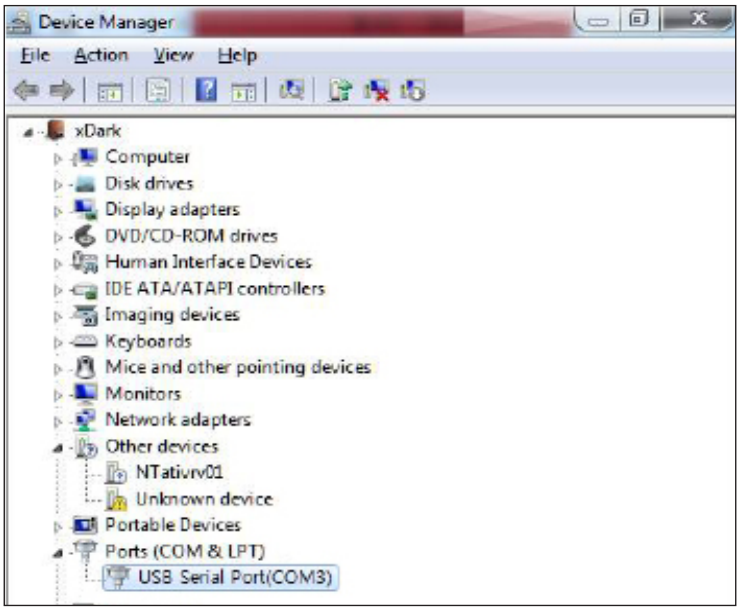
As mentioned previously, ATmega128 has 2 USART ports, USART0 and USART1. USART0 is connected in parallel format to 5 4pin ports which are used to communicate with the HerkuleX. USART1 is connected to PC through the earphone jack and connected to the Zigbee module internally. USART output signals from ATmega128 are either digital HIGH or digital LOW with LOW signal having 0V voltage level and HIGH signal having 5V voltage level. This type of level signal is called TTL signal. Ouput signals from PC Com ports and USB-to-Serial zender normally have voltage level of -5~-10V when the signal is LOW and +5~+10V when the signal is HIGH. This type of signal is said to have RES-232 level. The difference in signal level prevents communication even though same data is being transmitted. This problem is overcome by adding serial communication level conversion chip to the DRC circuit to convert TTL signal from ATmega128 to RS-232 before being tranmitted to the PC and converting RS-232 signal from PC to TTL before being transmitted to ATmega128.



Hardware Preparation

01 Connection

Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.



02 COM Port Connection

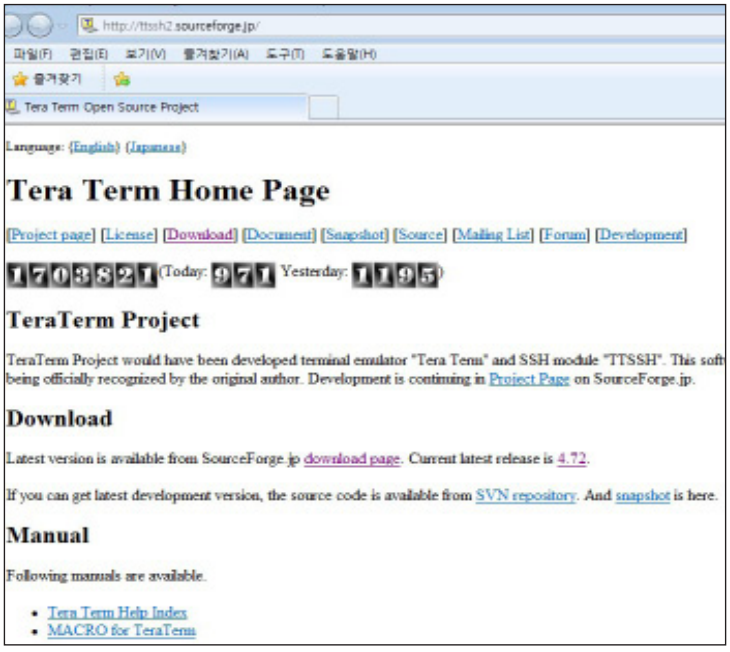
My Computer〉Property〉Device Manager. Check COM connection.

Software Preparation



01 Address Search

Use internet search engine to search for “tera term”



02 Homepage Connection

Connect to Tera Term home page and click “Download page”.



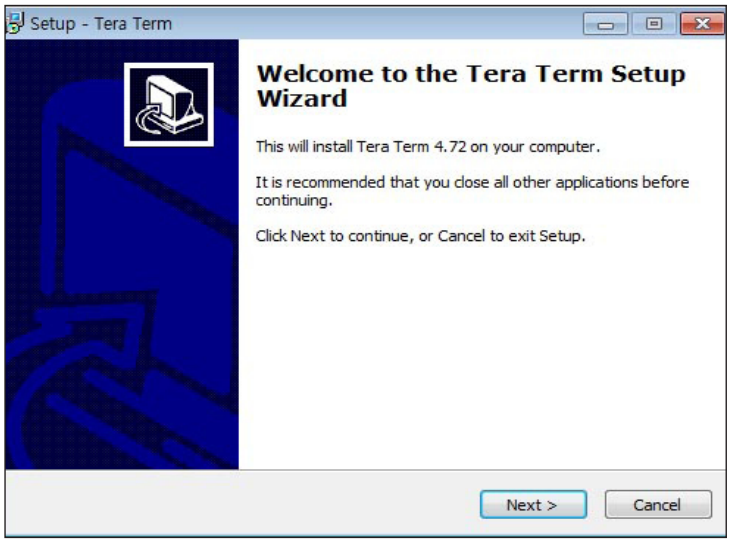
03 File Download

Click “Save” to download and save the file.



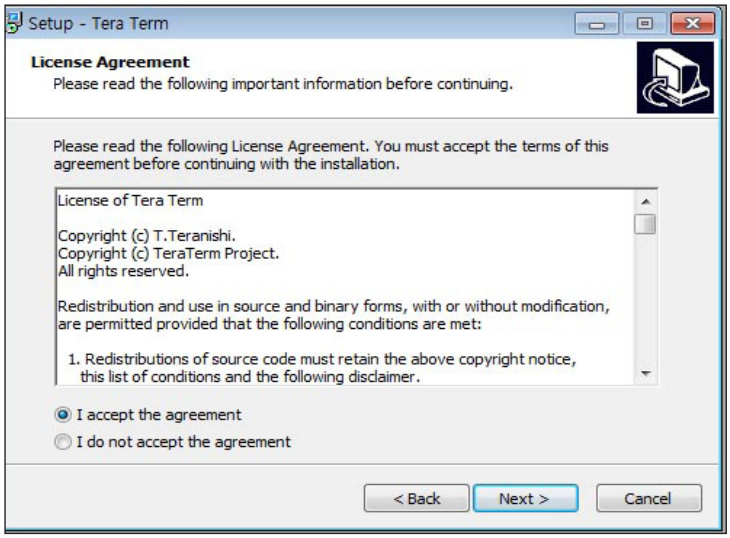
04 Start File Download

Setup file download in progress.



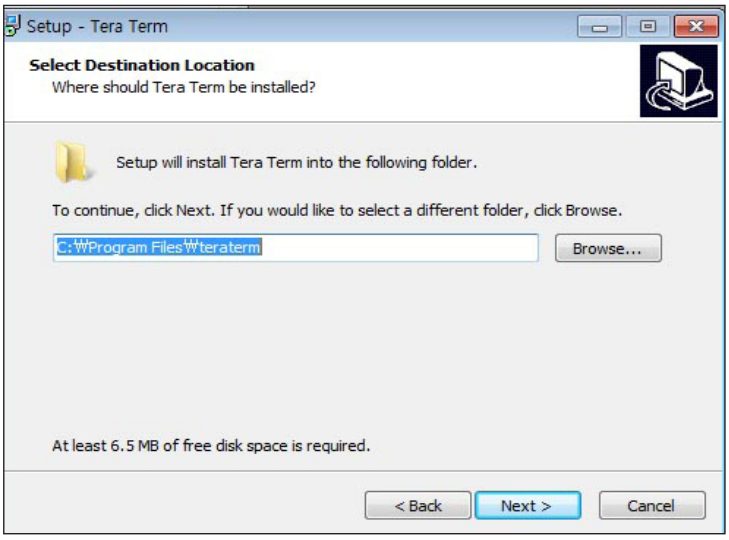
05 Run Setup Wizard

Click “Next” to run setup wizard.



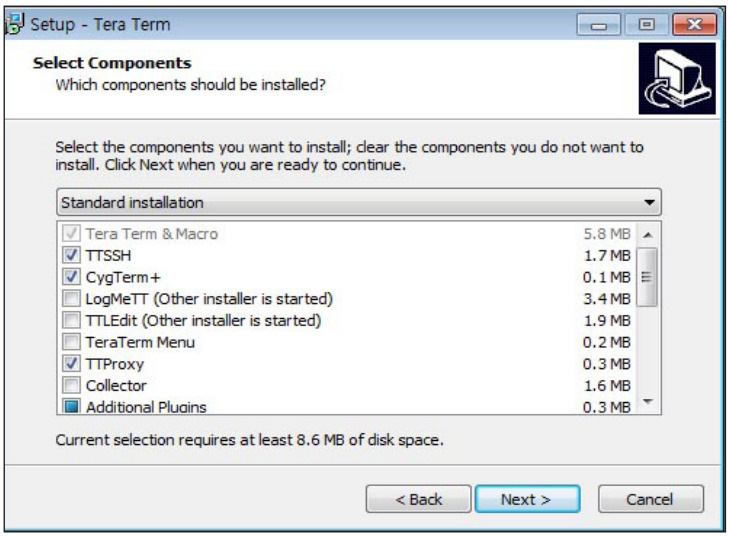
06 License Agreement

Click “I accept the agreement” and then click “Next”.



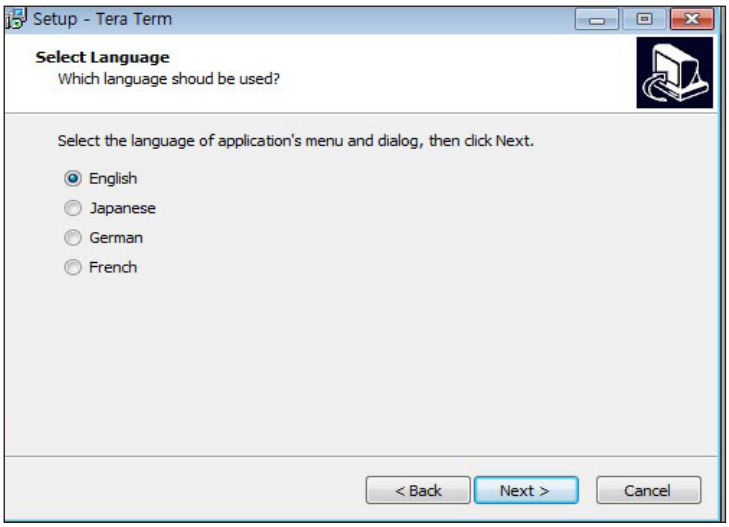
07 Confirm installation folder

Confirm program installation folder and then click “Next”.



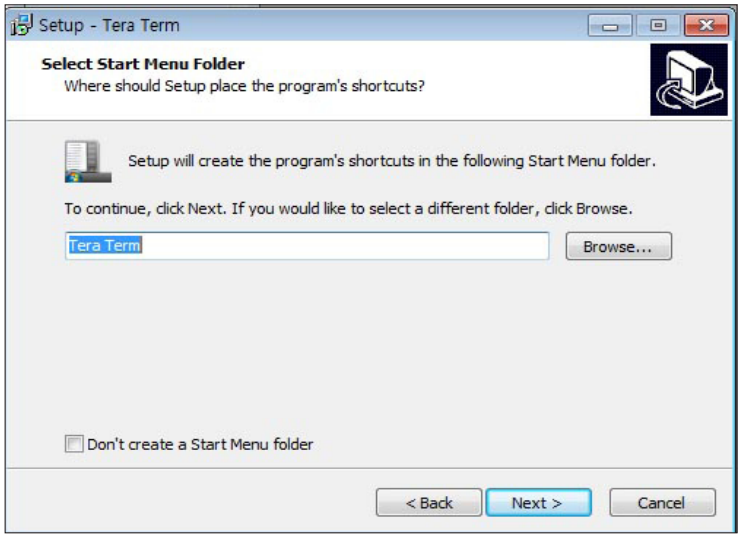
08 Select Componets

Click “Next”.



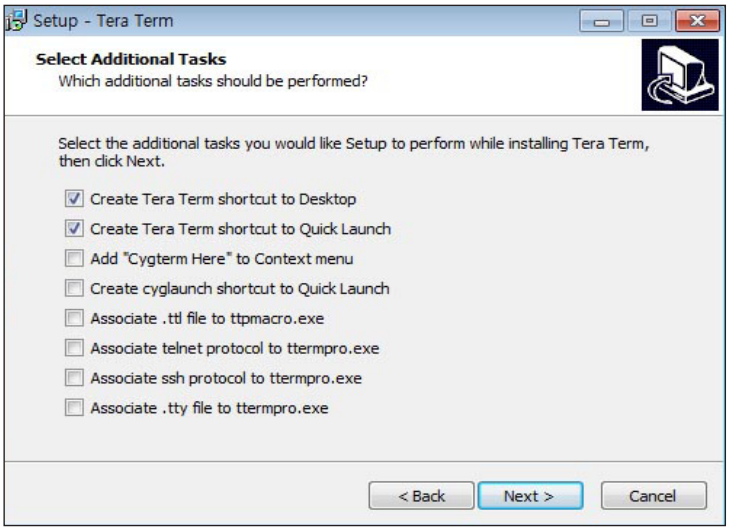
09 Select Language

Select desired language and then click “Next”.



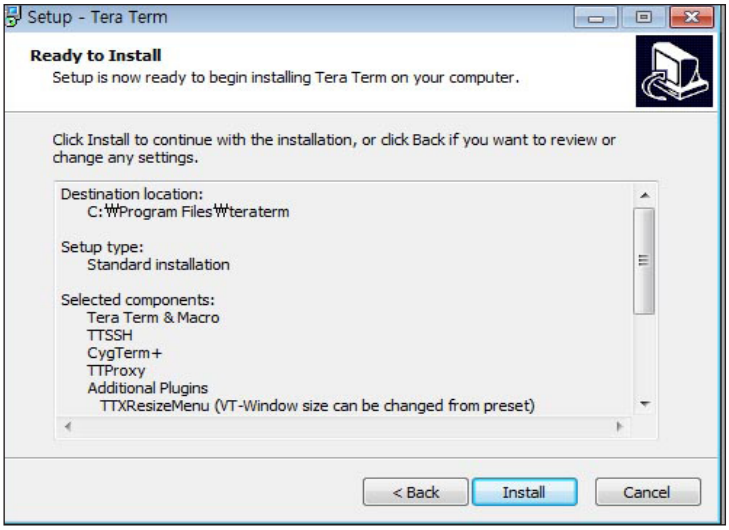
10 Select Start Menu Folder

Select start menu folder and then click "Next".



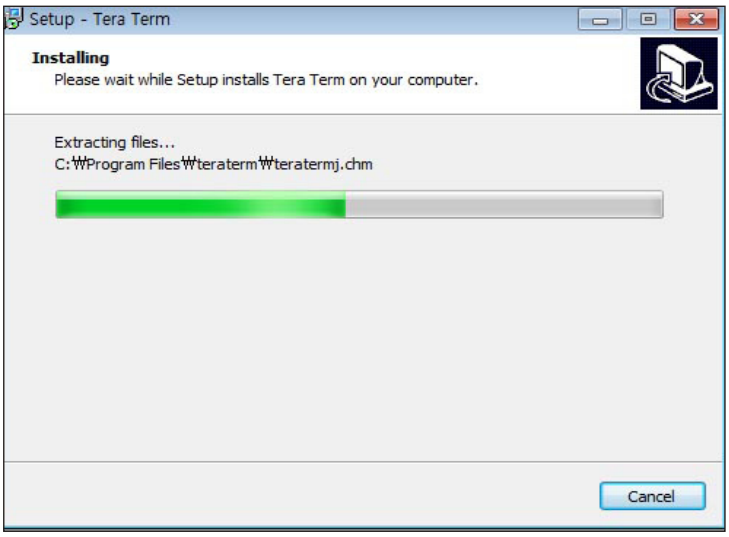
11 Select Additional Tasks

Select additional tasks and then click "Next".



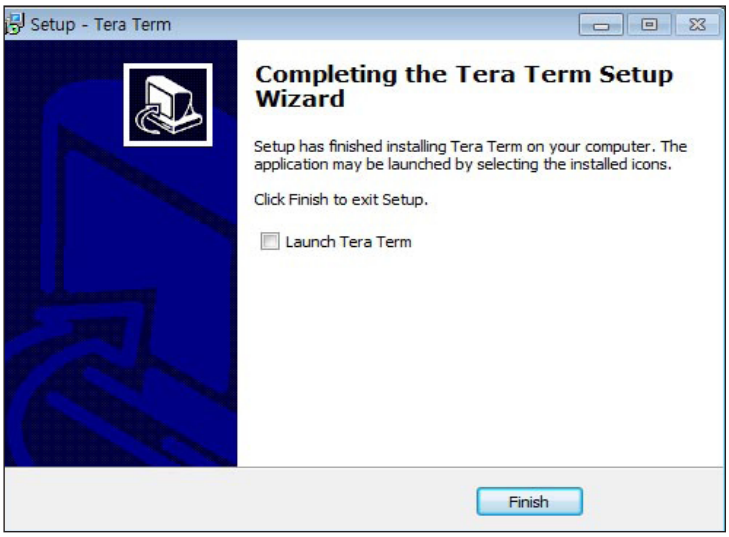
12 Start Installation

Click "Install" to start the installation.



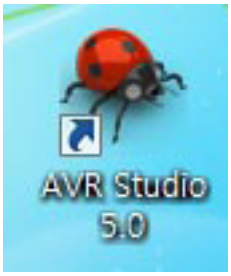
13 Confirm Installation

Program installation in progress.



14 Installation Complete

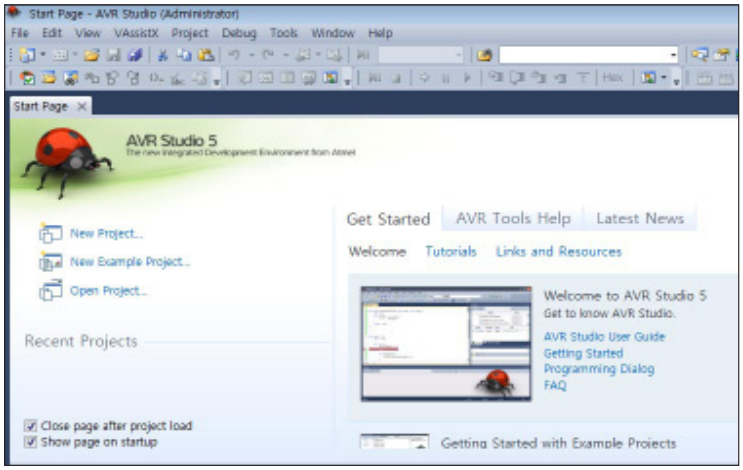
Click "Finish" to complete the installation.



Running AVR Studio & Coding Preparation

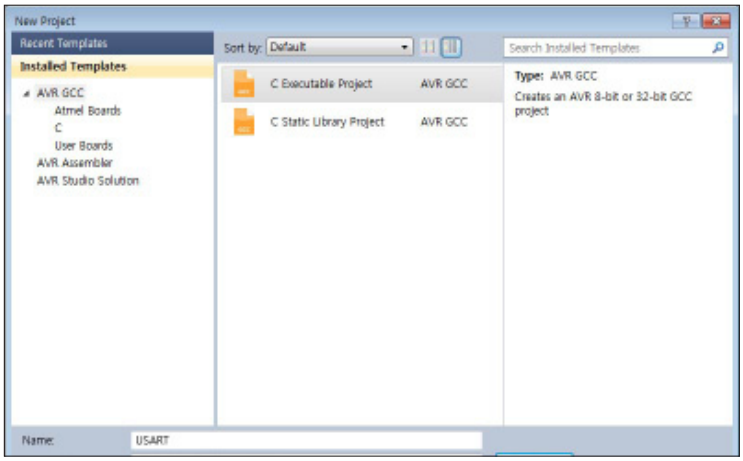
01 Run

Click icon to run AVR Studio.



02 New Project

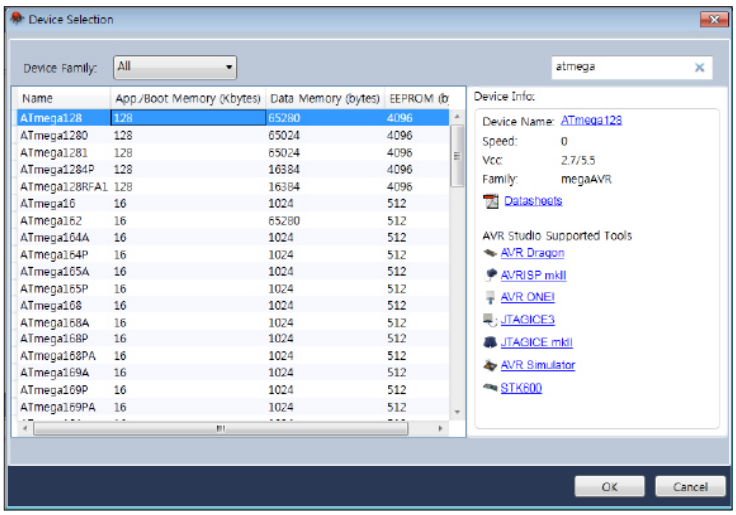
Click New Project.



03 Select GCC

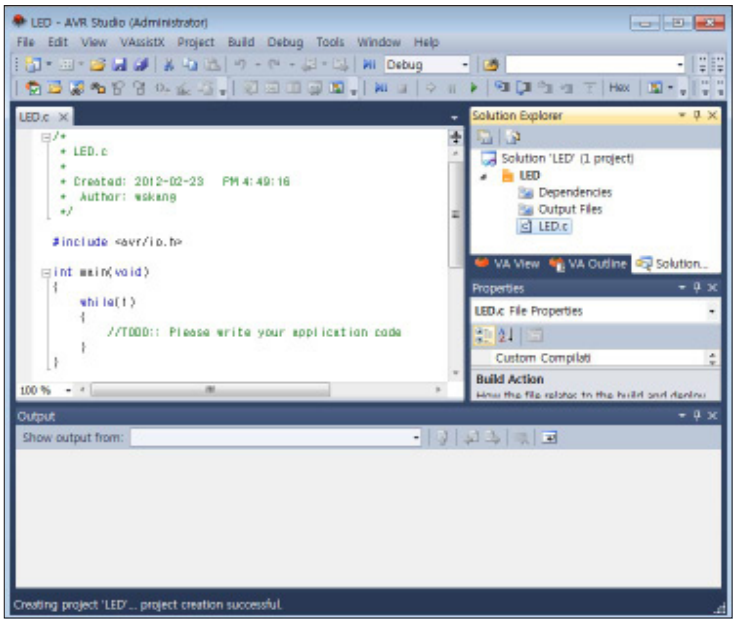
Select C Executable Project

Input 'USART' as Name, Click Create directory for solution, Press OK button.



04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



05 Editor

Start coding in the edit window.
Follow the example below.

Example

In this example, characters sent by using hyperterminal will be used control the LED on DRC. DRC is connected to the PC by USB-to-Serial cable.

USART.c

```
//Set CPU Clock
#define F_CPU 16000000

//ATmega128 registers defined
#include <avr/io.h>
//Information requred for using interrupt defined
#include <avr/interrupt.h>
//_delay_ms function defined
#include <util/delay.h>

//Define Port C pin corresponding to LED
#define LED_MASK 0b01111111

//Declare global variable to use when receiving data from
USART1
volatile unsigned char gucRxBuffer;
volatile unsigned char gucRxFlag;

//USART1 Receive complete interrupt service routine
ISR(USART1_RX_vect){
        gucRxBuffer = UDR1;
        gucRxFlag = 1;
}

//Function for initializing USART1
void USART1_Init(unsigned long ulBaudRate){
        unsigned long ulTemp;

        //Delay to prevent collision with bootloader
        _delay_ms(10);

        //Enable transmit and receive interrupt
        UCSR1B = ((1 << TXEN1) | (1 << RXEN1) | (1 << RXCIE1));
        //set 2x speed mode
        UCSR1A = (1 << U2X1);

        //Use clock and Baud Rate to calculate value for UBRR1
        ulTemp = F_CPU / (ulBaudRate * 8L) - 1;
```

```
        //set calculated value to UBRR1
        UBRR1H = (ulTemp >> 8) & 0x0F;
        UBRR1L = ulTemp & 0xFF;

        //Initialize global variable
        gucRxBuffer = 0;
        gucRxFlag = 0;

        return;
}

//Function for sending 1 character by USART1
void USART1_PutChar(char cData){
        //Wait until transmit data register is empty
        while(!(UCSR1A & (1 << UDRE1)));
        //Write data to transmit data register
        UDR1 = cData;

        return;
}

//Function fro sending character string by USART1
void USART1_PutString(char* pcStr){
        //Repeat USART1_PutChar until '₩0' met
        while(*pcStr)
                USART1_PutChar(*pcStr++);
}

int main(void)
{
        //Define 8bit variable
        unsigned char ucLED;

        //Set Port C I/O direction
        DDRC = 0b01111111;
        //Set Port C output value
        PORTC = 0b01111111;

        //Disable all interrupt
        cli();

        //Initalize USART1
        USART1_Init(115200);
```

```
        //Enable all interrupt
        sei();

        //Transmit initialization complete message
        USART1_PutString("Serial Initialized₩r₩n");

        //Turn on Servo LED
        ucLED = 0b00000001;
        PORTC = (~ucLED) & LED_MASK;

        //Continuosly repeat conetent of while statement
        while(1)
        {
                //Run when data received
                if(gucRxFlag){
                        //Run if received data is 'z' or 'Z'
                        if(gucRxBuffer == 'z' || gucRxBuffer == 'Z'){
                                //Shift LED bit to be lit to the right
                                if(ucLED > 0b00000001){
                                        ucLED >>= 1;
                                }
                                else{
                                        ucLED = 0b01000000;
                                }
                                PORTC = (~ucLED) & LED_MASK;

                                //Transmit message
                                USART1_PutString("LED Moved by ₩'z₩'₩r₩n");
                        }
                        //Run if received data is  'x' or 'X'
                        else if(gucRxBuffer == 'x' || gucRxBuffer == 'X'){
                                //Shift LED bit to be lit to the left
                                if(ucLED < 0b01000000){
                                        ucLED <<= 1;
                                }
                                else{
                                        ucLED = 0b00000001;
                                }
                                PORTC = (~ucLED) & LED_MASK;

                                //Transmit Message
                                USART1_PutString("LED Moved by ₩'x₩'₩r₩n");
                        }
                        //Initialize data receive flag
                        gucRxFlag = 0;
                }
        }

        //Return integer value to fit format of the function
        return 1;
}
```

Example Description

```
#define F_CPU 16000000
```

#define statement is one of the preprocessor commands, #define command will replace any F_CPU found later with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on the clock speed, this command sets the clock speed so that delay function and other fuctions execute properly.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. avr/interrupt.h header file has defined functions required for using the interrupt. util/delay.h header file has _delay_ms() and other commonly used delay functions defined. Use #include statement to include the functions.

```
#define LED_MASK                    0b01111111
```

#define LED_MASK 0b01111111 will replace any LED_MASK found with 0b01111111 before compiling. All DRC LED bits will be replaced by LED_MASK to facilitate programming.

```
volatile unsigned char gucRxBuffer;
volatile unsigned char gucRxFlag;
```

Define global variable to be used in interrupt subroutines and in functions. Variable gucRxBuffer is used to hold the received USART1 data register value. Variable gucRxFlag is ia flag variable showing data has been received by the register.

```
ISR(USART1_RX_vect){
        gucRxBuffer = UDR1;
        gucRxFlag = 1;
}
```

USART1 receive complete interrupt subroutine. Data received from UDR1 will be read and saved in gucRxBuffer and gucRxFlag will be set to 1. Data and the flag will be used in main function.

```
void USART1_Init(unsigned long ulBaudRate)
```

Function used for initializing the USART1. Baud rate received in parameter ulBaudRate is used to calculate the UBRR1 value.

```
unsigned long ulTemp;
```

Declare variable FulTemp to hold UBRR value calculated with ulBaudRate.

```
_delay_ms(10);
```

As DRC bootloader also uses USART1 for firmware fusing, communication error can occur if USART1 setting changes before bootloader completes transmitting the data. _delay_ms(10); provides about 10ms delay to give bootloader enough time to complete the data transmission.

```
UCSR1B = ((1 << TXEN1) | (1 << RXEN1) | (1 << RXCIE1));
UCSR1A = (1 << U2X1);
```

Set UCSR1B and UCSR1A registers. Enable transmit, receive, and transmit interrupt. Set U2X1 to 1 to use 2x speed asynchronous mode. UCSR1C register does not need separte setup since inital value will be used.

```
ulTemp = F_CPU / (ulBaudRate * 8L) - 1;


UBRR1H = (ulTemp >> 8) & 0x0F;
UBRR1L = ulTemp & 0xFF;
```

$$UBRR = \frac{f_{osc}}{8BAUD} - 1$$

Calculate UBRR1 value using ulBaudRate and save the value in ulTemp. Use the formulat above for calculation. Save the upper 4bits of ulTemp value in UBRR1H and lower 8bits in UBRR1L. USART1 will now operate with the baud rate in ulBaudRate.

```
gucRxBuffer = 0;
gucRxFlag = 0;
```

Initialize global value to 0.

```
void USART1_PutChar(char cData){
        while(!(UCSR1A & (1 << UDRE1)));
        UDR1 = cData;
return;
}
```

USART1_PutChar function is used to send 1byte of data through USART1. As data transmission can only take place if UDRE1 bit of UCSR1A is 1, transmission will be delayed until UDRE1 bit becomes 1 if some other data is in the midst of being transmitted. while(!(UCSR1A & (1 << UDRE1))); is used to wait until UDRE1 becomes 1 and then cData received by the parameter is written to UDR1 to transmit cData.

```
void USART1_PutString(char* pcStr){
        while(*pcStr)
                USART1_PutChar(*pcStr++);
}
```

USART1_PutString function is used to send character string through USART1. Character string ends with '₩0'(NULL character) and the value is 0. While(*pcStr) statement uses USART1_PutChar to send data pointed by pcStr, increases pcStr value by 1 and continues to repeat until *pcStr becomes NULL character.

```
unsigned char ucLED;

DDRC = 0b01111111;
PORTC = 0b01111111;
```

Declare 8bit variable ucLED where LED output value will be saved, set Port C pins 0~6 for output and output value to 1. All LEDs will go off.

```
cli();
USART1_Init(115200);
sei();
```

Use cli() command to disable all interrupts, interrupts are disabled to prevent error by an interrupt while settings are being changed. Use USART1_Init function to initalize USART1. Use baud rate of 115200 as the parameter value. After USART1 initialization, use sei() command to enable all interrupts.

```
USART1_PutString("Serial Initialized₩r₩n");

ucLED = 0b00000001;
PORTC = (~ucLED) & LED_MASK;
```

Use USART1_PutString to notify intialization of serial communication. ₩r₩n at the end of the character string are used to change the line. ₩r is a Carriage Return that moves the cursor to begining of the line and ₩n is used to change to a new line. Change the ucLED bit 0 to 1 to turn on the Servo LED and then change the PORTC value to actually turn on the LED selected in ucLED. As LED operates in active low mode and turn on when bit is 0, result of &(bitwise AND) operation on LED_MASK and toggled bits from the result of ~(bitwise NOT) opeation perfomed on ucLED are written to PORTC.

```
if(gucRxFlag){
…
        gucRxFlag = 0;
}
```

If statement within the while statement is executed when gucRxFlag has a value other than 0. gucRxFlag is initialized back to 0 when the if statment ends. As long as gucRxFlag value is 0, while statement goes around in loop without taking any action. When data is received by USART1, USART1 receive complete interrupt service routine reads data to gucRxBuffer and sets gucRxFlag to 1. When program encounters if statement after the interrupt, routine within the if statement is executed and guRxFlag is initialized back to 0. If statement is written so that it is executed each time USART1 receive complete interrupt occurs.

```
If(gucRxBuffer == 'z' || gucRxBuffer == 'Z'){
        if(ucLED > 0b00000001){
                ucLED >>= 1;
        }
        else{
                ucLED = 0b01000000;
        }
        PORTC = (~ucLED) & LED_MASK;

        USART1_PutString("LED Moved by ₩'z₩'₩r₩n");
}
```

Codes shown above are encounterd within the if statement. Bits saved in ucLED are shifted one space to the right when the data value saved in gucRxBuffer is either 'z' or 'Z'. If the ucLED value is 0b00000001 without any space to the right, bit value becomes 0b01000000 with the shift made to the further most space to the left . ucLED value is written to PORTC to turn on the LED. After LED has been turned on, USART1_PutString function is used to output character string stating "LED Moved by 'z' key".

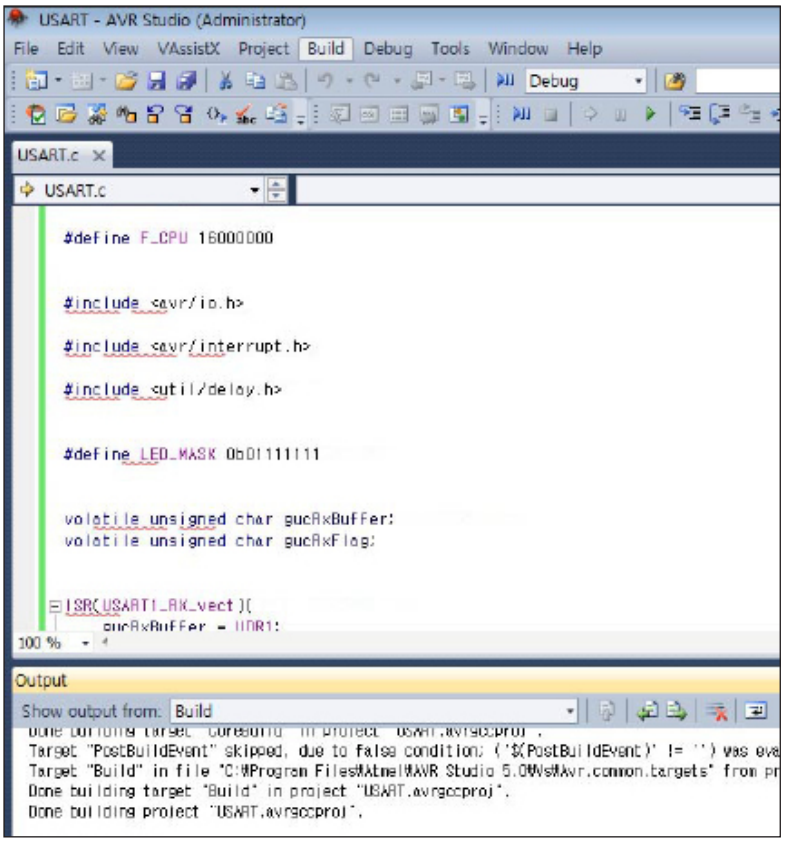
```
else if(gucRxBuffer == 'x' || gucRxBuffer == 'X'){
        if(ucLED < 0b01000000){
                ucLED <<= 1;
        }
        else{
                ucLED = 0b00000001;
        }
        PORTC = (~ucLED) & LED_MASK;

        USART1_PutString("LED Moved by ₩'x₩'₩r₩n");
}
```

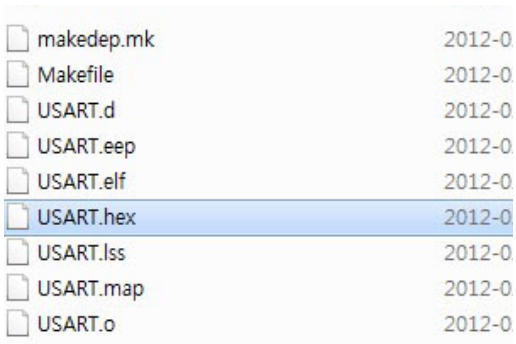
Codes shown above are encountered if the data value saved is not 'z' or 'Z'. ucLED bit is shifted to the left if the data value is 'x' or 'X'. If theucLED value is 0b01000000 without any space to the left, bit value becomes 0b00000001 with the shift made to the right most space. As before, PORTC value is changed to turn on the LED and USART1_PutString function is used to output character string stating "LED Moved by 'x' key". If–else statement ends at this stage, if value other than 'z', 'Z', 'x', 'X' is encountered no action will take place except for gucRxFlag initializtion to 0.

Run

Run the firmware and use Tera Term(Hyper Terminal) to input z, x. Lit LED will move and message will come up in the terminal window.







01 Build

Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.

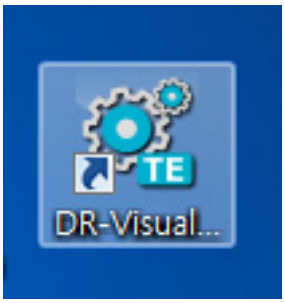
02 Debug Folder

After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.

03 Check hex

Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



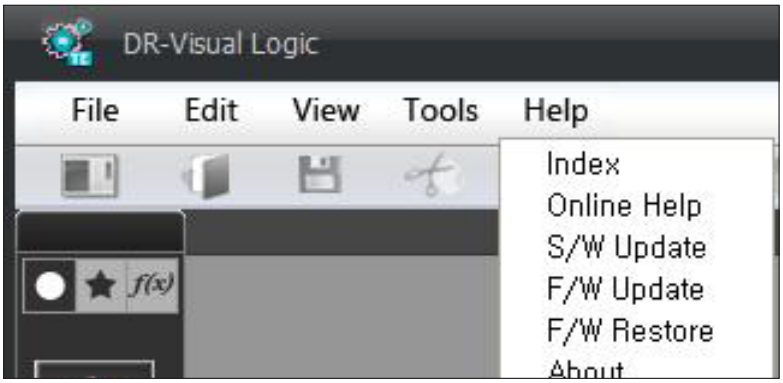
04 Visual Logic

Run DR-Visual Logic.



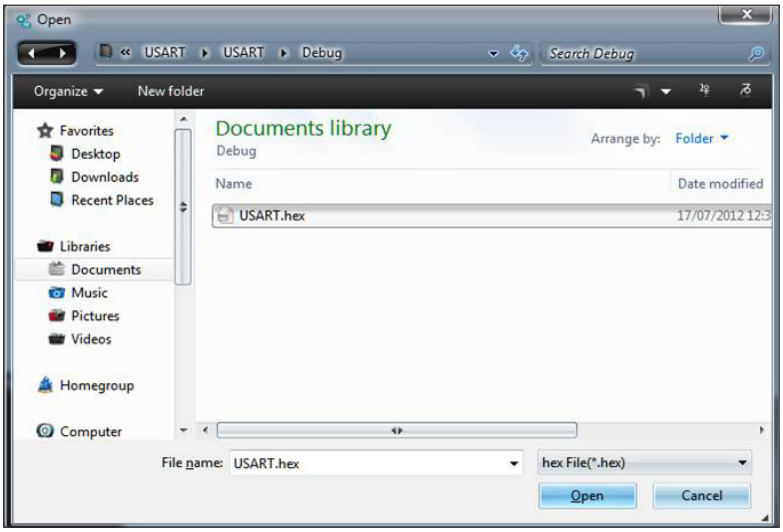
05 Connect

Select COM port and click Connect.



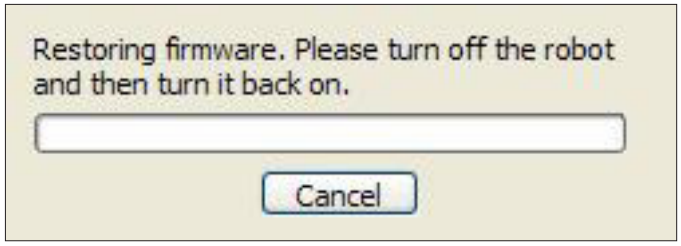
06 Firmware Restoration

Click Restore Firmware in the Help Menu.



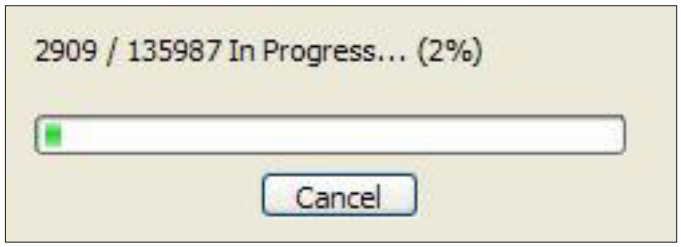
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

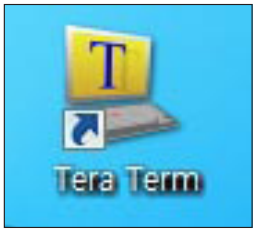
Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

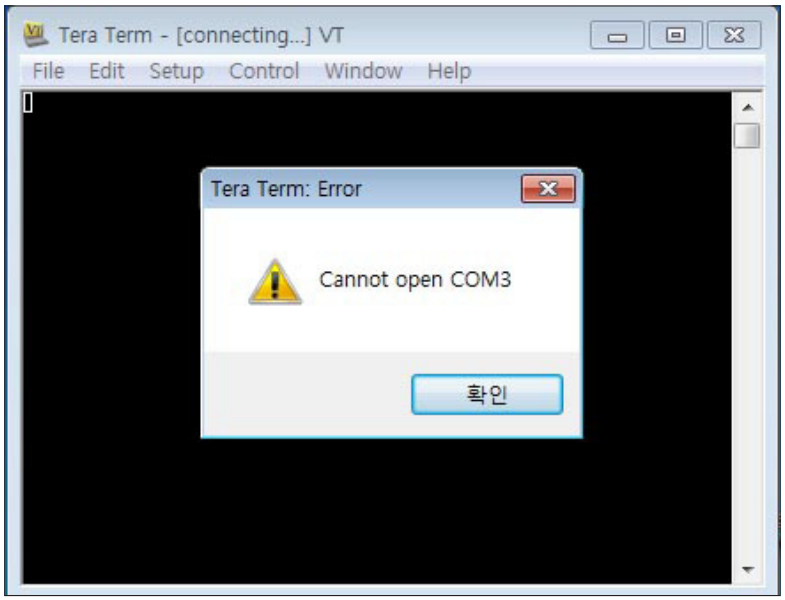
Firmware restoration in progress. Popup window will disappear when restoration is complete.

Use Tera Term to start communication test.



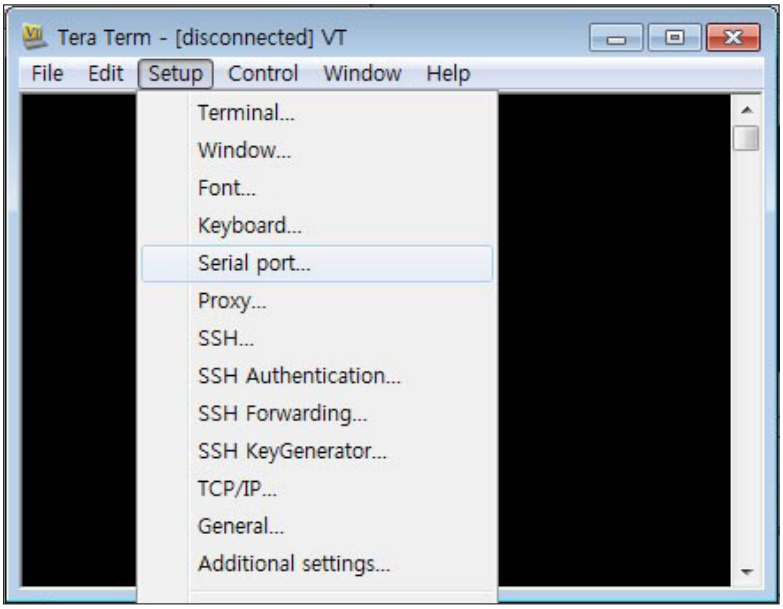
10 Run

Run Tera Term by clicking on the icon.



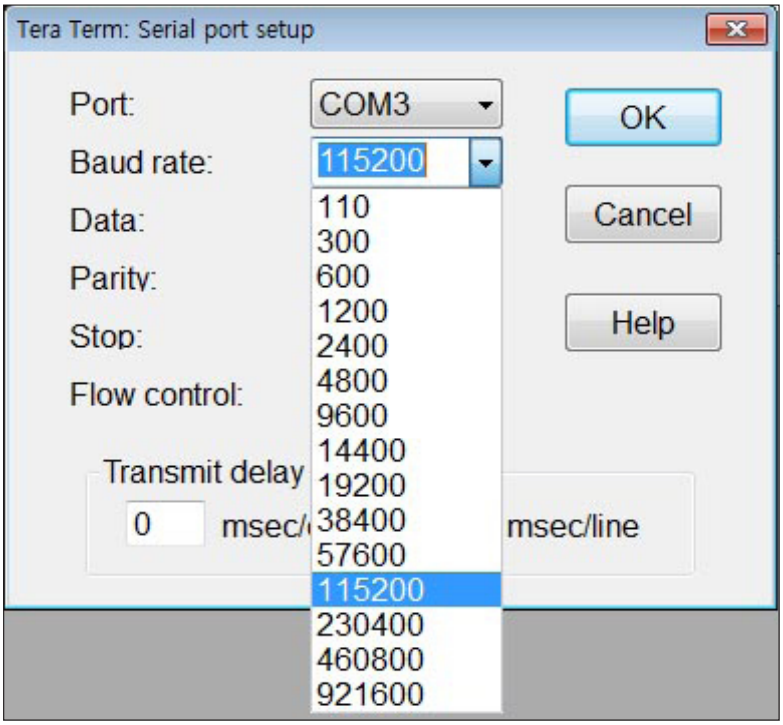
11 Open

Connection does not happen immediately. Click confirm button.



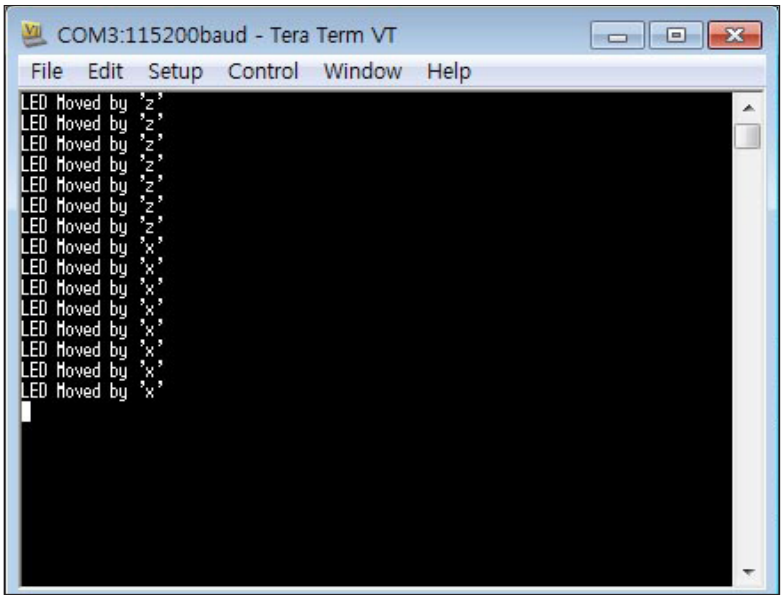
12 Set up

Go to Setup menu and select Serial port.



13 Baud rate

Select Baud rate 115200. This is the maximum baud rate for the controller.



14 Run

Run the firmware and use Tera Term(Hyper Terminal) to input z, x. Lit LED will move and message will come up in the terminal window.

Lit LED will move following the sequence



HerkuleX Control

In this chapter, We will learn to control Herkulex by sending command packets from Herkulex library.

H/W Fundamentals

ATmega128 USART actually consists of USART0 and USART1.USART0 is connected (parallel connection) to 5 4pin ports which are used for communicating with HerkuleX. HerkuleX requiring control can be connected to the DRC by cable and packets sent and received through USART0. As this chapter involves communicating with HerkuleX, some prior knowledge of HekuleX register map and protocol will be required. Refer to the table below for summary of the packet commands that will be used in this example.

Name	CMD	Description
RAM_WRITE	0x03	Write data to HerkuleX volatile register.
RAM_READ	0x04	Read data from HerkuleX volatile register.
I_JOG	0x05	Make HerkuleX move by controlling position/speed.

Summary of items in volatile register range used in this example are as follows.

Name	Address	Size	Description
Torque Control	52	1	Controls torque enable state. 0x00 : Free movement without any resistance 0x40 : Free movement with resistance 0x60 : Torque enabled state
Calibrated Position	58	2	Shows current position and result of GPIO input. Some HerkuleX models use the right side communication port as switch input port. GPIO refers to input value of this switch input port. Bit 15 : Not used Bit 14:13 : Shows GPIO input value Bit 12:0 : Shows current HerkuleX position

Refer to the HerkuleX manaul for more detailed information on protocols and register map.

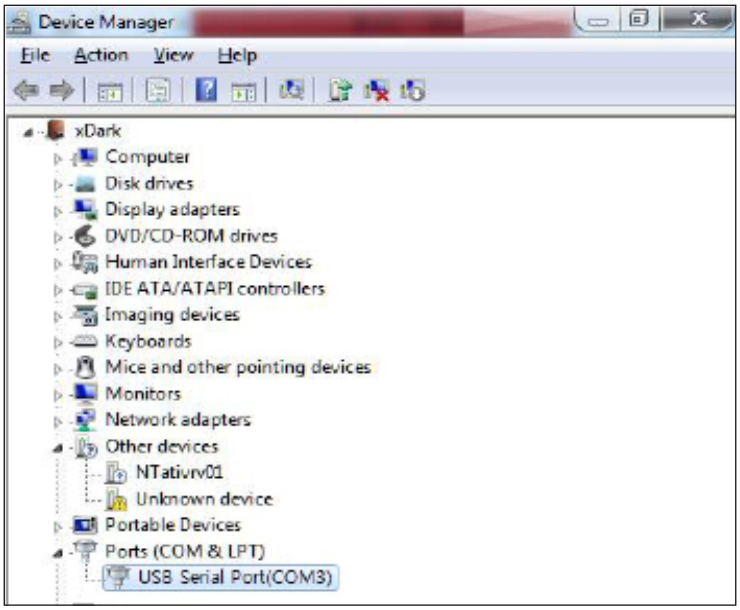


Hardware Preparation

01 Motor Preparation

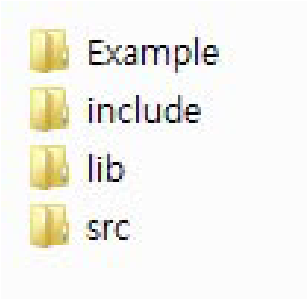
Plug in the DRC power adapter and connect the PC to DRC using the USB to Serial Cable.

Connect the servo motor to the port on the right side of the controller.



02 COM Port Connection

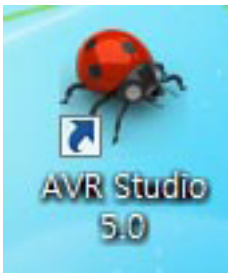
My Computer〉Property〉Device Manager. Check COM connection.



Running AVR Studio & Coding Preparation

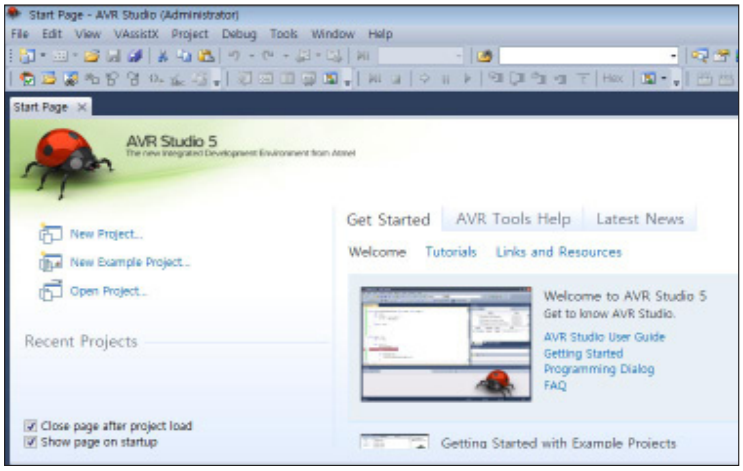
00 Library

Standard library files are required to contorl the servo motor.
Folder structure inclduding the library folder are shown on the left.



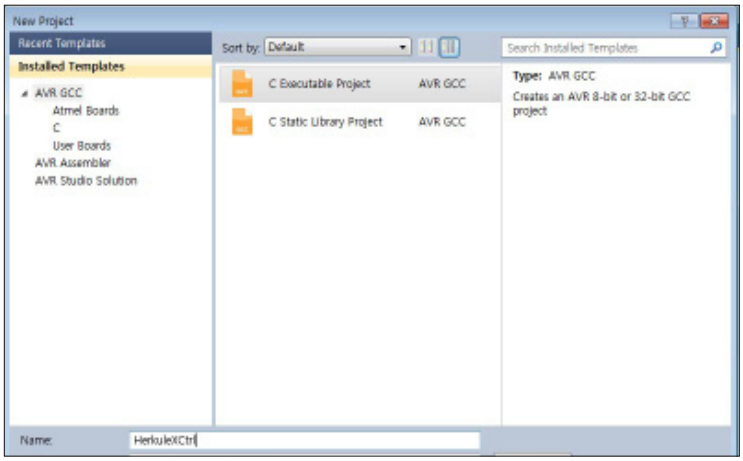
01 Run

Click icon to run AVR Studio.



02 New Project

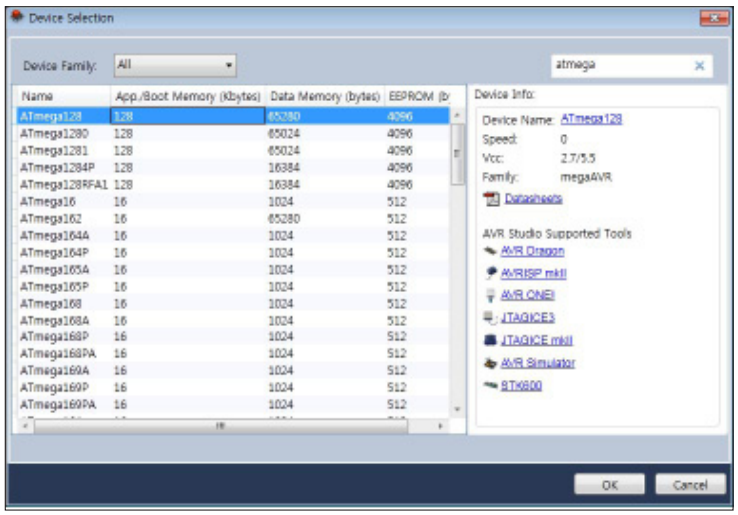
Click New Project.



03 Select GCC

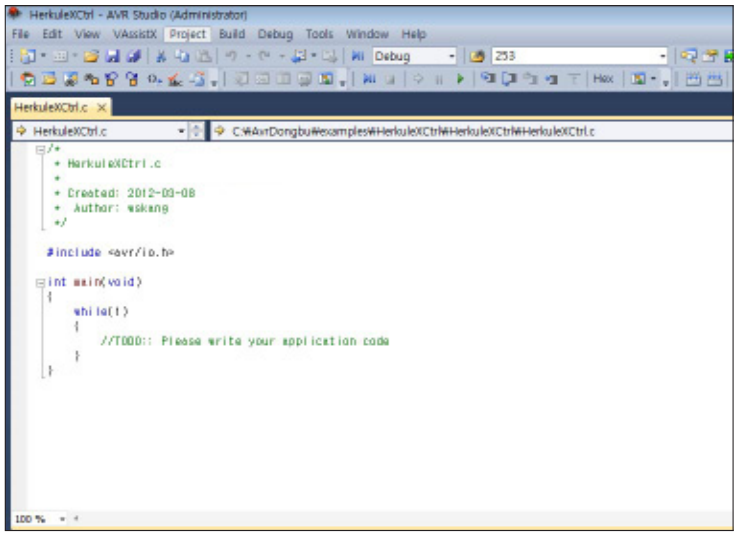
Select C Executable Project

Input 'USART' as Name, Click Create directory for solution, Press OK button.



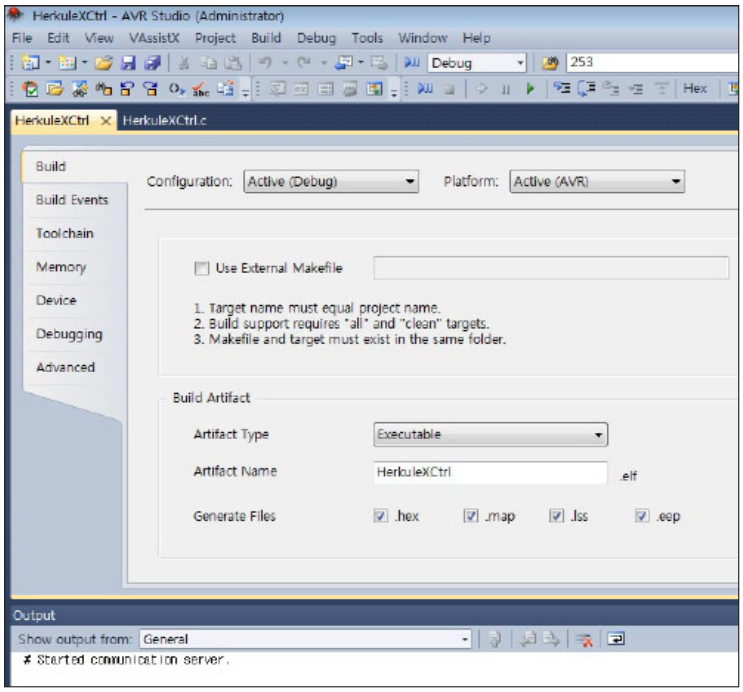
04 Select ATmega

Seach for atmega, Select ATmega128 from the list.



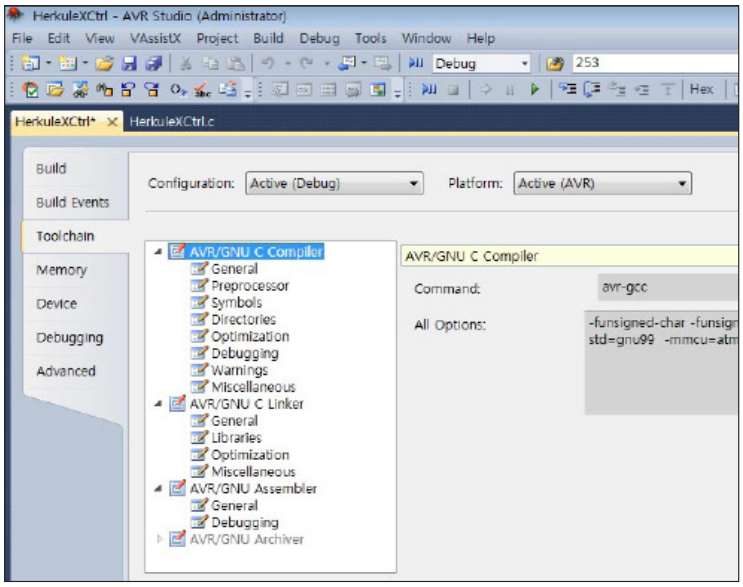
05 Editor

Start coding in the edit window.
Follow the example below.



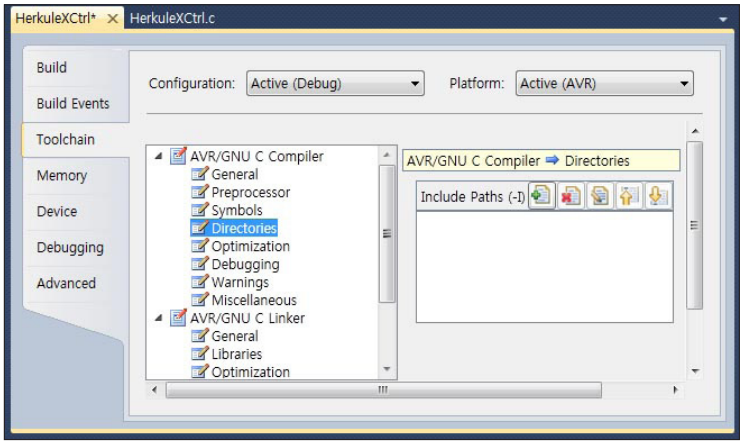
01 Run

From the menu bar found at top of AVR Studio, click on Project to view the HerkuleXCtrl (Name dffers according to project name) property menu. Setup window opens up as shown on the left.



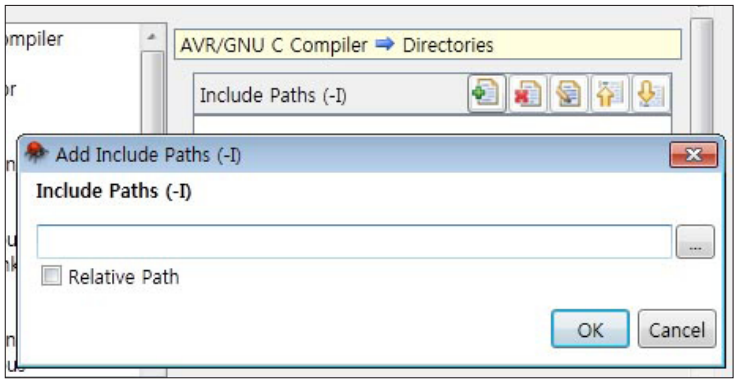
02 Toolchain

Click Toolchain from left tab buttons. Select AVR/GNU C Compiler.



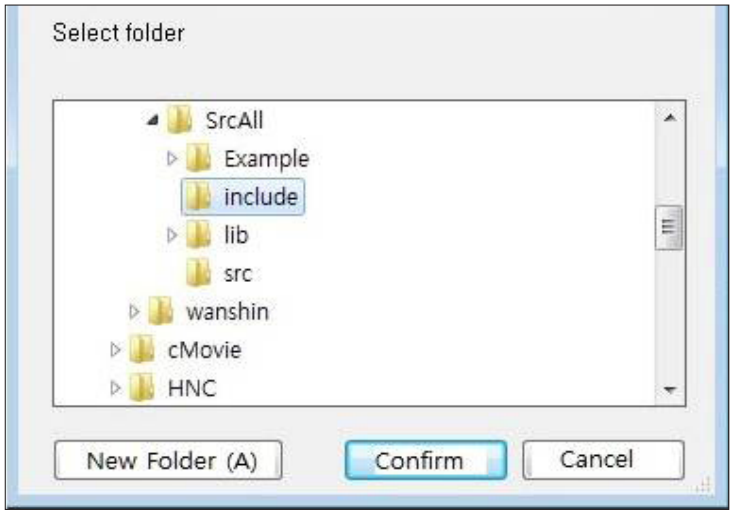
03 C Compiler

From the folders in C Compiler, select Directories.



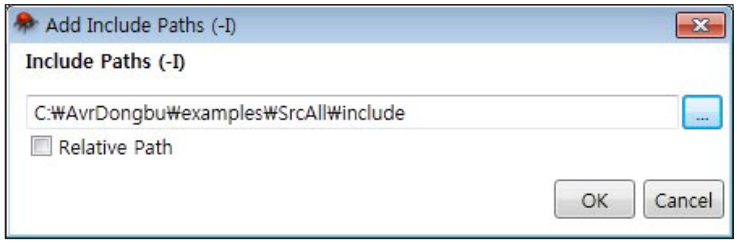
04 Path

Click '+' button on the right side of 'Include Paths' to open 'Include Paths' popup window. Uncheck 'Relative Path' check box and click on the folder search button ',,,' on the right.



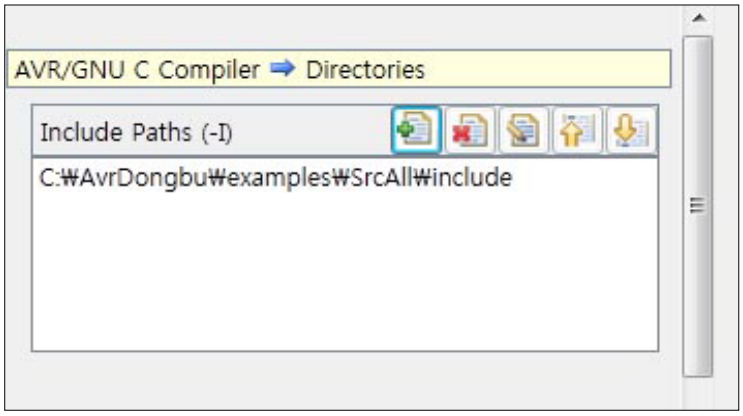
05 Include

select 'include' folder from the provided library folders.



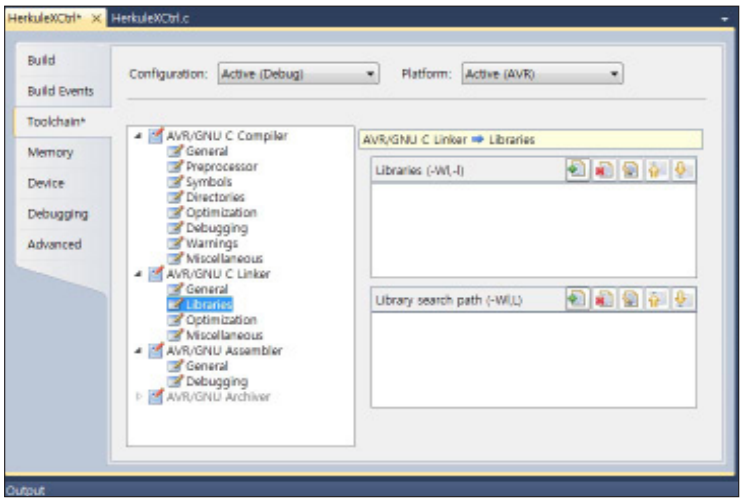
06 Include Complete

'Include' folder path selected. Click OK button.



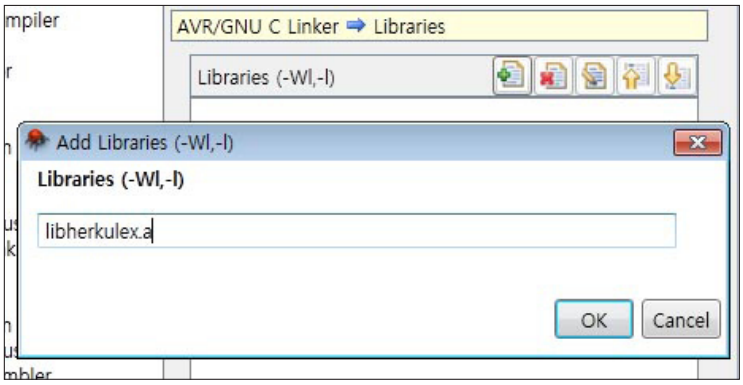
07 Path Complete

Include path selection complete.



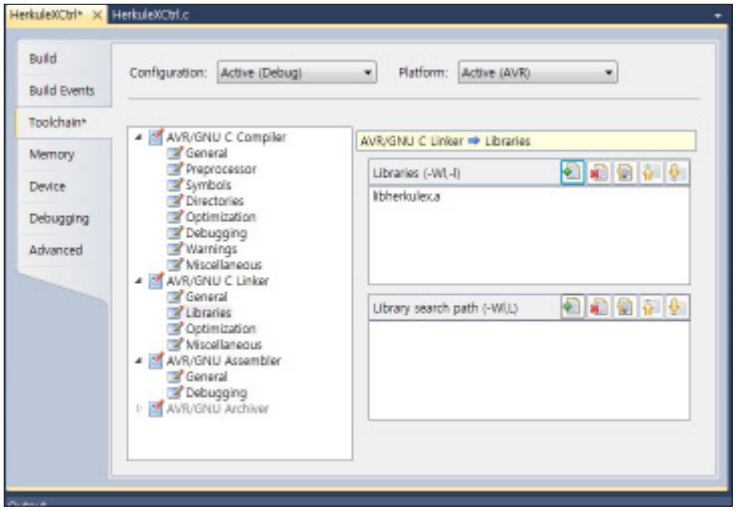
08 C Linker

Select 'Libraries' from C Linker folders.



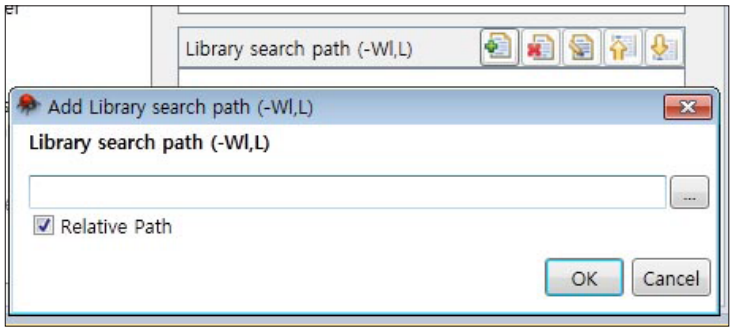
09 Libraries

Click '+' button on the right side of 'Libraries' to open 'Add Libraries' popup window. Type 'libherkulex.a' and click OK button.



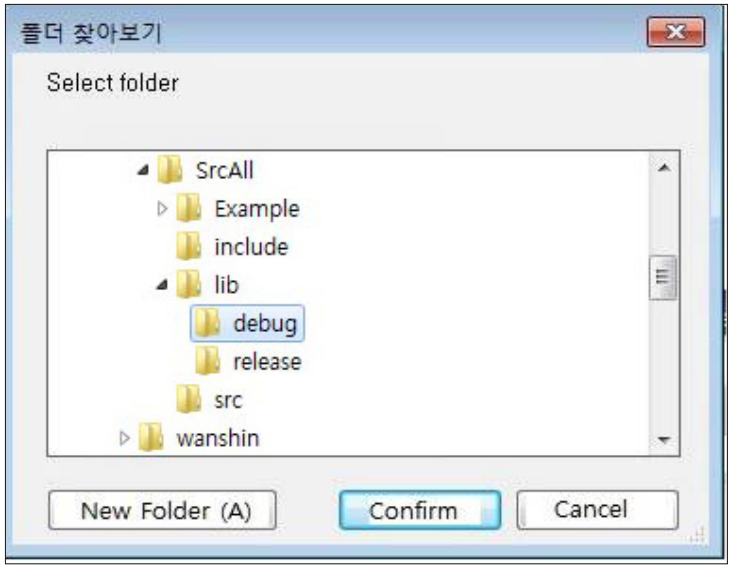
10 Search

Link to servo motor library file 'libherkulx.a' completed.
Next Setup Library search path
Click '+' button on the right.



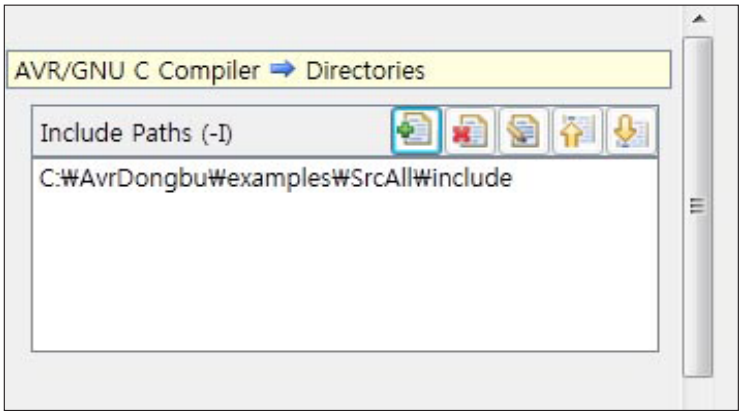
11 Path

Click on the '...' folder search button.



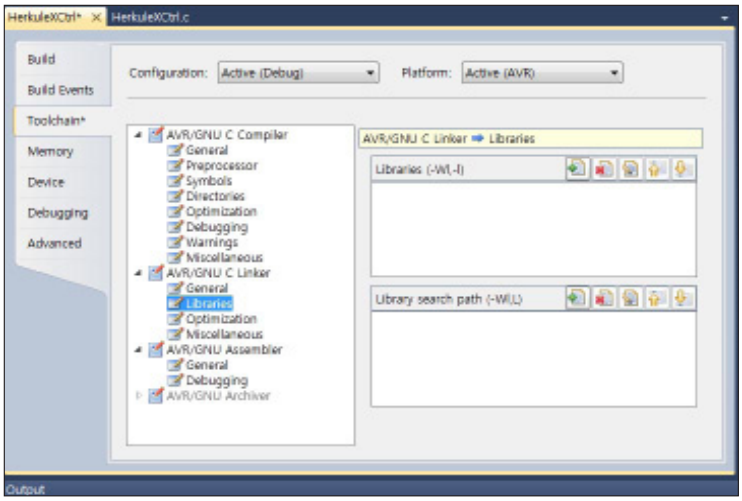
12 Debug

Select 'debug' folder from the provided libary folders. 'debug' folder is found within the 'lib' folder.



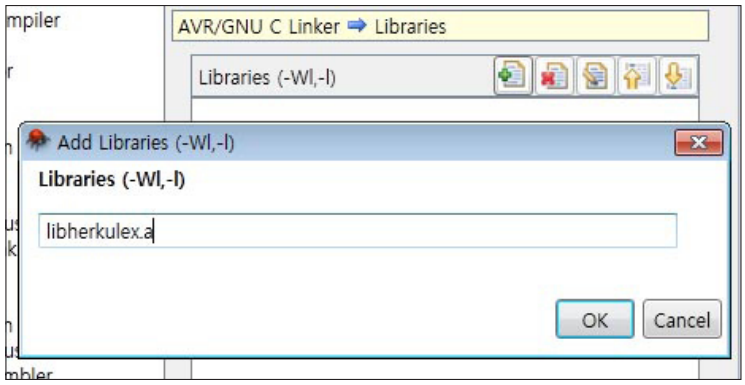
13 Path Complete

Uncheck 'Relative Path' check box.
Library search path completed.
Click OK button.



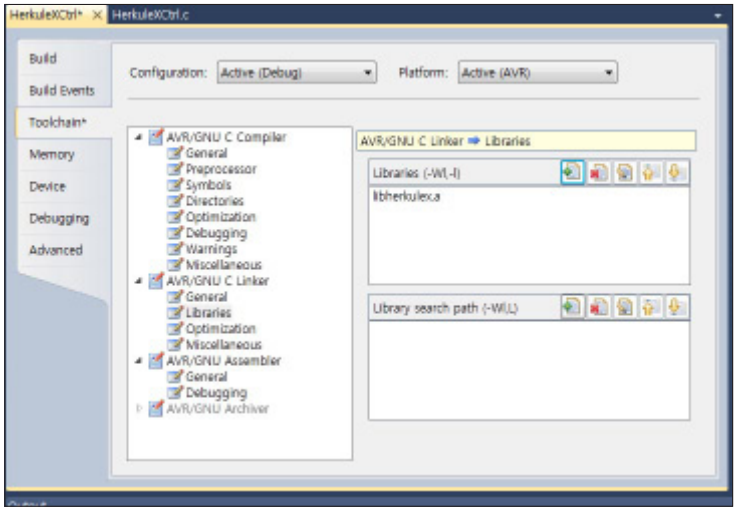
14 Link Complete

C Linker liberary file and path setup completed.



15 Save

Select save button to save setup. Tab bar name changes from HerkuleXCtrl *〉 HerkuleXCtrl.



16 Coding

Select HerkuleX.c tab bar and write the example provided below.

When servo is purchased indivdually, it has default ID of 253. However, servos that are included in the humanoid set has individual ID numbers marked on the case. Marked ID number should be used when operating the servo.

Example

This example will repeatdly move the servo motor DRS-1010 with ID 253 and read the position value to corfirm that it has arrived at the goal position.

TIMER0.h

```
#ifndef TIMER0_H_
#define TIMER0_H_

#ifdef __TIMER0_C
        #define TIMER0_EXT
#else
        #define TIMER0_EXT extern
#endif

//Global variable for using the timer to estimate the time.
TIMER0_EXT volatile unsigned char gucTimerTick;
//Timer/Counter0 initialization function
TIMER0_EXT void TIMER0_Init(void);

#endif /* TIMER0_H_ */
```

herkulex.h

```
#ifndef HERKULEX_H_
#define HERKULEX_H_

////////////////////////////////Defining Protocol////////////////////////////////
//Index of individual items
#define PROTOCOL_SIZE_IDX               2
#define PROTOCOL_ID_IDX                 3
#define PROTOCOL_CMD_IDX                4
#define PROTOCOL_CS1_IDX                5
#define PROTOCOL_CS2_IDX                6
#define PROTOCOL_DATA_IDX               7

//Header related
#define HEADER                          0xFF

//SIZE related
#define MIN_PACKET_SIZE                 7
#define MIN_ACK_PACKET_SIZE             9
#define MAX_PACKET_SIZE                 223
#define MAX_DATA_SIZE                           (MAX_PACKET_SIZE-MIN_PACKET_
SIZE)
```

```
//ID related
#define MAX_ID                          0xFD
#define BROADCAST_ID                    0xFE

//CMD related - Request Packet
#define CMD_EEP_WRITE                   0x01
#define CMD_EEP_READ                    0x02
#define CMD_RAM_WRITE                   0x03
#define CMD_RAM_READ                    0x04
    #define CMD_RW_DATA_ADDR_IDX        7
    #define CMD_RW_DATA_LEN_IDX         8
#define CMD_I_JOG                       0x05
    #define CMD_I_JOG_STRUCT_SIZE       5
    #define CMD_I_JOG_MAX_DRS           (MAX_DATA_SIZE/CMD_I_JOG_STRUCT_
SIZE)
#define CMD_S_JOG                       0x06
    #define CMD_S_JOG_STRUCT_SIZE       4
    #define CMD_S_JOG_MAX_DRS           (MAX_DATA_SIZE/CMD_S_JOG_
STRUCT_SIZE)
#define CMD_STAT                        0x07
#define CMD_ROLLBACK                    0x08
#define CMD_REBOOT                      0x09

#define CMD_MIN                         (CMD_EEP_WRITE)
#define CMD_MAX                         (CMD_REBOOT)

//CMD related - ACK Packet
#define CMD_ACK_MASK                    0x40

#define CMD_EEP_WRITE_ACK               (CMD_EEP_WRITE|CMD_ACK_MASK)
#define CMD_EEP_READ_ACK                (CMD_EEP_READ|CMD_ACK_MASK)
#define CMD_RAM_WRITE_ACK               (CMD_RAM_WRITE|CMD_ACK_
MASK)
#define CMD_RAM_READ_ACK                (CMD_RAM_READ|CMD_ACK_MASK)
#define CMD_I_JOG_ACK                   (CMD_I_JOG|CMD_ACK_MASK)
#define CMD_S_JOG_ACK                   (CMD_S_JOG|CMD_ACK_MASK)
#define CMD_STAT_ACK                    (CMD_STAT|CMD_ACK_MASK)
#define CMD_ROLLBACK_ACK                (CMD_ROLLBACK|CMD_ACK_MASK)
#define CMD_REBOOT_ACK                  (CMD_REBOOT|CMD_ACK_MASK)

#define CMD_ACK_MIN                     (CMD_EEP_WRITE_ACK)
#define CMD_ACK_MAX                     (CMD_REBOOT_ACK)

//CheckSum related
#define CHKSUM_MASK                     0xFE
```

```
//////////////////////////////Protocol Structure//////////////////////////////
typedef struct DrsJog
{
        unsigned int    uiValue : 15;
        unsigned int    reserved : 1;
}DrsJog;

typedef struct DrsSet
{
        unsigned char   ucStopFlag : 1;
        unsigned char   ucMode : 1;
        unsigned char   ucLedGreen : 1;
        unsigned char   ucLedBlue : 1;
        unsigned char   ucLedRed : 1;

unsigned char   ucJogInvalid : 1;
        unsigned char   reserved : 2;
}DrsSet;

typedef struct DrsIJog
{
        DrsJog          stJog;
        DrsSet          stSet;
        unsigned char   ucId;
        unsigned char   ucPlayTime;
}DrsIJog;

typedef struct DrsSJog
{
        DrsJog          stJog;
        DrsSet          stSet;
        unsigned char   ucId;
}DrsSJog;

typedef struct DrsIJogData
{
        DrsIJog         stIJog[CMD_I_JOG_MAX_DRS];
}DrsIJogData;

typedef struct DrsSJogData
{
        unsigned char   ucPlayTime;
        DrsSJog         stSJog[CMD_S_JOG_MAX_DRS];
}DrsSJogData;

typedef struct DrsRWData
{
        unsigned char   ucAddress;
```

```
unsigned char    ucLen;
        unsigned char    ucData[MAX_DATA_SIZE-2];
}DrsRWData;

typedef union DrsData
{
        unsigned char    ucData[MAX_PACKET_SIZE-MIN_PACKET_SIZE];

        DrsRWData        stRWData;
        DrsIJogData      stIJogData;
        DrsSJogData      stSJogData;
}DrsData;

typedef struct
{
        unsigned char    ucHeader[2];
        unsigned char    ucPacketSize;
        unsigned char    ucChipID;
        unsigned char    ucCmd;
        unsigned char    ucCheckSum1;
        unsigned char    ucCheckSum2;
        DrsData          unData;
}DrsPacket;

//////////////////////////////Reception status result values//////////////////////////////
enum{
        DRS_RXWAITING,
        DRS_RXCOMPLETE,
        DRS_HEADERNOTFOUND,
        DRS_INVALIDSIZE,
        DRS_UNKNOWNCMD,
        DRS_INVALIDID,
        DRS_CHKSUMERROR,
        DRS_RXTIMEOUT
}DrsRxStatus;

#ifdef __HERKULEX_C
        #define HERKULEX_EXT
#else
        #define HERKULEX_EXT extern
#endif

//Intialization function to control HerkuleX
HERKULEX_EXT void hklx_Init(unsigned long ulBaudRate);
//Function used for sending packet to HerkuleX
HERKULEX_EXT void hklx_SendPacket(DrsPacket stPacket);
//Function used for receiving packt from HerkuleX
HERKULEX_EXT unsigned char hklx_ucReceivePacket(DrsPacket *pstPacket);

#endif /* HERKULEX_H_ */
```

HerkuleXCtrl.c

```
//CPU clock setup
#define F_CPU 16000000

//ATmega128 registers defined
#include <avr/io.h>
//information required for using interrupt defined
#include <avr/interrupt.h>
//_delay_ms function defined
#include <util/delay.h>

//herkulex library header file included
#include <herkulex.h>
#include <TIMER0.h>

//DRS-0101의 Structure for using Calibrated Position information
typedef struct DrsCaliPos
{
        int             iPosition : 13;
        unsigned int    uiGPIO1 : 1;
        unsigned int    uiGPIO2 : 1;
        unsigned int    reserved : 1;
}DrsCaliPos;

//Union for using Calibrated Position information
typedef union DrsUnionCaliPos
{
        DrsCaliPos      stCaliPos;
        unsigned int    uiCaliPos;
}DrsUnionCaliPos;

int main(void)
{
        //Define packet structure for saving request/ack packets
        DrsPacket stSendPacket, stRcvPacket;
        //Declare union variable for saving Calibrated Position value
        DrsUnionCaliPos unCaliPos;
        //Declare variables to be used
        unsigned char ucResult;
        int iPos = 400;

        //Set PORTC I/O direction
        DDRC = 0b01111111;
        //Set PORTC output value
        PORTC = 0b01111111;
```

```
//Disable all interrupts
	cli();

	//Initialize HerkuleX prior to use
	hklx_Init(115200);

	//Enable all interrupts
	sei();

	//Write 0x60 to Torque Control to form packet for setting torque
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
	stSendPacket.ucChipID = 253;
	stSendPacket.ucCmd = CMD_RAM_WRITE;
	stSendPacket.unData.stRWData.ucAddress = 52;
	stSendPacket.unData.stRWData.ucLen = 1;
	stSendPacket.unData.stRWData.ucData[0] = 0x60;

	//Send packet
	hklx_SendPacket(stSendPacket);

	while(1)
	{
		//Change goal position variable value
		if(iPos == 400){
			iPos = 624;
		}
		else{
			iPos = 400;
		}

		//Form I_JOG packet to opeate DRS-0101
		stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_
SIZE;
		stSendPacket.ucChipID = BROADCAST_ID;
		stSendPacket.ucCmd = CMD_I_JOG;
		stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = iPos;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
		stSendPacket.unData.stIJogData.stIJog[0].ucId = 253;
		stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

		 //Send packet
		hklx_SendPacket(stSendPacket);

		//delay 1s until movement is complete
		_delay_ms(1000);
```

```
        //Form RAM_REad packet to read Calibrated Position value
                stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2;
                stSendPacket.ucChipID = 253;
                stSendPacket.ucCmd = CMD_RAM_READ;
                stSendPacket.unData.stRWData.ucAddress = 58;
                stSendPacket.unData.stRWData.ucLen = 2;

                //Send packet
                hklx_SendPacket(stSendPacket);

                //Set TIMEOUT wait time. 2*15 = 30(ms)
                gucTimerTick=15;
                while(1){
                        //Call function for receiving packets and save the result in ucResult
                        ucResult = hklx_ucReceivePacket(&stRcvPacket);
                        //Exit if result is not  DRS_RXWAITING이
                        if(ucResult != DRS_RXWAITING){
                                break;
                        }
                        //Wait 30ms and exit when  gucTimerTick is 0
                        if(gucTimerTick==0){
                                ucResult = DRS_RXTIMEOUT;
                                break;
                        }
                }

                //When packet reception is complete without error
                if(ucResult == DRS_RXCOMPLETE){
                        //Save 2byte of received data in union variable
                        unCaliPos.uiCaliPos = stRcvPacket.unData.stRWData.ucData[0] |
                                        (stRcvPacket.unData.stRWData.ucData[1]<<8);

                        //Turn off all LED when within 5 from goal position
                        if((iPos - unCaliPos.stCaliPos.iPosition) > -5 &&
                          (iPos - unCaliPos.stCaliPos.iPosition) < 5){
                                PORTC = 0b01111111;
                        }
                        //Turn on all LED when beyond 5 from goal position
                        else{
                                PORTC = 0b00000000;
                        }
                }
                //Turn on all LED when packet not received properly
                else{
                        PORTC = 0b00000000;
                }
        }
                //Return integer value to fit format of the function
        return 1;
}
```

Example Description

Header file using #include statement to include in HerkuleXCtrl.c will be explained first.

```
#ifndef TIMER0_H_
#define TIMER0_H_

…

#endif /* TIMER0_H_ */
```

```
#define __TIMER0_C
        #include “TIMER0.h”
#undef __TIMER0_C
```

(Part of TIMER0.c)

Top most code is for defining all global variables and original fuction forms once. As seen in the TIMER0.c part, _TIMERT0_C is first defined by #define before including TIMER0.h by #include statement in TIMERT0.c. Then, when TIMER0.h is compiled in TIMER0.c, the header file is compiled as below:

```
volatile unsigned char gucTimerTick;
void TIMER0_Init(void);
```

extrern in front of variables and functions are used to call or access variables and functions defined in other source files from the current source file. To use the TIMER0.c variables and functions from a file other than TIMER0.c, extern is added and then compiled. Compiler directives will not be mentioned again in other header files since compiler directives explained so far are applied to other header files iln the same manner as in TIMER0.h, As same as the variable in Chapter 5, global variable gucTimerTick in TIMER0.h decreases by 1 every 2ms if the variable value is not 0. TIMER0_Init() fuction is a TIMER0 initialization fuction that sets TCNT0 value to initialization value, enables Timer/Counter0 overflow interrupt, and starts Timer/Counter 0 using specified clock divider.

herkulex.h

herkulex.h includes protocol declarations and structure definitions necessary for controlling HerkuleX. First part of protocol declartions include declarations on various information related to Header, Size, ID, CMD, and Check Sum making up the packet.

Following the declarations, packet structures are defined for each CMD to facilitate packet creation by the users. DRSJog is a tructure carrying information on HerkuleX goal position/speed in I_JOG and S_JOG packet. DrsSet is also a structure carrying various setup information in I_JOG and S_JOG packet. DrsIJog and DrsSJog carries information related to single servo motor in I_JOG and S_JOG packet. DrsIJogData and DrsSJogData carries information relavant to all servos. DrsRWData is used in following four commands; EEP_WRITE, EEP_READ, RAM_WRITE, and RAM_READ. Packet format with register address in fisrt data byte, length of the data to be written or read in the second byte, and data to be written (only applies to WRITE command) from the third byte is shown as a structure. DrsIJogData, DrsSJogData, and DrsRWData are all structures showing Optional Data part in a packet which can be tied together by a union DrsData so that each CMD type can access the same memory range in different way. DrsPacket is a structure for all packets and include HEADER, SIZE, ID, CMD, Check Sum and Optional Data. Declare DrsPacket variable in HerkuleXCtrl.c to compose, send, receive, and analyze received packet.

```
enum{
        DRS_RXWAITING,
        DRS_RXCOMPLETE,
        DRS_HEADERNOTFOUND,
        DRS_INVALIDSIZE,
        DRS_UNKNOWNCMD,
        DRS_INVALIDID,
        DRS_CHKSUMERROR,
        DRS_RXTIMEOUT
}DrsRxStatus;
```

After the packet structure, receive stataus value returned from hklx_ucReceivePacket is declared as variable (enum) DrsRxStatus.

```
HERKULEX_EXT void hklx_Init(unsigned long ulBaudRate);
HERKULEX_EXT void hklx_SendPacket(DrsPacket stPacket);
HERKULEX_EXT unsigned char hklx_ucReceivePacket(DrsPacket *pstPacket);
```

Above codes are functions declared in herkulex.h. hklx_Init() is a function used for initializing ATmega128 to control HerkuleX. Timer/Counter0 and USART0 are also initialized. hklx_SendPacket() is a function used for sending packets to HerkuleX. When hklx_SendPacket() is called after entering SIZE, ID, CMD, and Optional Data in variable DrsPacket, function will send the packet to USART0 after entering header and Checksum into the stPAcket. Hklx_ucReceivePacket() is a function used for receiving packet from the HerkuleX. Function will check the USART0 receive buffer to verify the received data and then copy the content to pstPacket before returning DRS_RXCOMPLETE. If the received data is corrupt or incomplete, DrsRxStatus with value fitting the circumstances will be sent back.

HerkuleXCtrl.c

```
#define F_CPU 16000000
```

#define statement is one of the preprocessor commands, #define command will replace any F_CPU found later with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on the clock speed, this command sets the clock speed so that delay function and other fuctions execute properly.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. avr/interrupt.h header file has defined functions required for using the interrupt. util/delay.h header file has _delay_ms() and other commonly used delay functions defined. Use #include statement to include the functions.

```
#include <herkulex.h>
#include <TIMER0.h>
```

herkulex.h header file must be always inlcuded. Header includes declarations and structures for creating and sending packets, and defined functions. TIMER0.h is also included to make packet Timeout decision using gucTimerTick.

```
typedef struct DrsCaliPos
{
        int             iPosition : 13;
        unsigned int    uiGPIO1 : 1;
        unsigned int    uiGPIO2 : 1;
        unsigned int    reserved : 1;
}DrsCaliPos;

typedef union DrsUnionCaliPos
{
        DrsCaliPos      stCaliPos;
        unsigned int    uiCaliPos;
}DrsUnionCaliPos;
```

As mentioned previously, Calibrated Position in HerkuleX register map contains position value and GPIO input value. Everytime Calibrated position value is read, GPIO value is read together with the position value but since only the position value is required, bit operation has to be performed to isolate the position value only. In order to avoid this type of inconvenience, structure and union are declared as shown above. DrsCaliPos is a structure that shows how each bit in Calibrated Position is being used, DrsUnionCaliPos ties DrsCaliPos togeter with unsigned int so that 2 bytes of data in Calibrated Position can be accessed by DrsCaliPos and unsigned in two differnt ways.

```
DrsPacket stSendPacket, stRcvPacket;
DrsUnionCaliPos unCaliPos;
unsigned char ucResult;
int iPos = 400;
```

Declare variables for use in main function. Declare variables for sending and receiving DrsPacket as structure type and declare unCaliPos as DrsUnionCaliPos union type to process received data. Declare ucResult to receive hklx_ucReceivePacket() result values. Declare goal position iPOS and initialize to 400.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Set PORT C pin numbers 0~6 for output and output value as 1 to turn off all LED

```
cli();
hklx_Init(115200);
sei();
```

Use cli() command to disable all interrupts, interrupts are disabled to prevent error by an interrupt while settings are being changed. Use hklx_Init function to initalize Timer/Counter0 and USART0. Use baud rate of 115200 as the parameter value. Finally, use sei() command to enable all interrupts.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
stSendPacket.ucChipID = 253;
stSendPacket.ucCmd = CMD_RAM_WRITE;
stSendPacket.unData.stRWData.ucAddress = 52;
stSendPacket.unData.stRWData.ucLen = 1;
stSendPacket.unData.stRWData.ucData[0] = 0x60;

hklx_SendPacket(stSendPacket);
```

Torque must be enabled so that motor can be controlled to move on its own. Codes shown above are for writing and sending RAM_WRITE packet to enable the motor torque. The size of the packet MIN_PACKET_SIZE(7byte), address + length + data(1byte) equals MIN_PACKET_SIZE+3(10bytes), motor ID is 253. CMD is CMD_RAM_WRITE(0x03). Set ucAddress, ucLen, and ucData[0] values to 52, 1, 0x60, these three belong to structure stRWData which in turn is a member of the union unData. Finally, call function hklx_SendPacket() to send the packet.

```
if(iPos == 400){
        iPos = 624;
}
else{
        iPos = 400;
}
```

Codes starting from this point are content of the while statement. iPos value showing the goal postion of the servo motor is changed as shown above so that value can shift back and forth from 624 to 400 and vice versa everytime while statement is repeated.while

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = iPos;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
stSendPacket.unData.stIJogData.stIJog[0].ucId = 253;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

hklx_SendPacket(stSendPacket);

_delay_ms(1000);
```

This section deals with creating and sending I_JOG packet to move the motor to iPos position. Packet size is composed of MIN_PACKET_SIZE(7bytes) and data per motor CMD_I_JOG_STRUCT_SIZE(5bytes) for total of 12bytes. I_JOG will use broadcast ID BROADCAST_ID(254) since this command is used for sending command to several motors all at once. CMD is CMD_I_JOG(0x05), Appropriate data is written to stIJog[0] which belongs to structure atIJogData which in turn is a member of union unData.

Write goal position value iPos to uiValue of stJog. Since this is a position control mode, set ucMode of stSet to 0, set ucLedGreen, ucLedBlue, and ucLedRed of stSet to 1 to turn on all 3 LEDs, set ucID to 256, and set ucPlayTime to 50(560ms). Next, call function hklx_SendPacket() to send the packet and wait 1s until the movement ends.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2;
stSendPacket.ucChipID = 253;
stSendPacket.ucCmd = CMD_RAM_READ;
stSendPacket.unData.stRWData.ucAddress = 58;
stSendPacket.unData.stRWData.ucLen = 2;

hklx_SendPacket(stSendPacket);
```

After the motor movement ends, current position of the motor is read to confirm whether the motor has reached its' goal position. Codes above create and send RAM_READ packet to read the Calibrated Position value. The size of the packet MIN_PACKET_SIZE(7bytes), address + length equals MIN_PACKET_SIZE+2(9bytes). Motor IID is 253. CMD is CMD_RAM_READ(0x04). Set ucAddress value to 58 and cuLen value to 2, these belong to structure stRWData which in turn is a member of union unData. Finally, function hklx_SendPacket() is called to send the packet.

```
gucTimerTick=15;
while(1){
        ucResult = hklx_ucReceivePacket(&stRcvPacket);
        if(ucResult != DRS_RXWAITING){
                break;
        }

        if(gucTimerTick==0){
                ucResult = DRS_RXTIMEOUT;
                break;
        }
}
```

This function sets the Timeout and waits for ACK Packet to arrive. libherkulex.a file contains Timer/Counter0 overflow interrupt service routine. When gucTimerTic value is set to 15 in HerkuleXCtrl, this variable will decrease by 1 every 2ms or each time Timer/Counter0 overflow interrupt occurs until it become 0 in about 30ms. During the 30s time frame, hklx_ucReceivePacket() will be called continuously in the while statement and the result saved in ucResult. Resulting value is DRS_RXWAITING during the waiting period for ACK packet to arrive. When complete packet is received or when packet error occurs, value appropriate to the condition is selected from DrsRxStatus and sent back. When ucResult vaule is not DRS_RXWAITING, break statement is used to exit the while statement.

Also, if ucResult value is still DRS_RXWAITING evern after 30ms has passed and gucTimerTick value has tuned to 0, ucResult value will be set to DRS_RXTIMEOUT and program will exit the while statement.

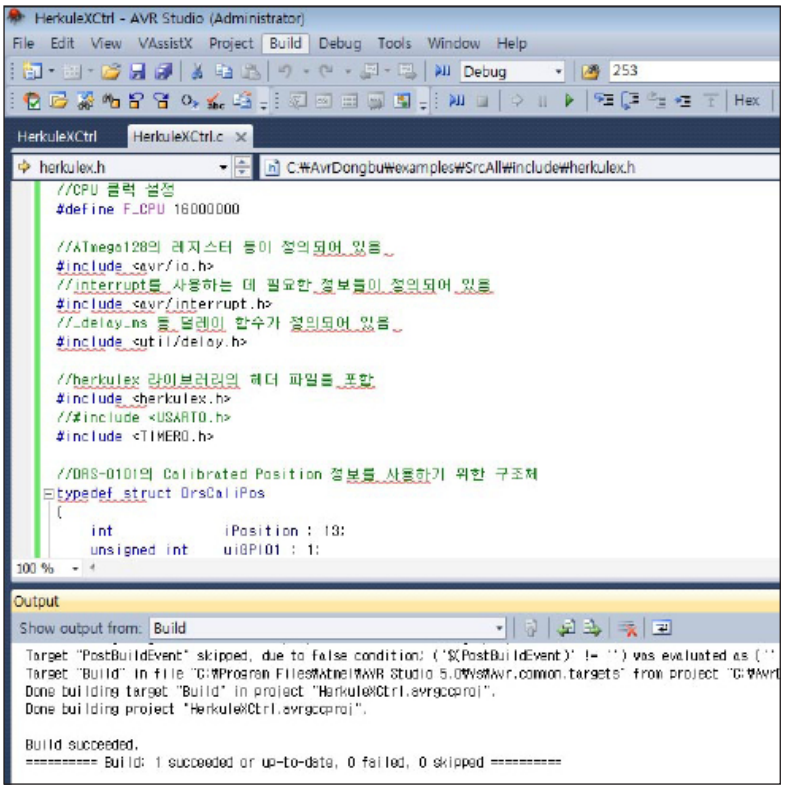
```
if(ucResult == DRS_RXCOMPLETE){
        unCaliPos.uiCaliPos = stRcvPacket.unData.stRWData.ucData[0] |
                              (stRcvPacket.unData.stRWData.ucData[1]<<8);

        if((iPos - unCaliPos.stCaliPos.iPosition) > -5 &&
          (iPos - unCaliPos.stCaliPos.iPosition) < 5){
                PORTC = 0b01111111;
        }
        else{
                PORTC = 0b00000000;
        }
}
else{
        PORTC = 0b00000000;
}
```

When complete ACK packet is received, ucResult value becomes DRS_RXCOMPLETE and data portion with Calibrated Position located in RAM_READ ACK packet is saved to uiCaliPos which is a member of union variable uiCaliPos. To read and use the lower 13 bytes containing the locataion data, access the iPosition in stCaliPos which belong to union unCaliPos, Use the if statement to check the difference between goal position iPos and current position unCaliPos. stCaliPos. Turn off all DRC LEDs If the difference is less than 5 and turn on all DRC LEDs if the difference is greater than 5. All LEDs will also be turned on if incomplete packet is received due to a packet error.

Run

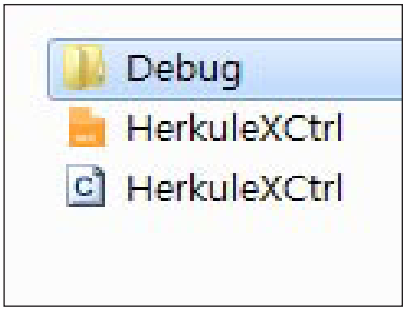
Motor will move to the left and right. When program is run after downloading the firmware, HerkuleX with ID 253 will start to move back and forth from position 400 to 624 every 1s. All DRC LED will be off if the servo motor is operating properly.



01 Build

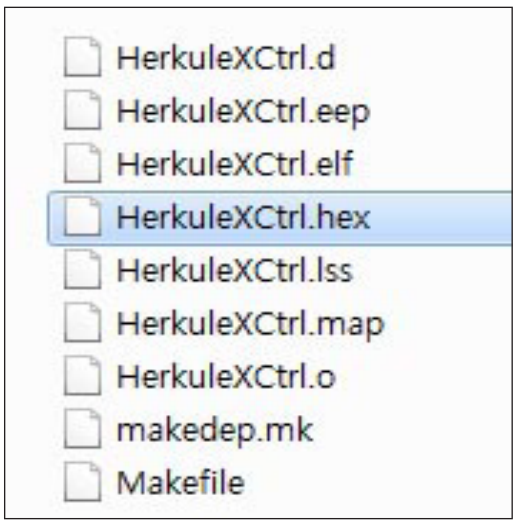
Select Build menu and click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.



02 Debug Folder

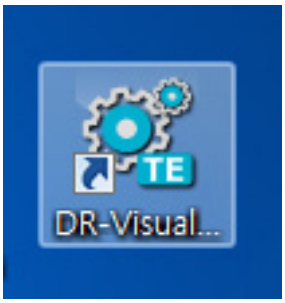
After bulid, debug folder will be created within the soruce folder. Click debug folder to view the folder content.



03 hex Check

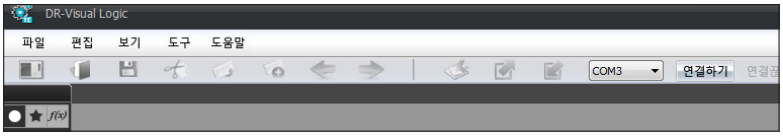
Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



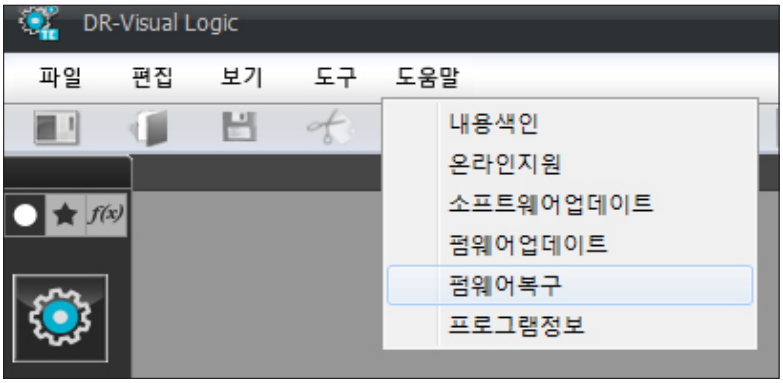
04 Run Visual Logic

Run DR-Visual Logic.



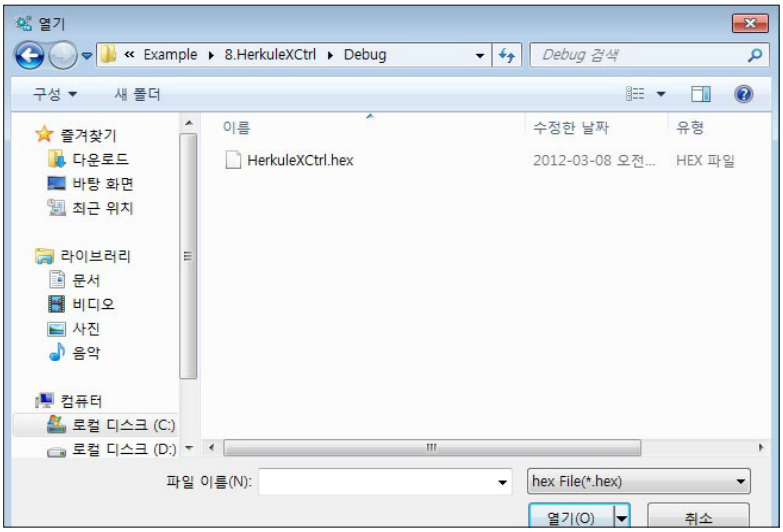
05 Connect

Select COM port and click Connect.



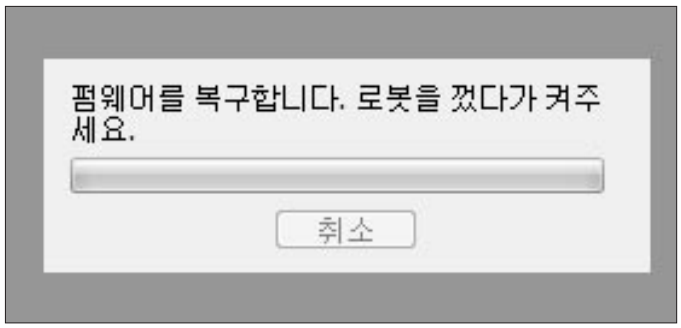
06 Firmware Restoration

Click Restore Firmware from the Help Menu.



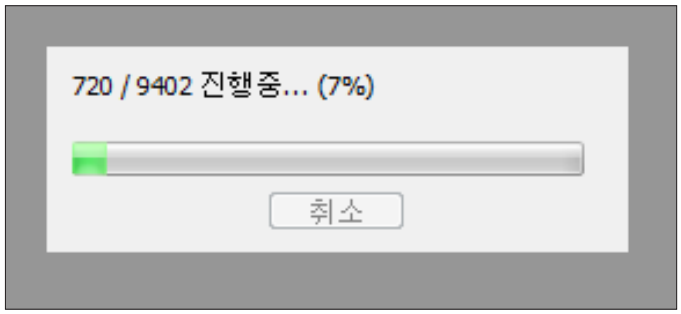
07 Load

Load Hex file from Debug folder.



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress. Popup window will disappear when restoration is complete.

Use Tera Term to start communication test.



10 Connect

Firmware has been ported properly if no changes are noticed with the LED after the downlod is complet. LED will start to blink if firmware was not ported properly.



11 Run

Motor will move to the left and right. When program is run after downloading the firmware, HerkuleX with ID 253 will start to move back and forth from position 400 to 624 every 1s. All DRC LED will be off if the servo motor is operating properly.



12 Disconnect

All LEDs will go off when servo motor is disconnected from the DRC. This means communication between the DRC and the motor has been terminated.

PART 04

Applying To The Robot

CH.1 : Running 16 Axis Humanoid Motion

Use herkulex library and motion data file to run motion on 16 axis Hovis Lite.

H/W Fundamentals

As mentioned previously in Chapter 8, ATmega128 installed in DRC has 2 USART ports, USART0 and USART1. USART0 is connected (parallel connection) to 5 4pin ports which are used for communicating with HerkuleX. HerkuleX requiring control can be connected to the DRC by cable and packets sent and received through USART0. Refer to the table below for summary of the packet commands that will be used in the example in this chapter.

Name	CMD	Description
RAM_WRITE	0x03	Write data to HerkuleX volatile register.
S_JOG	0x06	Controls HerkuleX position/speed. I_JOG command is used to set the operating time of each motor individually whereas S_JOG command sets identical operating time to all motors with all mo-tors arriving at the goal position as the same time. In I_JOG and S_JOG pacekt, 1byte size data is used to express time. At this time 1 = 11.2ms. In other words, if I_JOG packet con-tains 50 as time data, motor will operate for 50*11.2=560ms to move towards goal positon.

Summary of items in volatile register range used in this example are as follows.

Name	Address	Size	Description
Torque Control	52	1	Controls torque enable state. 0x00 : Free movement without any resistance 0x40 : Free movement with resistance 0x60 : Torque enabled state

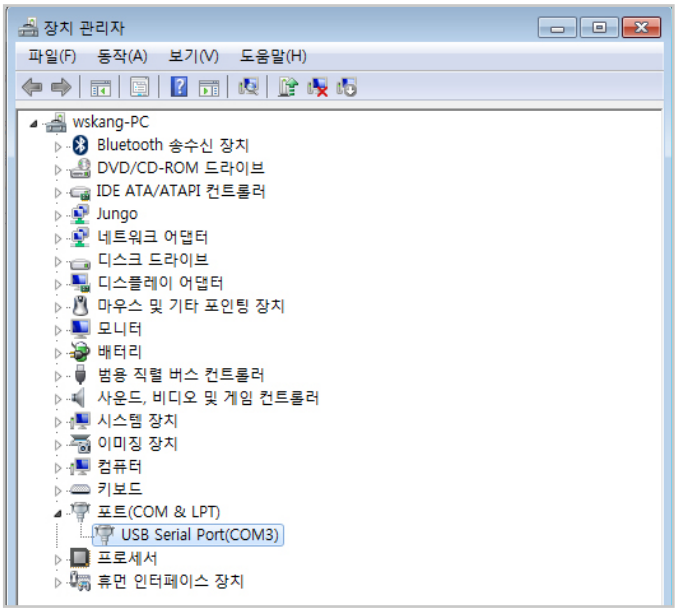
Refer to the HerkuleX manaul for more detailed information on protocols and register map.



Hardware Preparation

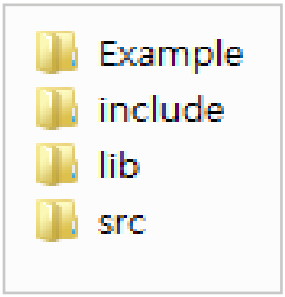
01 Connection

Connect PC to DRC using USB to Serial Cable. Conntect 16 axis humanoid Hovis Lite to PC.



02 COM Port Connection

My Computer 〉 Property 〉 Device Manager. Check COM connection.

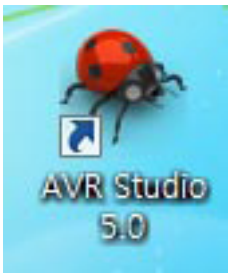


Running AVR Studio & Cod–

00 Library

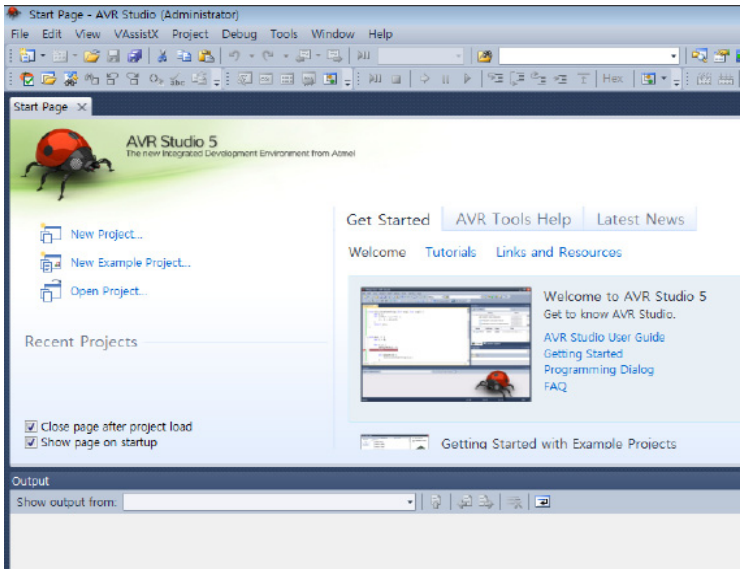
Basic libarary file is required to control Hovis Lite.

Picture shows the content of the folder which also includes the library.



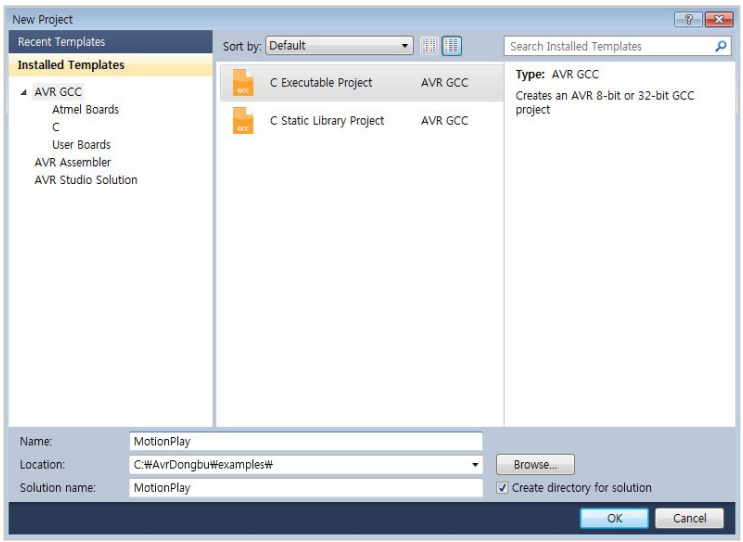
01 Run

Click icon to run AVR Studio.



02 New Project

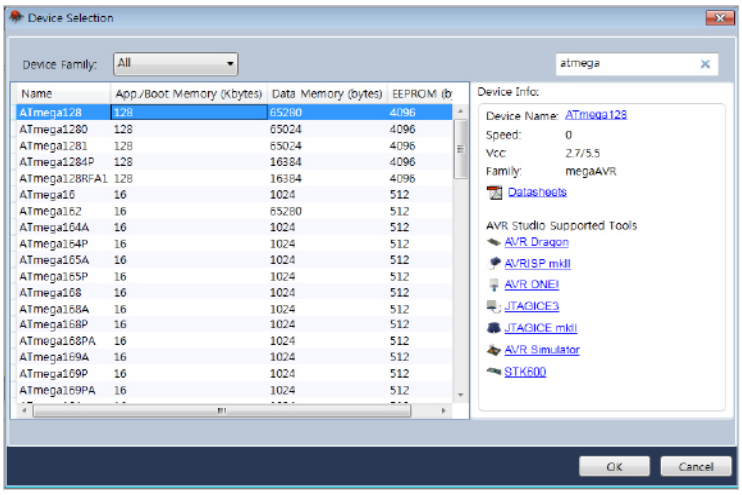
Click New Project.



03 Select GCC

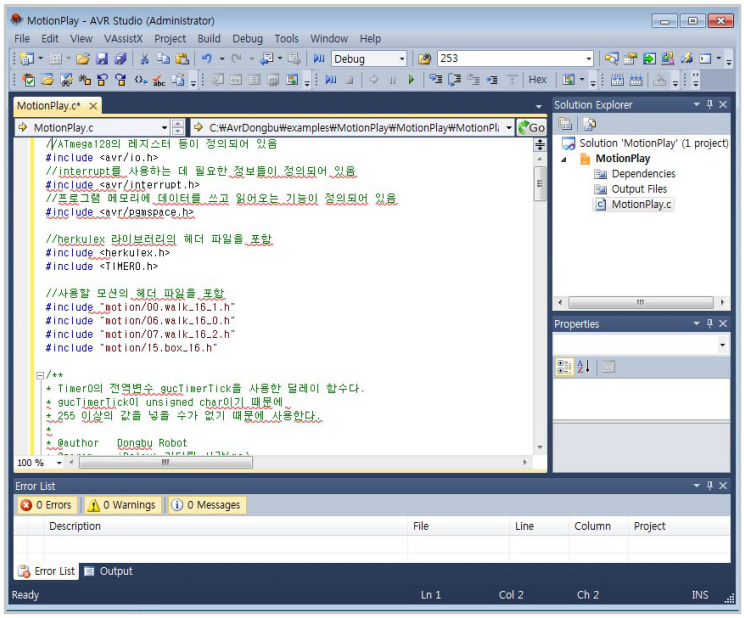
Select C Executable Project

Input 'Motionplay' as Name, Click Create directory for solution, Press OK button



04 Select ATmega

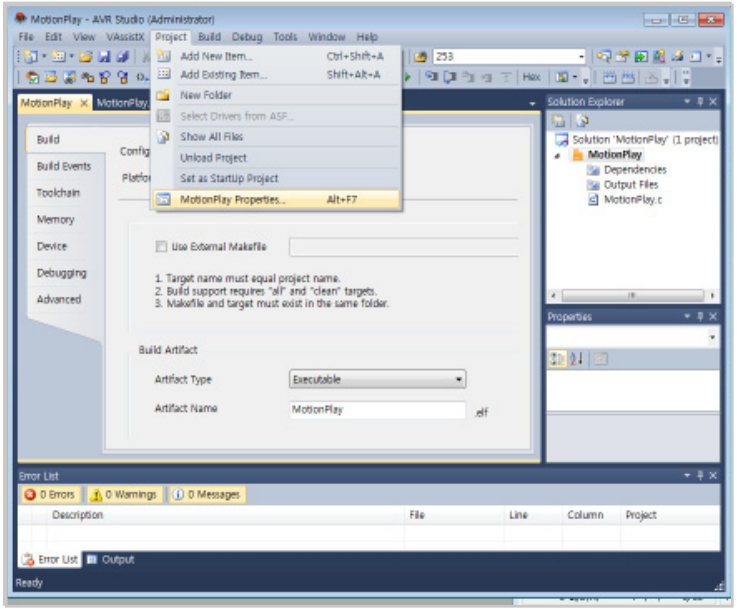
Seach for atmega, Select ATmega128 from the list.



05 Editor

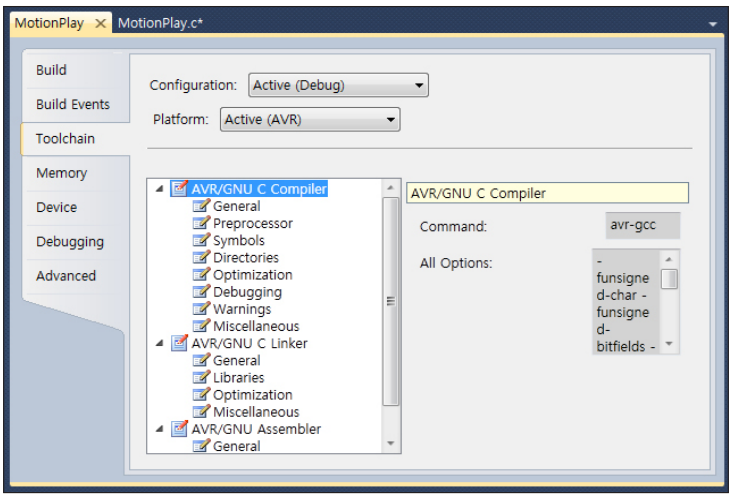
Start coding in the edit window.

Library Link & Setup



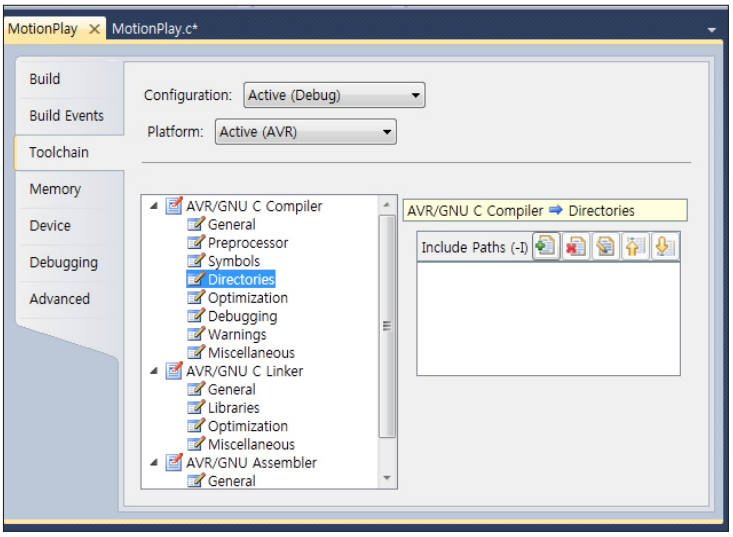
01 Setup

From the menu bar found at top of AVR Studio, click on Project to view the MotionPlay (Name dffers according to project name) property menu. Setup window opens up as shown on the left.



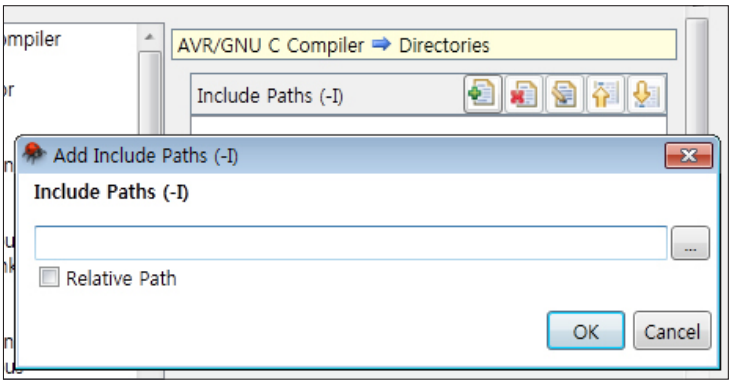
02 Toolchain

From the left tab, click Toolchain, and then select AVR/GNU C Compiler.



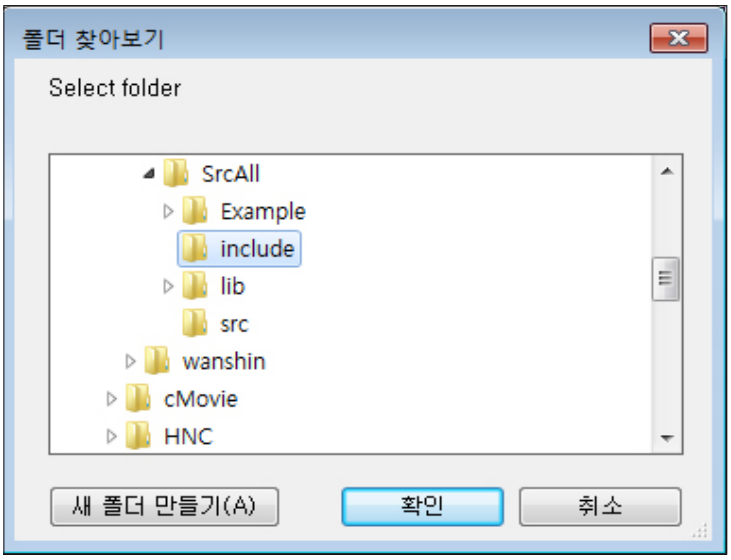
03 C Compiler

From the C Compiler folders, select Directories .



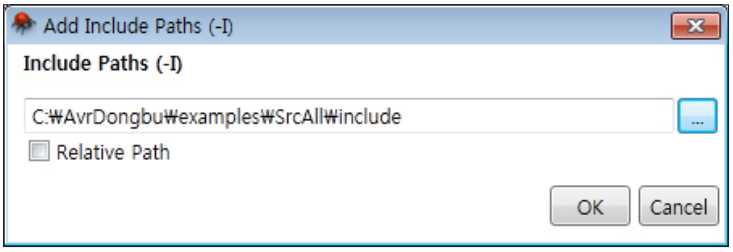
04 Path

click + button on right side of Include Paths to open the popup window. Deselect Relative Path and press right folder search button.



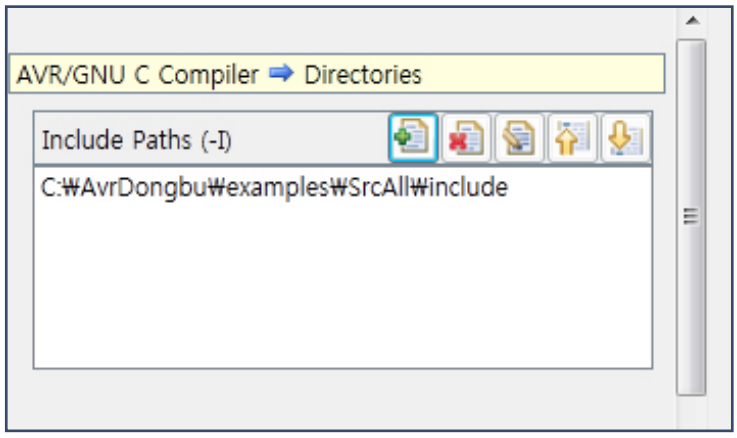
05 Include

From the supplied library, select include.



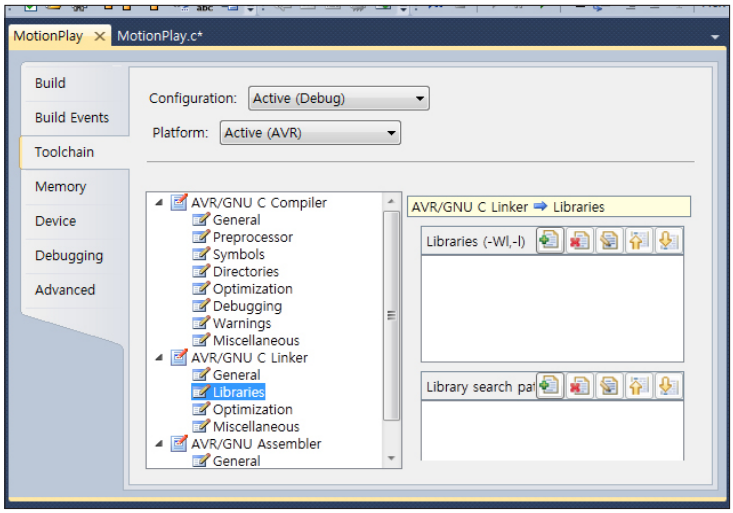
06 Include Complete

Include folder path selected.
Click OK button.



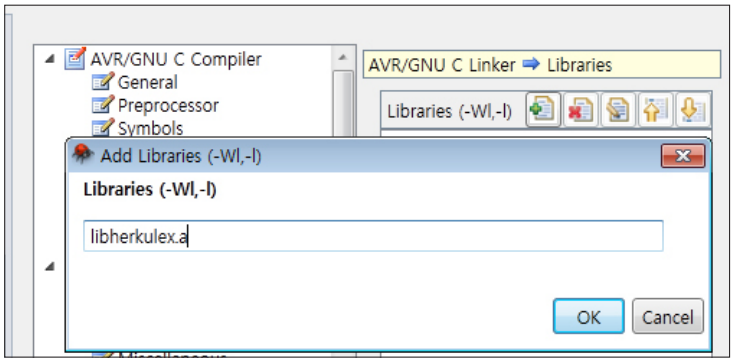
07 Path Complete

Include path selection complete.



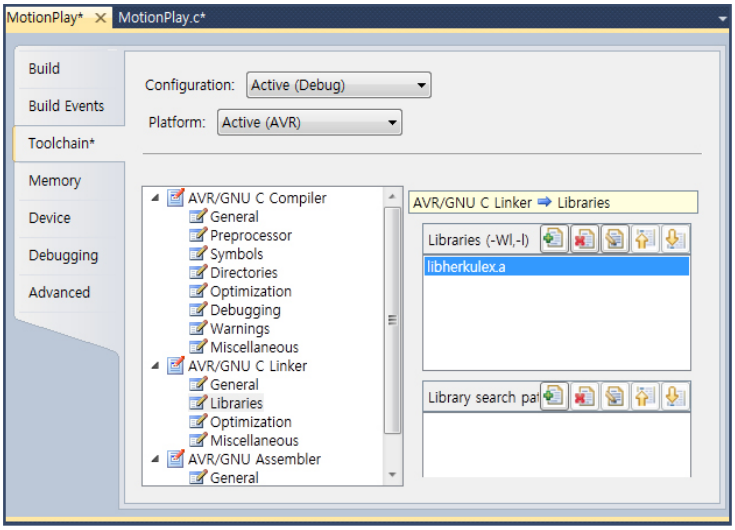
08 C Linker

From the C Linker folders, selet Libraries .



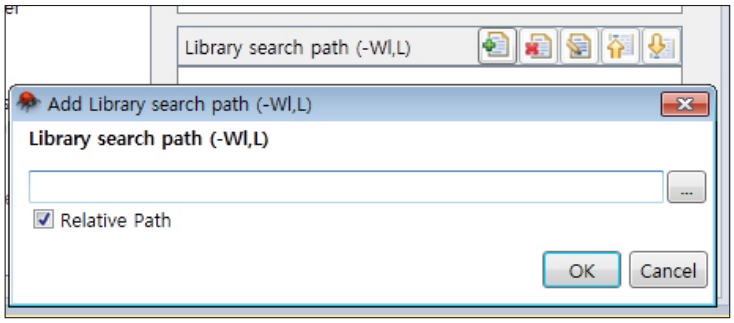
09 Libraries

Click + on the right side of Libraries window to open Libraries file name input window. Input libherkulex.a
Click OK button.



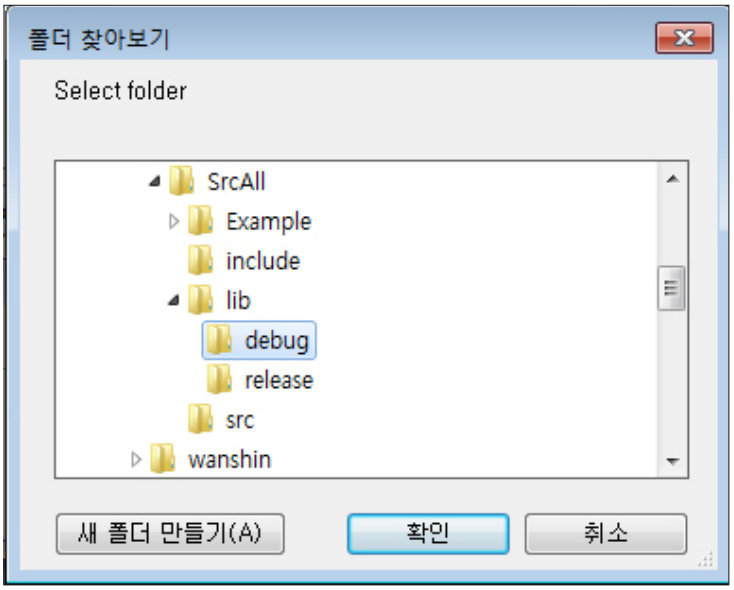
10 Search

Servo motor library file libherkulex.a link complete.
Next
Setup Library search path
Click right + button.



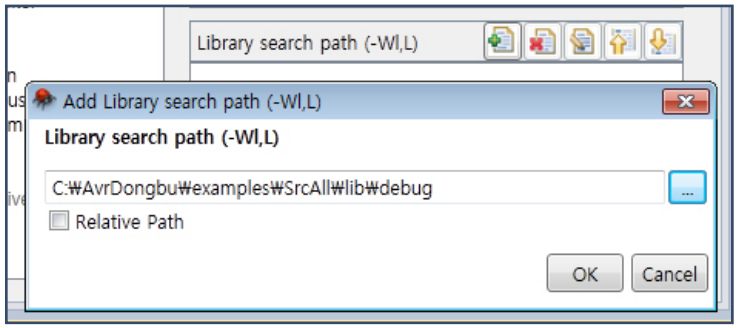
11 Path

Click folder search button on the right side of Library search path pop up window.
Click search button.



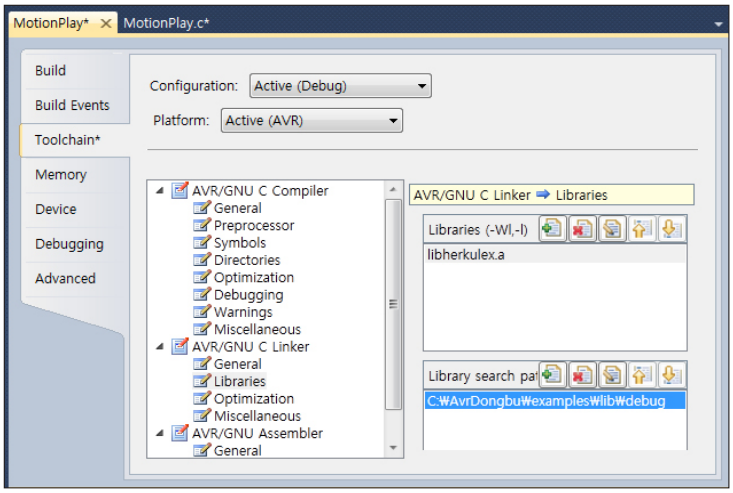
12 Debug

From the supplied library forlder, select debug below the lib.



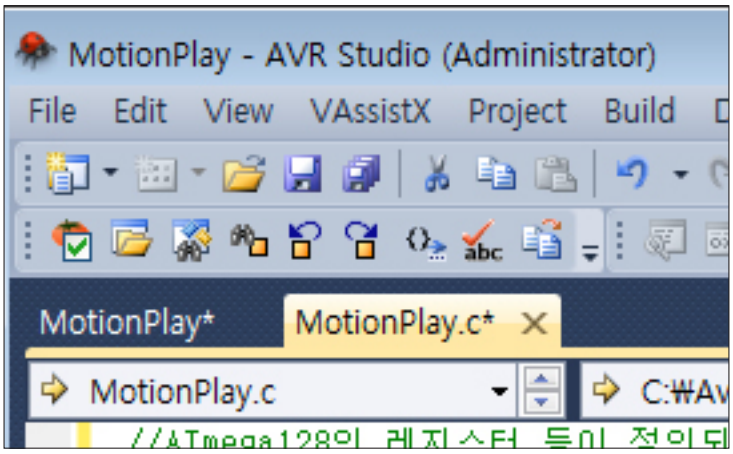
13 Path complete

Deselect Relative Path.
Library search path setup complete.
Click OK button.



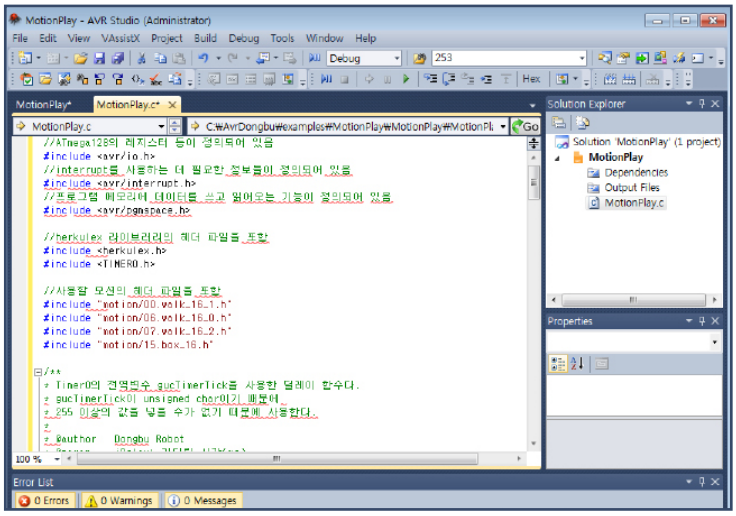
14 Link complete

C Linker library file and path setup complete.



15 Save

Press save button to save current setup values. Tab bar names changes from MotionPlay * –〉 MotionPlay .



16 Coding

Select MotionPlay.c tab bar and enter example below.

Example

When power is turned on, robot will repeatedly move forward 10 steps and start boxing.

00.Walk_16_1.h

```
#ifndef WALK_16_1_H_
#define WALK_16_1_H_

//Number of frames to play in current motion
prog_int16_t walk_16_1_FrameCnt = 48;

//Number of motors to control in current motion
prog_int16_t walk_16_1_MotorCnt = 16;

//Motor goal position value applicable to the frame
prog_int16_t walk_16_1_MotorData[48][16] = {
        {269,398,362,828,730,559,532,398,261,671,533,544,699,759,430,533},
        {267,381,379,832,713,575,523,402,262,674,526,536,697,761,426,526},
        {264,364,396,836,696,593,515,406,263,676,519,527,695,762,423,519},
        {262,346,414,840,678,610,506,410,265,678,512,518,692,764,419,512},
        {260,328,431,844,660,628,497,414,267,680,505,509,690,765,415,505},
        {257,311,449,848,643,645,488,417,269,682,498,501,687,766,411,498},
        {254,294,465,852,626,662,480,421,271,684,491,492,685,767,408,491},
        {252,279,481,856,610,678,472,425,273,686,484,484,682,768,404,484},
        {249,264,496,860,596,693,465,430,276,687,478,477,679,769,401,478},
        {246,251,509,864,582,706,458,434,278,689,473,470,676,769,397,473},
        {239,248,512,867,582,706,457,437,274,689,468,470,669,762,397,468},
        {232,248,512,866,585,703,457,436,268,688,464,472,663,755,398,464},
        {226,248,512,860,587,701,457,430,260,685,463,473,656,749,398,463},
        {220,248,512,851,589,699,457,421,250,680,463,474,650,743,398,463},
        {215,248,512,837,591,698,457,407,240,673,463,475,645,738,397,463},
        {210,248,512,826,592,697,457,396,240,659,463,475,641,736,396,463},
        {207,248,512,815,592,696,457,385,246,641,463,475,637,734,393,463},
        {204,248,512,800,592,697,457,370,249,625,463,475,634,735,390,463},
        {202,248,512,784,591,698,457,354,247,613,463,475,632,737,386,463},
        {201,248,512,767,589,699,457,337,241,605,464,474,631,741,381,464},
        {200,248,512,754,587,701,457,324,238,600,468,473,631,746,375,468},
        {200,251,509,750,588,701,458,320,248,593,473,473,630,752,369,473},
        {200,264,496,750,598,690,465,320,257,589,478,478,630,758,362,478},
        {199,279,481,753,610,678,472,323,267,590,484,484,630,765,355,484},
        {196,294,465,755,626,662,480,325,265,594,491,492,626,763,353,491},
        {192,311,449,757,643,645,488,327,263,598,498,501,622,762,350,498},
        {188,328,431,760,660,628,497,329,262,601,505,509,618,761,348,505},
```

```
        {184,346,414,762,678,610,506,332,260,605,512,518,614,759,346,512},
        {180,364,396,764,696,593,515,334,259,609,519,527,610,757,344,519},
        {176,381,379,767,713,575,523,337,258,613,526,536,607,755,342,526},
        {172,398,362,770,730,559,532,339,257,616,533,544,603,753,340,533},
        {168,414,346,772,745,543,540,342,256,620,540,552,599,751,338,540},
        {164,428,331,775,760,528,547,345,255,623,546,559,594,748,337,546},
        {160,442,318,778,773,515,554,348,255,627,551,566,590,746,335,551},
        {157,442,318,785,776,512,554,355,262,627,556,567,587,750,335,556},
        {158,439,321,792,776,512,552,361,269,626,560,567,588,756,336,560},
        {164,437,323,798,776,512,551,368,275,626,561,567,594,764,339,561},
        {173,435,325,804,776,512,550,374,281,626,561,567,603,774,344,561},
        {187,433,326,809,776,512,549,379,286,627,561,567,617,784,351,561},
        {198,432,327,814,776,512,549,383,288,628,561,567,628,784,365,561},
        {209,432,328,817,776,512,549,387,290,631,561,567,639,778,383,561},
        {224,432,327,820,776,512,549,390,289,634,561,567,654,775,399,561},
        {240,433,326,822,776,512,549,392,287,638,561,567,670,777,411,561},
        {257,435,325,823,776,512,550,393,283,643,560,567,687,783,419,560},
        {270,437,323,824,776,512,551,393,278,649,556,567,700,786,424,556},
        {274,436,323,824,773,515,551,394,272,655,551,566,704,776,431,551},
        {274,426,334,824,760,528,546,394,266,662,546,559,704,767,435,546},
        {271,414,346,825,745,543,540,394,259,669,540,552,701,757,434,540}
};

//Motor operating time for each frame
prog_int16_t walk_16_1_PlayTimeData[48] = {
        70,70,70,70,70,70,70,70,70,70,
        70,70,70,70,70,70,70,70,70,70,
        70,70,70,70,70,70,70,70,70,70,
        70,70,70,70,70,70,70,70,70,70,
        70,70,70,70,70,70,70,70
};

//Delay to the next frame for each frame
prog_int16_t walk_16_1_DelayData[48] = {
        -14,-14,-14,-14,-14,-14,-14,-14,-14,-14,
        -14,-14,-14,-14,-14,-14,-14,-14,-14,-14,
        -14,-14,-14,-14,-14,-14,-14,-14,-14,-14,
        -14,-14,-14,-14,-14,-14,-14,-14,-14,-14,
        -14,-14,-14,-14,-14,-14,-14,-14
};

#endif /* WALK_16_1_H_ */
```

MotionPlay.c

```
//CPU Clock select
#define F_CPU 16000000

//ATmega128 register defined
#include <avr/io.h>
//Information required for using interrupt defined
#include <avr/interrupt.h>
//Function for R/W data to program memory defined
#include <avr/pgmspace.h>

//Include herkulex libray header file
#include <herkulex.h>
#include <TIMER0.h>

//Include motion header file
#include "motion/00.walk_16_1.h"
#include "motion/06.walk_16_0.h"
#include "motion/07.walk_16_2.h"
#include "motion/15.box_16.h"

/**
* Delay function using Timer0 global variable gucTimerTick.
* Used since gucTimerTick is unsigned char and cannot input
* value greater than 255.
*
* @author       Dongbu Robot
* @param        iDelay: Delay time(ms)
* @return       None
* @version      1.0.0.0
*/
void delay_ms(int iDelay){
        //Repeat if iDelay > 0
        while(iDelay>0){
                //If iDelay > 500ms
                if(iDelay>500){
                        //Input 250 to gucTimerTick deduct 500 from iDelay.
                        gucTimerTick=250;
                        iDelay -= 500;
                        //Wait unitl gucTimerTick becomes 0 due toTimer0
                        while(gucTimerTick>0);
                }
                //Of iDelay is < 500ms
                else{
                        //Input iDealy/2 tp gucTimerTick amd set iDelay to 0.
                        gucTimerTick=iDelay/2;
                        iDelay = 0;
                        //Wait unitl gucTimerTick becomes 0 due toTimer0
                        while(gucTimerTick>0);
                }
        }
}
```

```
/**
* Function for running motion
* Read information declared in motion header file from the memory and
* send appropriate S_JOG packe to run motion
*
* @author       Dongbu Robot
* @param        FrameCnt: Number of motion frames to run
*               MotorCnt: Number of axis in the motion to be run
*               MotorData[FrameCnt][MotorCnt]: Motor goal postion for each frame
*               PlayTimeData[FrameCnt]: Motor operating time for each frame.
*               DelayData[FrameCnt]: Delay time value forme each frame
* @return       None
* @version      1.0.0.0
*/
void PlayMotion(        prog_int16_t FrameCnt, prog_int16_t MotorCnt,
                        prog_int16_t MotorData[FrameCnt][MotorCnt],
                        prog_int16_t PlayTimeData[FrameCnt],
                        prog_int16_t DelayData[FrameCnt])
{
        //Declare variable and structure
        unsigned char i;
        unsigned int uiCurrentFrame;
        DrsPacket stSendPacket;

        //Repeat from first frame to last frame
        for(uiCurrentFrame=0;uiCurrentFrame<FrameCnt;uiCurrentFrame++){
                //Setup packet size, minimum size(7) + (Motor quantity * Data size/motor(4)) +
operating time(1)
                stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_S_JOG_STRUCT_SIZE * MotorCnt + 1;
                //Set as 0XFE since command is for multiple servos
                stSendPacket.ucChipID = BROADCAST_ID;
                //Setup as S_JOG command
                stSendPacket.ucCmd = CMD_S_JOG;
                //Read PlayTime Data applicable to current frame, divide by 112 and roundoff
and substitute
                stSendPacket.unData.stSJogData.ucPlayTime =
                                (pgm_read_word(PlayTimeData+uiCurrentFrame)*10+56)/112;
                //Repeat from fist motor to the last motor
                for(i=0;i<MotorCnt;i++){
                        //Read Motor Data applicable to current frame and motor and susti-
tute₩                   stSendPacket.unData.stSJogData.stSJog[i].stJog.uiValue =
                                pgm_read_word(*(MotorData+uiCurrentFrame)+i);
                        stSendPacket.unData.stSJogData.stSJog[i].stJog.reserved = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucStopFlag = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucMode = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedGreen = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedBlue = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedRed = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucJogInvalid = 0;
                        stSendPacket.unData.stSJogData.stSJog[i].stSet.reserved = 0;
                        //Since motion data is in motor ID 0~15 sequence susbstitute i
                        stSendPacket.unData.stSJogData.stSJog[i].ucId = i;
                }
```

```
                    //Send constructed packet
                    hklx_SendPacket(stSendPacket);

                    //Delay for  PlayTimeData+DelayData applicable to current frame and then run
next frame
                    delay_ms(pgm_read_word(PlayTimeData+uiCurrentFrame)+
                                        pgm_read_word(DelayData+uiCurrentFrame));
          }

          return;
}

int main(void)
{
          //Declare packet structure for saving packet to be sent/received.
          DrsPacket stSendPacket;

          //Setup Port C I/O direction
          DDRC = 0b01111111;
          //Setup Port C ouput value
          PORTC = 0b01111111;

          //Deactivate all interrupts
          cli();

          //Initialize to use HerkuleX
          hklx_Init(115200);

          //Activate all interrupts
          sei();

          //Construct packet for writing 0x60 to Torque Control to torqure on
          stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
          stSendPacket.ucChipID = BROADCAST_ID;
          stSendPacket.ucCmd = CMD_RAM_WRITE;
          stSendPacket.unData.stRWData.ucAddress = 52;
          stSendPacket.unData.stRWData.ucLen = 1;
          stSendPacket.unData.stRWData.ucData[0] = 0x60;

          //Send packet
          hklx_SendPacket(stSendPacket);

          while(1)
          {
                    //Start walkin motion(Forward start - forward * 3 - forward end)
                    PlayMotion(walk_16_0_FrameCnt, walk_16_0_MotorCnt,
                         walk_16_0_MotorData, walk_16_0_PlayTimeData, walk_16_0_DelayData);
                    PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt,
                         walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
                    PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt,
                         walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
                    PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt,
                         walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
                    PlayMotion(walk_16_2_FrameCnt, walk_16_2_MotorCnt,
                         walk_16_2_MotorData, walk_16_2_PlayTimeData, walk_16_2_DelayData);

                    //Pause 1s
                    delay_ms(1000);
```

```
                //Run boxing motion
                PlayMotion(box_16_FrameCnt, box_16_MotorCnt,
                        box_16_MotorData, box_16_PlayTimeData, box_16_DelayData);

                //Pause 1s
                delay_ms(1000);
        }

        //Return integer to fit function structure
        return 1;
}
```

Example Description

Refer to Chapter 8. Controlling HekuleX for explanation about TIMER0.h and herkulex which are related to libherkulex.a .lib– herkulex .a is one of the header files which is included from hMotionPlay.c using #include statement.. Since motion files such as 00.Walk_16_1.h which was included using #include statement all have same format, only 00.Walk_16_1.h will be explained.

00.Walk_16_1.h

```
#ifndef WALK_16_1_H_
#define WALK_16_1_H_

…

#endif /* WALK_16_1_H_ */
```

00.Walk_16_1.h and other motion files are surrounded by #ifndef statement and #endif. Like regular C language header files, check if WALK_16_1_H_ was declared using #define. If not, delcare WALK_16_1_H_ and compile the content of the header file. This prevents same motion header file from being included and compiled many times.

```
prog_int16_t walk_16_1_FrameCnt = 48;
```

In the motion file, one robot position or one instant is called frame. When frame is played in sequence according to the time, one motion is being played.00.Walk_16_1 is a motion comprised of 48 frames.

prog_int16_t in front of the variable shows variable type similar to unsigned char or unsigned int. prog_int16_t is a special type which will delcare the variable in the Program memory (flash) instead of the RAM. Motions declared in motion header file has large data size and if all motion data are declared as regular data and saved in RAM, 4kByte ATmega128 RAM will not be enough. Due to this reason, motion data are saved in the program memory and read when required.

If prog_int16_t types of variables are saved in the program memory, variable value cannot be changed while the program is being executed and the value will remain as delcared in the beginning. Since program memory contains progam being executed, if the program memory value changes during program execution, this could damage the program and cause malfunction. Since motion data values are fixed numbers, there is no danger of value being changed and therefore it is appropriate to save the data in the program memory.prog_int16_t type can only be used if 〈avr/pgmspace.h〉 file is included.

```
prog_int16_t walk_16_1_MotorCnt = 16;
```

To execute motion, number of motors comprising the robot has to be known. 00.Walk_16_1 motion is a motion with 16 axis..

```
prog_int16_t walk_16_1_MotorData[48][16] = {
        {269,398,362,828,730,559,532,398,261,671,533,544,699,759,430,533},
…
        {271,414,346,825,745,543,540,394,259,669,540,552,701,757,434,540}
};
```

Position value of each motor in each frame are saved in 2 dimensional array. There are 48 frames in ther motion with 16 motors., [48][16] 2 dimensional array is required.

```
prog_int16_t walk_16_1_PlayTimeData[48] = {
        70,70,70,70,70,70,70,70,70,70,
…
        70,70,70,70,70,70,70,70
};
```

Data showing the time robot has to move with applicable posture when S_JOG packet applicable to each frame in the motion file is sent to HekuleX. This data is in (ms) unit which has to be changed to 11.2ms unit to enter S_JOG packet.. 00.Walk_16_1 has motion time of 70ms,which when converted to 11.2ms unit is 70/11.2 = 6.25, In other words, when 6.25 is rounded off, 6 has to go into S_JOG packet.

```
prog_int16_t walk_16_1_DelayData[48] = {
        -14,-14,-14,-14,-14,-14,-14,-14,-14,-14,
…
        -14,-14,-14,-14,-14,-14,-14,-14
};
```

After motion file ends, delay is used to determine when the next frame will be sent. Delay data is also in (ms) unit. For example, when 00.Walk_16_1 motion running time is 70ms and delay is 0ms, next S_JOG packet starting the next frame will be sent 70ms after previous frame. However if the delay is 14ms, 70ms+14ms = 84ms will pass before the next packet is sent. In other words, packet will be sent 14ms after the robot movement ends. If motion delay is –14ms like this motion, 70ms–14ms=56ms will pass until the next packet is sent which means frame packet is sent before the robot motion ends. Th advantage to – Delay is smooth robot movement.

MotionPlay.c

```
#define F_CPU 16000000
```

#define statement is one of the preprocessor commands, #define command will replace any F_CPU found later with 16000000. ATmega128 in DRC runs at 16MHz clock speed. Command processing speed is dependent on the clock speed, this command sets the clock speed so that delay function and other fuctions execute properly.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
```

aDDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h header file. avr/interrupt.h header file has defined functions required for using the interrupt. . avr/pgmspace.h has defined function for reading/writing program memory. Function for reading program memory such as prog_int16_t variable and pgm_read_word() are defined so must be included

```
#include <herkulex.h>
#include <TIMER0.h>
```

herkulex.h header file must be always inlcuded. Header includes declarations and structures for creating and sending packets, and defined functions. TIMER0.h is also included to set Delay using gucTimerTick.

```
#include “motion/00.walk_16_1.h”
#include “motion/06.walk_16_0.h”
#include “motion/07.walk_16_2.h”
#include “motion/15.box_16.h”
```

Header filie containg current example motion data. File has motion frame number, axis number, notor position data, run time data, and delay data declared

```
void delay_ms(int iDelay){
        while(iDelay>0){
            if(iDelay>500){
```

```
                gucTimerTick=250;
                iDelay -= 500;
                while(gucTimerTick>0);
        }
        else{
                gucTimerTick=iDelay/2;
                iDelay = 0;
                while(gucTimerTick>0);
        }
    }
}
```

Delay funtion usingTIMER0. Has iDelayTick as mediating variable and expresses delay time in (ms)unit. gucTimerTick is a global variable declared in TIMER0.h, normally maintained as 0, when non 0 value is substituted, value decreases by 1 every 2ms due to TIMER) overflow service routine defined in libherkulex.a. For example, to wait 10ms, input 5 in gucTimerTick and wait until gucTimerTick becomes 0. This is how delay_ms function is operates. gucTimerTick is a unsigned char, in other words, 1byte variable without sign. Therefore it can only have values between 0~255. 255 in gucTimerTick is equal to 510ms. gucTimerTick can only operate 0.5 ms with single substitution. Therefore, function cuts delay value iDelay and substitutes gucTimerTick in500ms segments. If iDelay is larger than 500, 250 is sustituted in gucTimerTick and iDelay is changed to value with 500 deducted and using while(gucTimerTick〉0); delay is repeated continuously as long as gucTimerTick is 〉 0 and exits while statement when gucTimerTick becomes 0.When iDelay is 500 or smaller, iDelay/2 is substituted in gucTimerTick and sets iDelay value to 0 and waits until the gucTimerTick becomes 0 using the while statement.This process is surrounded by while(iDelay〉0) and repeats as long as iDelay is 〉 0. While statement ends when iDelay becomes0 and delay_ms function ends.

```
void PlayMotion(        prog_int16_t FrameCnt, prog_int16_t MotorCnt,
                        prog_int16_t MotorData[FrameCnt][MotorCnt],
                        prog_int16_t PlayTimeData[FrameCnt],
                        prog_int16_t DelayData[FrameCnt])
```

Next function is PlayMotion function. Variables declared in motion header file go in as mediating variable. In sequence, frame number, axis number, motor dat, run time, delay time. MotorData, PlayTimeData, DelayData is name of the array. and array name is the pointer to the first element in the Array. Array data declared in program memory has to use pgm_read_word() or other speical function to read the data.

```
unsigned char i;
unsigned int uiCurrentFrame;
DrsPacket stSendPacket;
```

Declare variable. i and uiCurrentFrame are variables declared to to repeat in the for statement up to each motor number and frame number. stSendPacket is a space for saving S_JOG packet to be sent.

```
for(uiCurrentFrame=0;uiCurrentFrame<FrameCnt;uiCurrentFrame++){
                                    …
}
```

For statement repeating up to the number of existing frame. Within the for statement, frame is processed one at a time and motor value and run time applicable to each frame is read and sent to the motor in S_JOG packet. Run time + delay is added to delay_ms function and dealyed for that time.When delay_ms function ends next frame wil l be processed.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_S_JOG_STRUCT_SIZE * MotorCnt + 1;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_S_JOG;
```

Section for setting up packet id, packet size, and command for the packet to be sent from current frame. Following the S_JOG structure, packet size is standard 7byte(MIN_PACKET_SIZE) + 4bytes for motor *number of motors+runtime 1byte. If MotorCnt is 16 packet size is 7+14*16+1=72.

```
stSendPacket.unData.stSJogData.ucPlayTime = (pgm_read_word(PlayTimeData+uiCurrentFrame)*10+56)/112;
```

Section for setting S_JOG packet run time. First, pgm_read_word() is a function reading program memory value and it is declared in avr/pgmspace.h. Program memory address to be read goes into mediating variable. PlayTimeData is name of the array received by mediating variable of this function. It was mentioned previously that Array name is the address of the first element. In other words,PlayTimeData is same as &PlayTimeData[0]. By this principal, (PlayTimeData+i) is same as &PlayTimeData[i].In other words, mediating variable (PlayTimeData+uiCurrentFrame) going into pgm_read_word is same as &PlayTimeData[uiCurrentFrame], Therefore, pgm_read_word (PlayTimeData+uiCurrentFrame) is part reading PlayTimeData[uiCurrentFrame] data.In this formula, PlayTimeData[uiCurrentFrame] data is read and multiplied by 10 and 56 added and then divided by 112. This is PlayTimeData[uiCurrentFrame] divided by 11.2 calculated using integer numbers. In PlayTimeData, run time is saved in ms units, When being entered in S_JOG packet, data has to be dividedd by 11.2ms units. Simply dividing by 11.2 slows down calculation speed as it is a real number calculation. Due to AVR processor properties real number calculation is slower than integer calculation by multiples of 10. Therefore, instead of PlayTimeData[uiCurrentFrame]/11.2, 11.2 = 112/10 is used to make two integer calculations PlayTimeData[uiCurrentFrame]*10/112. However, because numbers below decimal point is deleted automaically, it is same as reading the PlayTimeData / 11.2 and throwing out the all the numbers below the decimal point.In other words if run time divided by 11.2 falls between 6.0~6.999, all numbers within the range will become 6.This might be a small error but it could cause robot to act unexpectdly during motion. To resolve this problem, when 56 is added prior to division by 112, range of real calculated result of 6.0~6.9999…becomes 6.5~7.4999…and real run time based calculateion 6.0~6.4999…becomes 6, and 6.5~6.9999…becomes 7. Result would be same as rounding off a number. To summarize, above code reads PlayTimeData[uiCurrentFrame] data from the program memory and divides the data by 11.2ms and the result is rounded and susbstitued into S_JOG packet operating time.

```
for(i=0;i<MotorCnt;i++){
        stSendPacket.unData.stSJogData.stSJog[i].stJog.uiValue =
                        pgm_read_word(*(MotorData+uiCurrentFrame)+i);
        stSendPacket.unData.stSJogData.stSJog[i].stJog.reserved = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucStopFlag = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucMode = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedGreen = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedBlue = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucLedRed = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.ucJogInvalid = 0;
        stSendPacket.unData.stSJogData.stSJog[i].stSet.reserved = 0;
        stSendPacket.unData.stSJogData.stSJog[i].ucId = i;
}
```

Increase i from 0 to MotorCnt-1 and fill S_JOG motor data section. Motor value we currently need is MotorData[uiCurrentFrame][i]. pgm_read_word needs mediating variable as pointer, &MotorData[uiCurrentFrame][i] has to be entered. &MotorData[uiCurrentFrame][i] is same as (*(MotorData+uiCurrentFrame)+i).

2 dimensional array MotorData[FrameCnt][MotorCnt] could be thought of as FrameCnt number of arrays composed of MotorCnt number or integers ,MotorData could be thought of as pointers pointer.. As PlayTimeData+uiCurrentFrame) was same as &PlayTimeData[uiCurrentFrame], (MotorData+uiCurrentFrame) = &MotorData[uiCurrentFrame], when * is added tob both sides *(MotorData+uiCurrentFrame) becomes MotorData[uiCurrentFrame]. MotorData[uiCurrentFrame] could be seen as pointer showing the start of array composed of ucMotorCnt number of int. When i is added to this value. as (PlayTimeData+uiCurrentFrame) was same as &PlayTimeData[uiCurrentFrame], (*(MotorData+uiCurrentFrame)+i) = &MotorData[uiCurrentFrame][i]. Therefore, pgm_read_word(*(MotorData+uiCurrentFrame)+i) section reads applicable frame number uiCurrentFrame and motor number i motor value from the program memory and eneters it in the applicable section of the S_JOG packet. The section below is for packet flag setup and not very relavant. stSendPacket.unData. stSJogData.stSJog[i].ucId = i; is for substituting Motor ID with i.

```
hklx_SendPacket(stSendPacket);
```

Call hklx_SendPacket() functio to send constructed S_JOG .

```
delay_ms(pgm_read_word(PlayTimeData+uiCurrentFrame)+pgm_read_word(DelayData+uiCurrentFrame));
```

This section involves wating for time to send the next frame. Use pgm_read_word(PlayTimeData+uiCurrentFrame) to read current frame run time once again. Use pgm_read_word(DelayData+uiCurrentFrame) to read current frame dealy and add two and send to delay_ms and wait for time to send next packet.
When the process is repeated on all frames from when uiCurrentFrame is 0 to when FrameCnt–1. Motion play will end and PlayMotion function willl be completed.

main function description.

```
DrsPacket stSendPacket;
```

Declare variable to be used in main function.Declare variable to be used when packet is sent as DrsPacket struct type.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Setup Port C pins 0~6 for output and set output value to 1. All LEDs will be turned off.

```
cli();
hklx_Init(115200);
sei();
```

Use cli() command to disable all interrupts, interrupts are disabled to prevent error by an interrupt while settings are be–ing changed. Use hklx_Init function to initalize Timer/Counter0 and USART0. Use baud rate of 115200 as the parameter value. Finally, use sei() command to enable all interrupts.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_RAM_WRITE;
stSendPacket.unData.stRWData.ucAddress = 52;
stSendPacket.unData.stRWData.ucLen = 1;
stSendPacket.unData.stRWData.ucData[0] = 0x60;

hklx_SendPacket(stSendPacket);
```

Torque must be enabled so that motor can be controlled to move on its own. Codes shown above are for writing and sending RAM_WRITE packet to enable the motor torque. The size of the packet MIN_PACKET_SIZE(7byte), address + length + data(1byte) equals MIN_PACKET_SIZE+3(10bytes), motor ID is 253. CMD is CMD_RAM_WRITE(0x03). Set ucAddress, ucLen, and ucData[0] values to 52, 1, 0x60, these three belong to structure stRWData which in turn is a member of the union unData. Finally, call function hklx_SendPacket() to send the packet.

```
PlayMotion(walk_16_0_FrameCnt, walk_16_0_MotorCnt, walk_16_0_MotorData, walk_16_0_PlayTimeData, walk_16_0_DelayData);
PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt, walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt, walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
PlayMotion(walk_16_1_FrameCnt, walk_16_1_MotorCnt, walk_16_1_MotorData, walk_16_1_PlayTimeData, walk_16_1_DelayData);
PlayMotion(walk_16_2_FrameCnt, walk_16_2_MotorCnt, walk_16_2_MotorData, walk_16_2_PlayTimeData, walk_16_2_DelayData);
```

This section is the content of the while statement. Within the while statement, motion will continue to run. First, forward start motion (walk_16_0) will be played. Next, forward motion(walk_16_1) will be played 3 times and then forward stop motion (walk_16_2) will be played. Robot will walk 10 steps and then stop.

```
delay_ms(1000);
```

Robot will wait 1s after forward motion ends.

```
PlayMotion(box_16_FrameCnt, box_16_MotorCnt, box_16_MotorData, box_16_PlayTimeData, box_16_DelayData);
```

Boxing motion(box_16) will run.

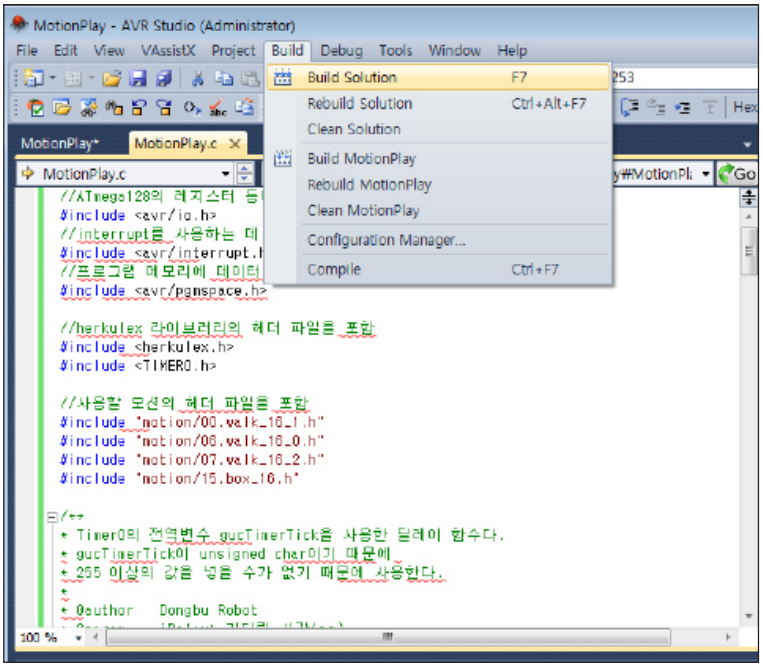
```
delay_ms(1000);
```

Robot will wait 1s after boxing motion ends.

Above steps will run continuosley within the while statement. Robot will walk 10 steps forward and then play boxing motion and then repeat the process again.

Run

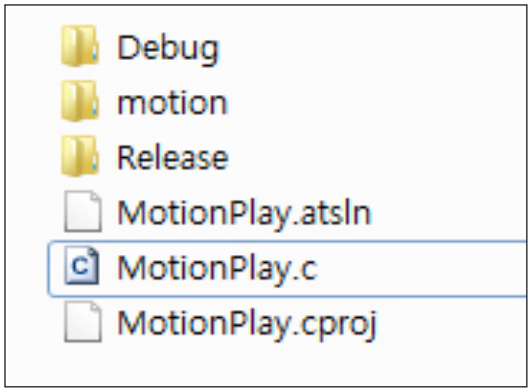
Connect DRC to 16 axis Hovis Lite, download firmware and run. Robot will walk 10 steps forwrad and then play boxing motion and then repeat the process again.



01 Build

Select Build menu and then click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.



02 Debug Folder

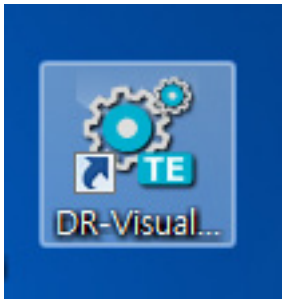
After bulid, debug folder will be created within the soruce folder Click debug folder to view the folder content



03 hex Check

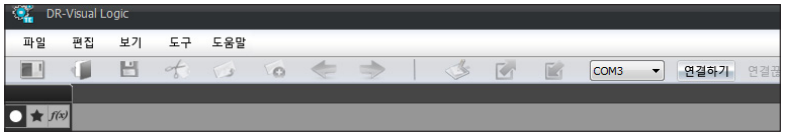
Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



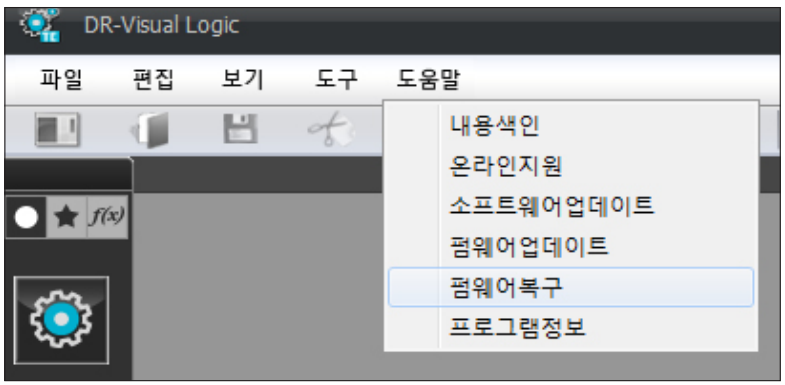
04 Visual Logic

Run DR-Visual Logic



05 Connect

Select COM port and click Connect.



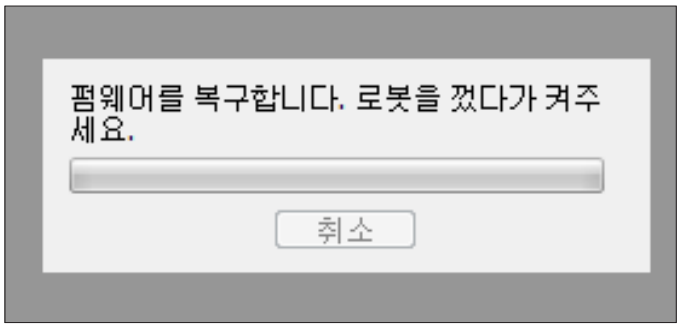
06 Restore Firmware

Click Restore Firmware in the Help Menu.



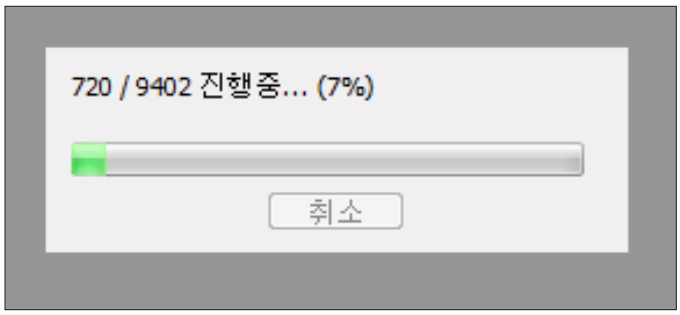
07 Load

Load Hex file from Debug folder



08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.



10 Run

Connect DRC to 16 axis Hovis Lite, download firmware and run. Robot will walk 10 steps forwrad and then play boxing motion and then repeat the process again.

CH.2 : Using IR Remote Control To Control The Dump Truck

Use herkulex libray, irremocon library to control the dump truck

H/W Fundamentals

In this example, we will use the HerkuleX library and IR remote control library simultaneously to control the dump truck robot. Refer to Chapter 8 for HerkuleX control information and Chapter 12 for Remote control information.

Summary of the packet commands that will be used in this example is as follows.

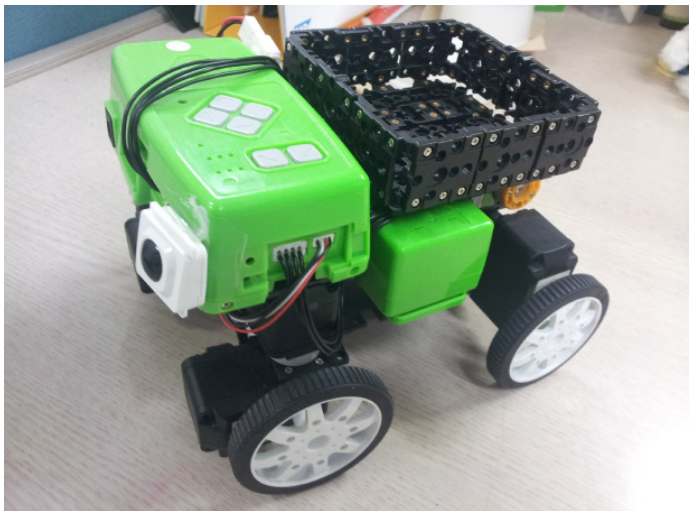
Name	CMD	Description
RAM_WRITE	0x03	Write data to HerkuleX volatile register.
I_JOG	0x05	Controls HerkuleX position/speed. I_JOG command is used to set the operating time of each motor individually whereas S_JOG command sets identical operating time to all motors with all mo–tors arriving at the goal position as the same time. In I_JOG and S_JOG packet, 1byte size data is used to express time. At this time 1 = 11.2ms. In other words, if I_JOG packet con–tains 50 as time data, motor will operate for 50*11.2=560ms to move towards goal positon.

Summary of items in volatile register range used in this example are as follows.

Name	Address	Size	Descripton
Torque Control	52	1	Controls torque enable state. 0x00 : Free movement without any resistance 0x40 : Free movement with resistance 0x60 : Torque enabled state

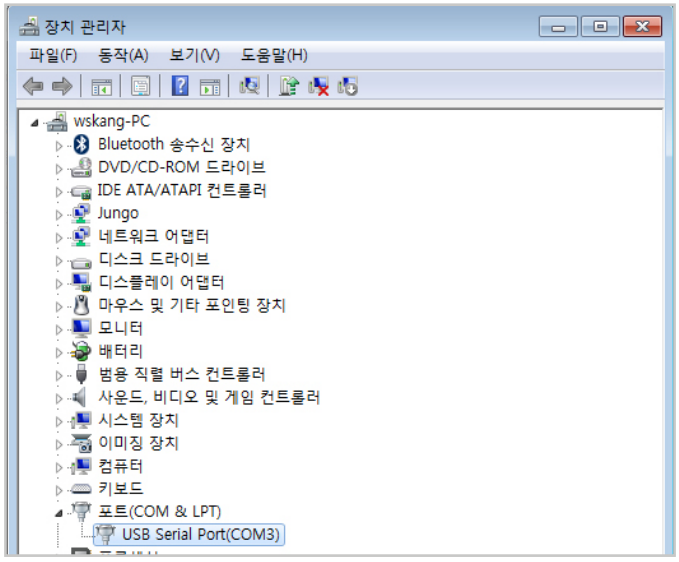
Refer to the HerkuleX manual for more detailed information on protocols and register map.

Hardware Preparation



01 Connection

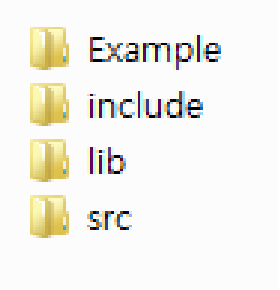
Connect PC to DRC using USB to Serial Cable. Connect dump truck to PC.



02 COM port Connection

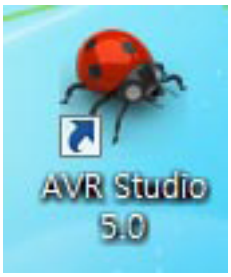
My Computer 〉 Property 〉 Device Manager. Check COM connection.

AVR Studio Preparation &



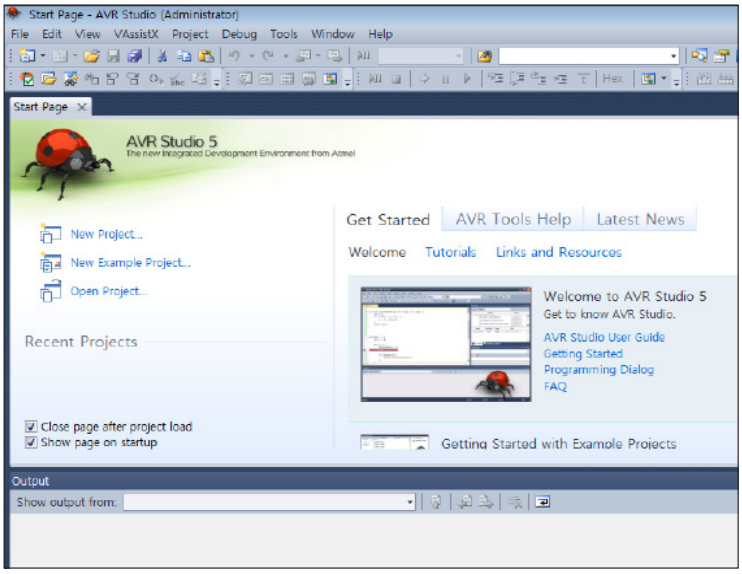
00 Library

Basic libarary file is required to con-trol dump truck
Picture shows the content of the folder which also includes the library.



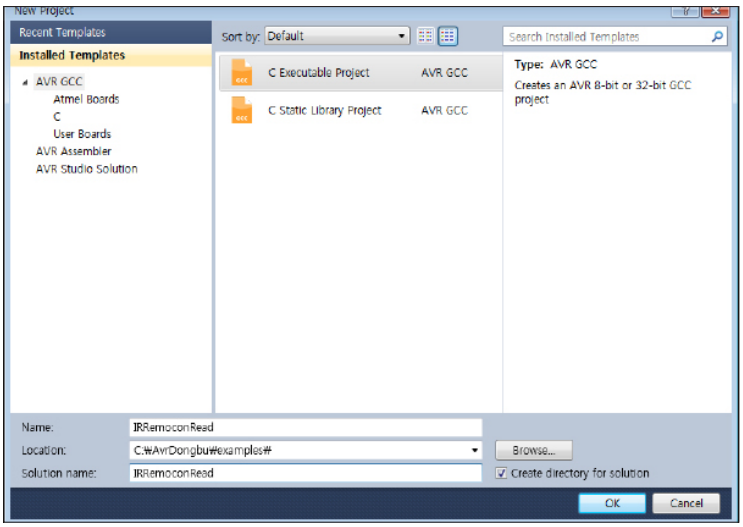
01 Run

Run AVR Studio



02 New Project

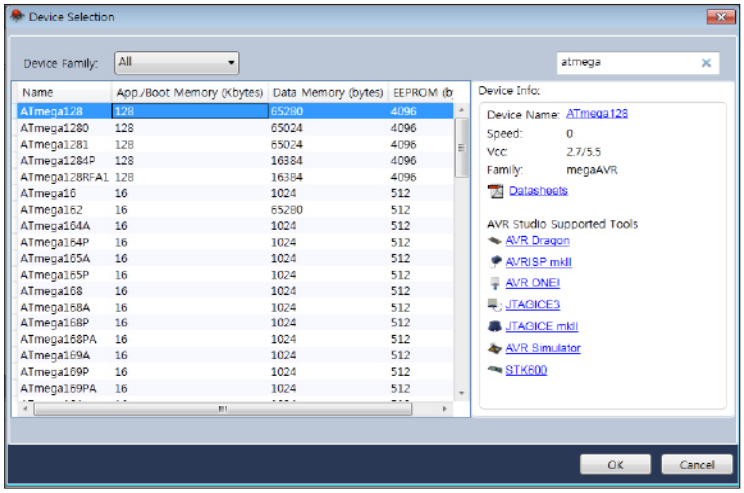
Click New Project .



03 Select GCC

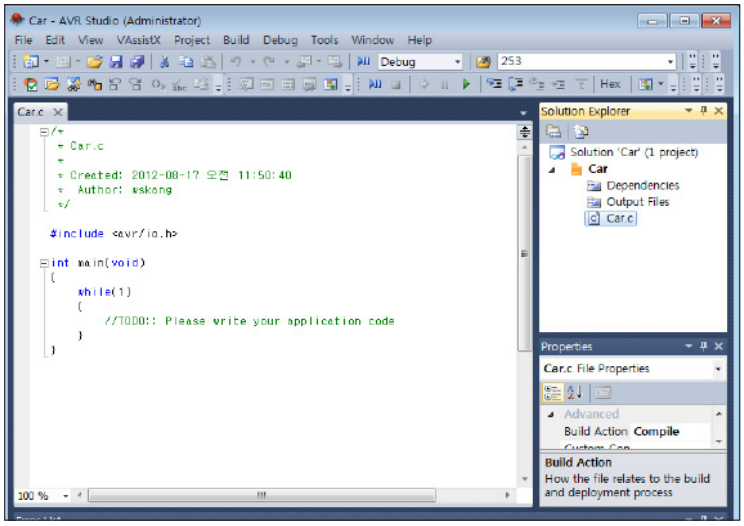
Select C Executable Project

Input 'Car' as Name, Click Create directory for solution, Press OK button



04 Select ATmega

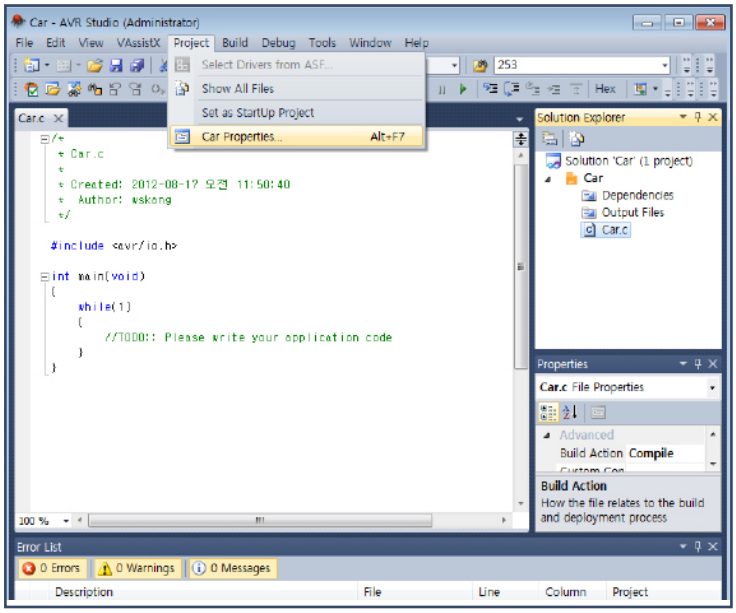
Seach for atmega, Select ATmega128 from the list.



05 Editor

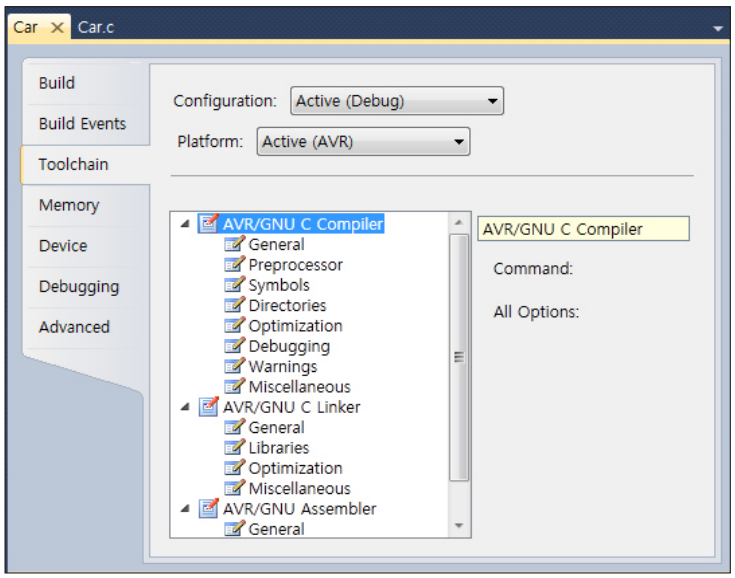
Start coding in the edit window

Library Link & Setup



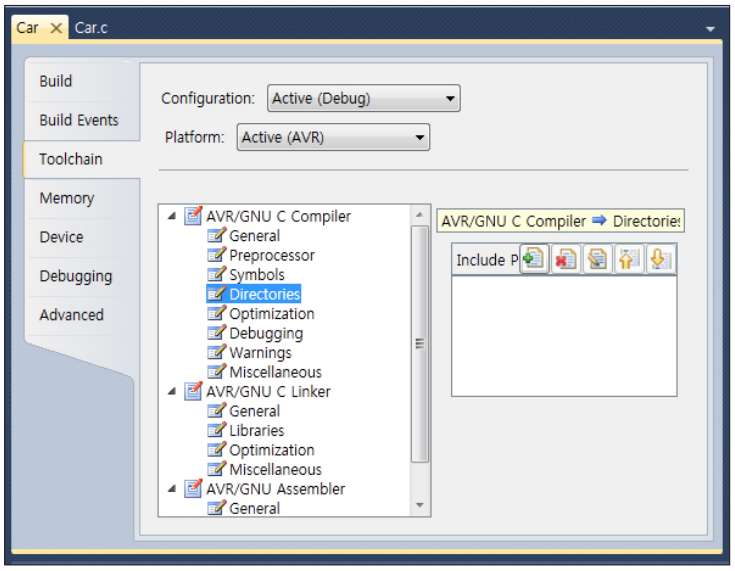
01 Setup

From the menu bar found at top of AVR Studio, click on Project to view the Car (Name dffers according to project name) property menu. Setup window opens up as shown on the left.



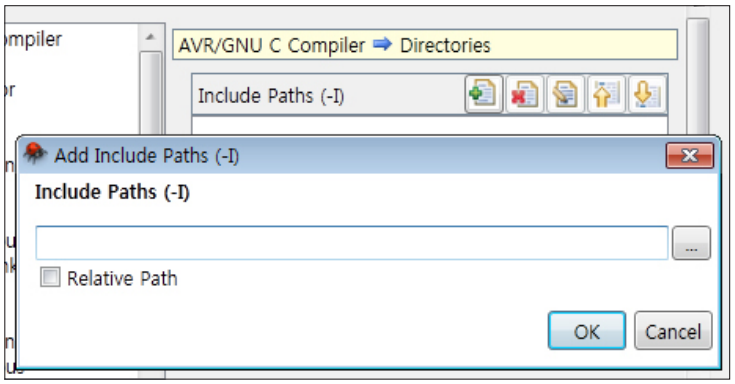
02 Toolchain

From the left tab, click Toolchain, and then select AVR/GNU C Com-piler.



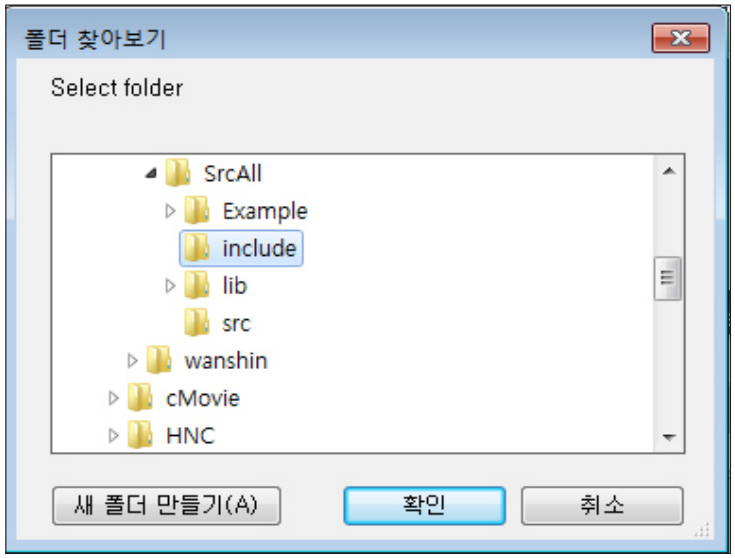
03 C Compiler

From the C Compiler folders, select Directories .



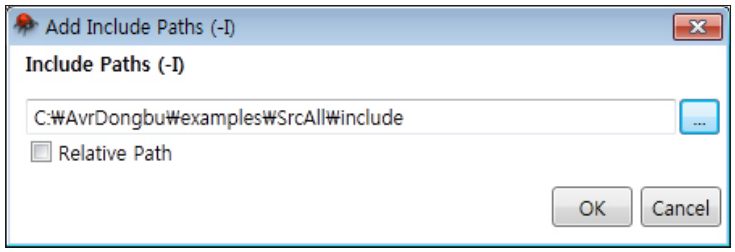
04 Path

click + button on right side of Include Paths to open the popup window. Deselect Relative Path and press right folder search button.



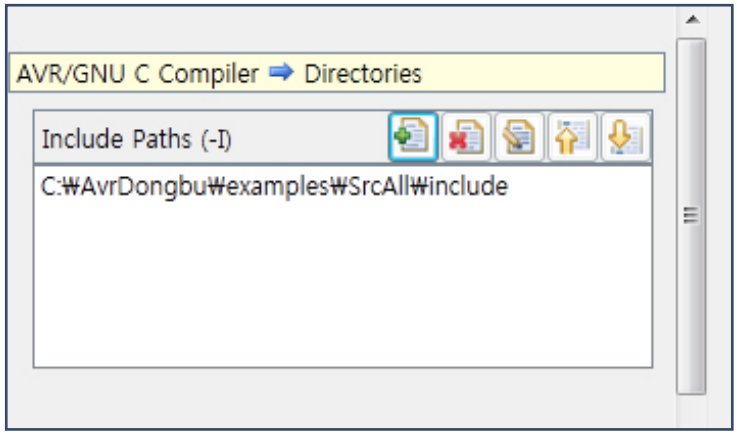
05 Include

From the supplied library, select include.



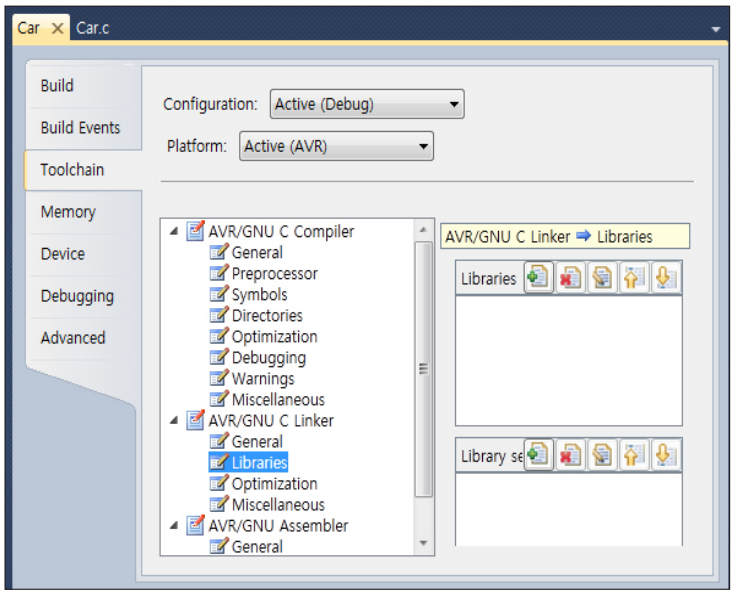
06 Include Complete

Include folder path selected.
Click OK button.



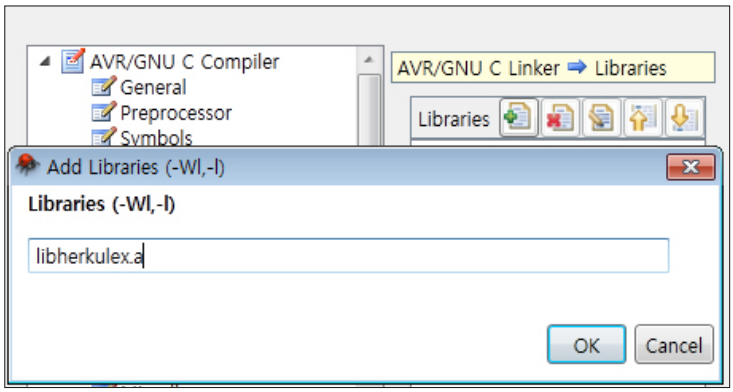
07 Path Complete

Include path selection complete.



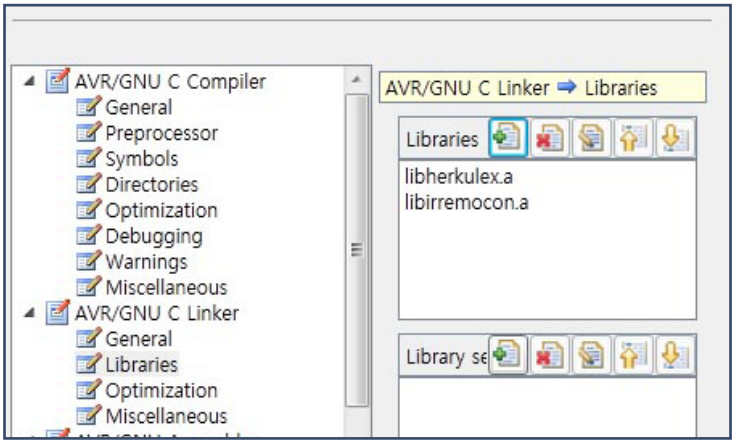
08 C Linker

From the C Linker folders, selet Li-braries .



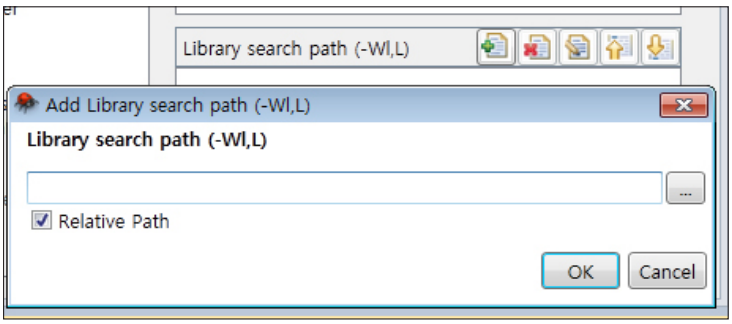
09 Libraries

Click + on the right side of Libraries window to open Libraries file name input window. Input libherkulex.a
Click OK button.



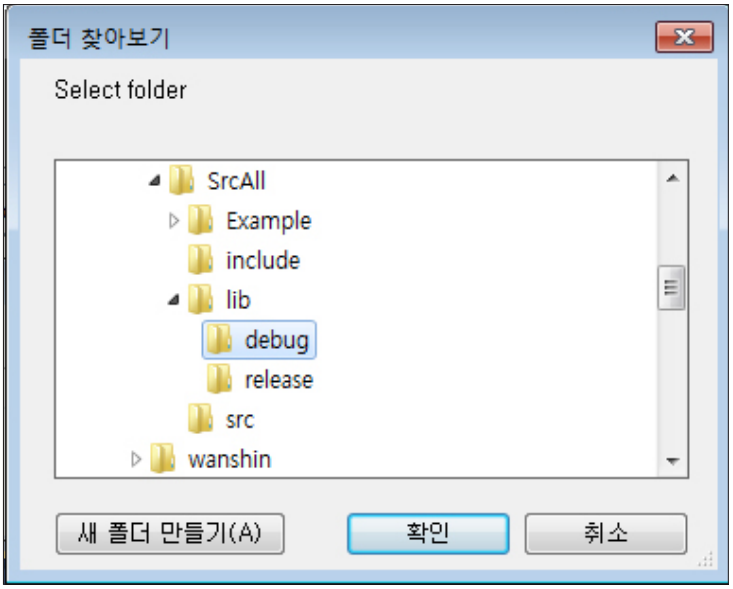
10 Search

Servo motor library file libherkulex.a link complete.
Next
Setup Library search path
Click right + button.



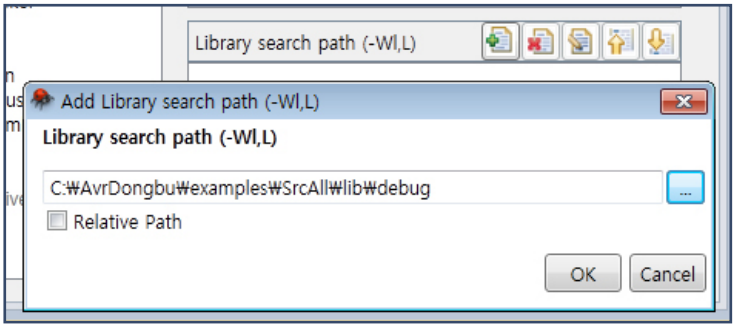
11 Path

Click folder search button on the right side of Library search path pop up window.
Click search button.



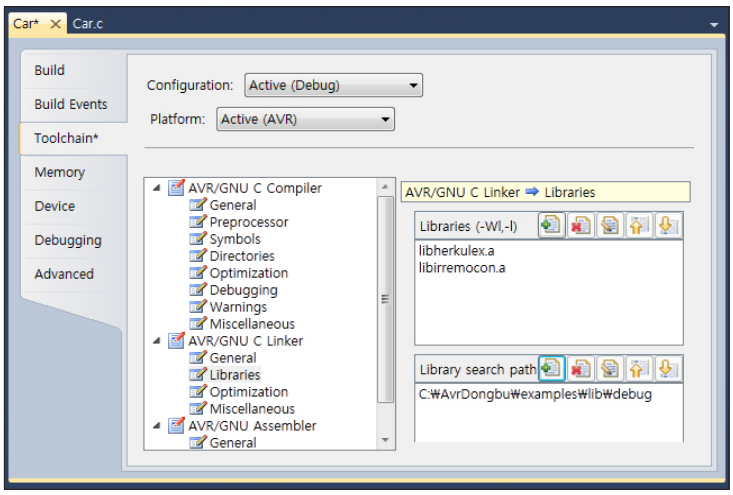
12 Debug

From the supplied library forlder, se-lect debug below the lib.



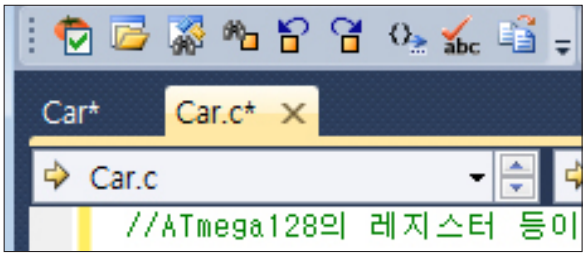
13 Path Complete

Deselect Relative Path.
Library search path setup complete.
Click OK button.



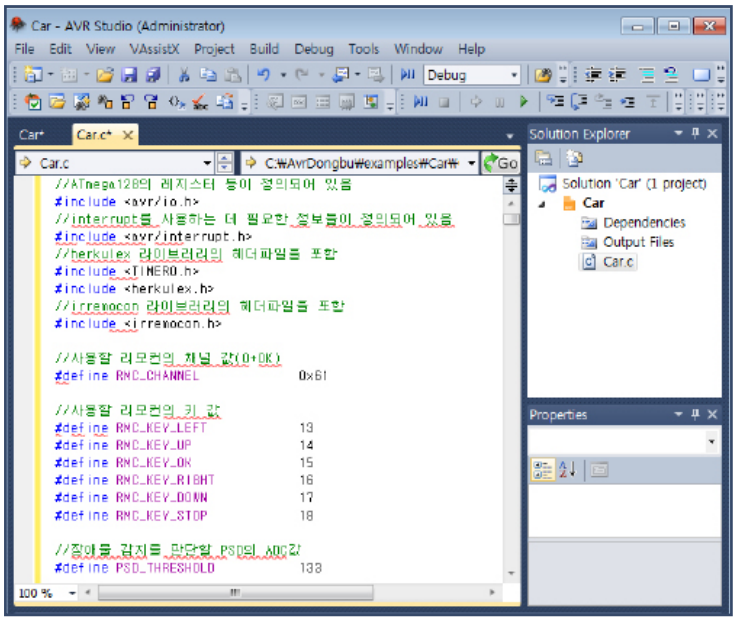
14 Link Complete

C Linker library file and path setup complete.



15 Save

Press save button to save current setup values. Tab bar names changes from Car * -> Car . .



16 코딩

Car.c 탭바를 선택하고, 아래 제공된 예제를 입력합니다.

Example

In this example, we will use the IR remote control to control the dump truck. Up key is forward, Down key reverse, L/R keys are L/R turn. OK button press will raise the dump box and then lower it to the original position. Stop button will stop the truck during movement. When the truck detects an obstacle using front PSD sensor during movement, it will turn right from the same position and move forward again.

Car.c

```
//ATmega128 register defined
#include <avr/io.h>
//nformtion required to use interrupt defined
#include <avr/interrupt.h>
//Include herkulex  library header file
#include <TIMER0.h>
#include <herkulex.h>
// Include irremocon library header file
#include <irremocon.h>

//Remote contral channel value(0+OK)
#define RMC_CHANNEL                    0x61

//Remote control key value
#define RMC_KEY_LEFT                   13
#define RMC_KEY_UP                     14
#define RMC_KEY_OK                     15
#define RMC_KEY_RIGHT                  16
#define RMC_KEY_DOWN                   17
#define RMC_KEY_STOP                   18

//PSD ADC vaue for making obstacle detected judgement
#define PSD_THRESHOLD                  133

//Mask expressing reverse turn bit in I_JOG packet
#define INVERSE_TURN_MASK              0x4000

/**
* Timer0 global variable,  delay function using gucTimerTick.
* Used since gucTimerTick cannot have value greater than
* 255 since it is unsigned char.
*
* @author       Dongbu Robot
* @param        iDelay: Delay time(ms)
* @return       None
* @version      1.0.0.0
*/
```

```
void delay_ms(int iDelay){
	//Repeat if iDelay is > 0
	while(iDelay>0){
		//When iDelay is > 500ms
		if(iDelay>500){
			//Input 250 in gucTimerTick and deduct 500 from iDelay.
			gucTimerTick=250;
			iDelay -= 500;
			//Wait until gucTimerTick becomes 0 due to Timer0
			while(gucTimerTick>0);
		}
		//When iDelay < 500ms
		else{
			//Input iDelay/2 into gucTimerTick and set  iDelay to 0.
			gucTimerTick=iDelay/2;
			iDelay = 0;
			//Wait until gucTimerTick becomes 0 due to Timer0
			while(gucTimerTick>0);
		}
	}
}

/**
* Fuction for stopping dump truc.
* Compose I_JOG packet containing following instructions and send to servo.
* Set wheel servo(0, 2, 4, 5) speed to 0
* Set steering servo(1, 3) direction to face front
* Set dump box servo(6) to original position

*
* @author		Dongbu Robot
* @param		None
* @return		None
* @version		1.0.0.0
*/
void stop(void){
	//Declare packet structure of the packet to be sent
	DrsPacket stSendPacket;
	//Declare variable for use in for statement
	unsigned char i;
	//I_JOG packet size, ID, CMD setup
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 7 * CMD_I_JOG_STRUCT_SIZE;
	stSendPacket.ucChipID = BROADCAST_ID;
	stSendPacket.ucCmd = CMD_I_JOG;
```

MotionPlay.c

```
        //Setup common parameters for 7 servos + ucId setup
        for(i=0;i<7;i++){
                stSendPacket.unData.stIJogData.stIJog[i].stJog.reserved = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.ucStopFlag = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedGreen = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedBlue = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedRed = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.ucJogInvalid = 0;
                stSendPacket.unData.stIJogData.stIJog[i].stSet.reserved = 0;
                stSendPacket.unData.stIJogData.stIJog[i].ucId = i;
        }

        // Servo 0 (Front left wheel) stop (Velocity control)
        stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 20;

        //Servo 1(Left steering) forward direction (Direction control)
        stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 789;
        stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 150;

        //Servo 2(Right front whee ) stop (Velocity control)
        stSendPacket.unData.stIJogData.stIJog[2].stJog.uiValue = 0;
        stSendPacket.unData.stIJogData.stIJog[2].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[2].ucPlayTime = 20;

        //Servo 3(Left steering  ) forward direction (Direction control)
        stSendPacket.unData.stIJogData.stIJog[3].stJog.uiValue = 235;
        stSendPacket.unData.stIJogData.stIJog[3].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[3].ucPlayTime = 150;

        //Servo 4(Left rear wheel)stop (Velcity control)
        stSendPacket.unData.stIJogData.stIJog[4].stJog.uiValue = 0;
        stSendPacket.unData.stIJogData.stIJog[4].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[4].ucPlayTime = 20;

        //Servo 5(Right rear wheel) stop (Velocity control)
        stSendPacket.unData.stIJogData.stIJog[5].stJog.uiValue = 0;
        stSendPacket.unData.stIJogData.stIJog[5].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[5].ucPlayTime = 20;

        //Servo 6(Dump box) original position (Direction control)
        stSendPacket.unData.stIJogData.stIJog[6].stJog.uiValue = 970;
        stSendPacket.unData.stIJogData.stIJog[6].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[6].ucPlayTime = 100;

        //Send packet
        hklx_SendPacket(stSendPacket);
}
```

```
/**
* Compose and send I_JOG packet to send dump box servo (6)
* to original position .
*
* @author       Dongbu Robot
* @param        None
* @return       None
* @version      1.0.0.0
*/
void load_on(void){
        //Declare packet structure of the packet to be send
        DrsPacket stSendPacket;

        //I_JOG packet size, ID, CMD setup
        stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_SIZE;
        stSendPacket.ucChipID = BROADCAST_ID;
        stSendPacket.ucCmd = CMD_I_JOG;

        //Setup paramter to send servo 6(dump box) to original position using position control
        stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 970;
        stSendPacket.unData.stIJogData.stIJog[0].stJog.reserved = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
        stSendPacket.unData.stIJogData.stIJog[0].stSet.reserved = 0;
        stSendPacket.unData.stIJogData.stIJog[0].ucId = 6;
        stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 100;

        //Send packet
        hklx_SendPacket(stSendPacket);
}

/**
* Compose and send I_JOG packet to send
* servo 6 (dump box) to up position.
*
* @author       Dongbu Robot
* @param        None
* @return       None
* @version      1.0.0.0
*/
void load_off(void){
        //Declare packet structure of the packet to be sent
        DrsPacket stSendPacket;

        //I_JOG packet side, ID, CMD setup
        stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_SIZE;
        stSendPacket.ucChipID = BROADCAST_ID;
        stSendPacket.ucCmd = CMD_I_JOG;
```

```
	//Setup paratmeters to send servo 6(Dump box) to up position
	stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 700;
	stSendPacket.unData.stIJogData.stIJog[0].stJog.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].ucId = 6;
	stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 100;

	//Send packet
	hklx_SendPacket(stSendPacket);
}

/**
* Compose and send I_JOG packet to turn Steering servo (1, 3)
* to left turn position.
*
* @author	Dongbu Robot
* @param	None
* @return	None
* @version	1.0.0.0
*/
void left_turn(void){
	//Declare packet structure of the packet to be sent
	DrsPacket stSendPacket;

	//I_JOG packet size, ID, CMD setup
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2 * CMD_I_JOG_STRUCT_SIZE;
	stSendPacket.ucChipID = BROADCAST_ID;
	stSendPacket.ucCmd = CMD_I_JOG;

	//Setup  parameters to turn servo 1(Left steering) to left turn position
	stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 689;
	stSendPacket.unData.stIJogData.stIJog[0].stJog.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].ucId = 1;
	stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

	//Setup parameters to turn servo 3(Right steering) to right turn position
	stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 135;
	stSendPacket.unData.stIJogData.stIJog[1].stJog.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucStopFlag = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedGreen = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedBlue = 0;
```

```
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedRed = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucJogInvalid = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[1].ucId = 3;
	stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 50;

	//Send packet
	hklx_SendPacket(stSendPacket);
}

/**
* Compose and send I_JOG packet to turn steering servo(1, 3)
* to right turn position
* @author	Dongbu Robot
* @param	None
* @return	None
* @version	1.0.0.0
*/
void right_turn(void){
	//Declare structure of packet to be sent
	DrsPacket stSendPacket;

	//I_JOG packet size, ID, CMD setup
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2 * CMD_I_JOG_STRUCT_SIZE;
	stSendPacket.ucChipID = BROADCAST_ID;
	stSendPacket.ucCmd = CMD_I_JOG;

	//Setup  parameters to turn servo 1(Left steering) to right turn position
	stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 889;
	stSendPacket.unData.stIJogData.stIJog[0].stJog.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[0].ucId = 1;
	stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

	//setup parameters to turn servo 3(Right steering) to right turn position
	stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 335;
	stSendPacket.unData.stIJogData.stIJog[1].stJog.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucStopFlag = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedGreen = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedBlue = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucLedRed = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucJogInvalid = 0;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.reserved = 0;
	stSendPacket.unData.stIJogData.stIJog[1].ucId = 3;
	stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 50;
```

```
	//Send packet
	hklx_SendPacket(stSendPacket);
}

/**
* Fuction to move dump truck forward.
*  Compose I_JOG packet containing following instructions and send to servo.
Set wheel servos(0, 2, 4, 5) to move toward front
* Set steering servos(1, 3) to face forward
*
*
* @author		Dongbu Robot
* @param		None
* @return		None
* @version		1.0.0.0
*/
void forward(void){
	//Setup packet structure for packets to be sent and received
	DrsPacket stSendPacket;
	//Declare variable for use in for statement
	unsigned char i;

	//I_JOG packet size, ID, CMD setup
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 6 * CMD_I_JOG_STRUCT_SIZE;
	stSendPacket.ucChipID = BROADCAST_ID;
	stSendPacket.ucCmd = CMD_I_JOG;

	//Setup common parameters for 6 servos + ucId setup
	for(i=0;i<6;i++){
		stSendPacket.unData.stIJogData.stIJog[i].stJog.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucStopFlag = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedGreen = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedBlue = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedRed = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucJogInvalid = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].ucId = i;
	}

	//Servo 0(Left front wheel) rotate forward direction(Velocity control)
	stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 800;
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 1;
	stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 10;

	//Servo 1(Left steering) forward position (Direction control)
	stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 789;
	stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 150;

	//Servo 2(Right front wheel) rotate forward direction (Velocity control)
	stSendPacket.unData.stIJogData.stIJog[2].stJog.uiValue = (800 | INVERSE_TURN_MASK);
	stSendPacket.unData.stIJogData.stIJog[2].stSet.ucMode = 1;
	stSendPacket.unData.stIJogData.stIJog[2].ucPlayTime = 10;

	//Servo 3 (Right steering) forward position (Direction control)
	stSendPacket.unData.stIJogData.stIJog[3].stJog.uiValue = 235;
	stSendPacket.unData.stIJogData.stIJog[3].stSet.ucMode = 0;
	stSendPacket.unData.stIJogData.stIJog[3].ucPlayTime = 150;
```

```
	//Servo 4(Left rear wheel) rotate forward direction(Velocity control)
	stSendPacket.unData.stIJogData.stIJog[4].stJog.uiValue = 800;
	stSendPacket.unData.stIJogData.stIJog[4].stSet.ucMode = 1;
	stSendPacket.unData.stIJogData.stIJog[4].ucPlayTime = 10;

	//Servo 5(right rear wheel) rotate forward direction (velocity control)
	stSendPacket.unData.stIJogData.stIJog[5].stJog.uiValue = (800 | INVERSE_TURN_MASK);
	stSendPacket.unData.stIJogData.stIJog[5].stSet.ucMode = 1;
	stSendPacket.unData.stIJogData.stIJog[5].ucPlayTime = 10;

	//Send packet
	hklx_SendPacket(stSendPacket);
}

/**Fuction to move dump truck reverse.
*  Compose I_JOG packet containing following instructions and send to servo
* Setup servos(0, 2, 4, 5) to rotate in revers direction
* Setup steering servos(1, 3) to face forwrard direction

*
* @author		Dongbu Robot
* @param		None
* @return		NOne
* @version		1.0.0.0
*/
void backward(void){
	//Declare structure for packet to be sent
	DrsPacket stSendPacket;
	//Declare variable for use in for statement
	unsigned char i;

	//I_JOG packet size, ID, CMD setup
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 6 * CMD_I_JOG_STRUCT_SIZE;
	stSendPacket.ucChipID = BROADCAST_ID;
	stSendPacket.ucCmd = CMD_I_JOG;

	//Setup common parameters for 6 servos + ucId setup
	for(i=0;i<6;i++){
		stSendPacket.unData.stIJogData.stIJog[i].stJog.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucStopFlag = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedGreen = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedBlue = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedRed = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucJogInvalid = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].ucId = i;
	}

	//Servo 0(Left front wheel) rotate in reverse direction (Velocity control)
	stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = (800 | INVERSE_TURN_MASK);
	stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 1;
	stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 10;
```

```
        //Servo 1(Left steering) forward direction (Direction control)
        stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 789;
        stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 150;

        //Servo 2(Right front wheel) rotate in reverse direction (Speed control)
        stSendPacket.unData.stIJogData.stIJog[2].stJog.uiValue = 800;
        stSendPacket.unData.stIJogData.stIJog[2].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[2].ucPlayTime = 10;

        //Servo 3(Right steering) forward direction (Direction control)
        stSendPacket.unData.stIJogData.stIJog[3].stJog.uiValue = 235;
        stSendPacket.unData.stIJogData.stIJog[3].stSet.ucMode = 0;
        stSendPacket.unData.stIJogData.stIJog[3].ucPlayTime = 150;

        //Servo 4(Left rear wheel) rotage in reverse direction (Velocity control)
        stSendPacket.unData.stIJogData.stIJog[4].stJog.uiValue = (800 | INVERSE_TURN_MASK);
        stSendPacket.unData.stIJogData.stIJog[4].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[4].ucPlayTime = 10;

        //Servo 5 (Right rear wheel) rotae in reverse direction (Velocity control)
        stSendPacket.unData.stIJogData.stIJog[5].stJog.uiValue = 800;
        stSendPacket.unData.stIJogData.stIJog[5].stSet.ucMode = 1;
        stSendPacket.unData.stIJogData.stIJog[5].ucPlayTime = 10;

        //Send packet
        hklx_SendPacket(stSendPacket);
}

/**Function to turn the truck rapidly from the same postion when obstacle encountered.
* Compose I_JOG packet containing following instructions and send to servo
* From servos (0, 2, 4, 5) left wheel rotate forward,
* Right wheel rotate in reverse to turn truck to the right
*
*
* @author       Dongbu Robot
* @param        None
* @return       None
* @version      1.0.0.0
*/
void quick_turn(void){
        //Delcare structure of the packet to be sent
        DrsPacket stSendPacket;
        //Declare variable for use in for
        unsigned char i;

        //I_JOG packet size, ID, CMD setup
        stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 4 * CMD_I_JOG_STRUCT_SIZE;
        stSendPacket.ucChipID = BROADCAST_ID;
        stSendPacket.ucCmd = CMD_I_JOG;
```

```
	//4개 서보에 공통적인 파라미터 설정(0, 3번은 전진 방향, 2, 5번은 후진 방향)
	for(i=0;i<4;i++){
		stSendPacket.unData.stIJogData.stIJog[i].stJog.uiValue = 900;
		stSendPacket.unData.stIJogData.stIJog[i].stJog.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucStopFlag = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucMode = 1;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedGreen = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedBlue = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedRed = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.ucJogInvalid = 0;
		stSendPacket.unData.stIJogData.stIJog[i].stSet.reserved = 0;
		stSendPacket.unData.stIJogData.stIJog[i].ucPlayTime = 10;
	}

	//ucId setup
	stSendPacket.unData.stIJogData.stIJog[0].ucId = 0;
	stSendPacket.unData.stIJogData.stIJog[1].ucId = 2;
	stSendPacket.unData.stIJogData.stIJog[2].ucId = 4;
	stSendPacket.unData.stIJogData.stIJog[3].ucId = 5;

	//Send packet
	hklx_SendPacket(stSendPacket);
}

int main(void)
{
	//Declare packet structure for packet to be sent and received
	DrsPacket stSendPacket;
	//Declare 8bit variable
	unsigned char ucData, ucDataOld, ucLength, ucForward;
	//Declare 16bit variable
	int iADC;

	//Setup Port C I/O direction
	DDRC = 0b01111111;
	//Setup Port C output value
	PORTC = 0b01111111;

	//VREF=AREF Input voltage, Left alignment off, ADC 6 single pole input
	ADMUX=0b00000110;
	//ADC activate, ADIF initialize, Clock divider 128
	ADCSRA=0b10010111;

	//Deactivate all interrupts
	cli();

	//Initialize to use HerkuleX
	hklx_Init(115200);

	//Initialize to use remote control receiver module
	RMC_Init();

	//Activate all interrupts
	sei();
```

```
//Write 0x60 to Torque Control. compose Torque ON packet
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
stSendPacket.ucChipID = 254;
stSendPacket.ucCmd = CMD_RAM_WRITE;
stSendPacket.unData.stRWData.ucAddress = 52;
stSendPacket.unData.stRWData.ucLen = 1;
stSendPacket.unData.stRWData.ucData[0] = 0x60;

//Send Packet
hklx_SendPacket(stSendPacket);

//Initialize ucDataOld to default value
ucDataOld = RMC_ERROR_DATA;

//Initialize ucForward to 0
ucForward = 0;

//Continuously repeat content of while statement
while(1)
{
        //Start ADC conversion
        ADCSRA |= (1<<ADSC);
        //Wait until ADC conversion ends
        while(!(ADCSRA & (1<<ADIF)));
        //Read ADCL & ADCH value and save
        iADC = ADCL | (ADCH<<8);
        //Initialize ADIF
        ADCSRA |= (1<<ADIF);

        // When obstacle encountered during forward movement (0.65/5*1024 = 133.12)
        if(iADC > PSD_THRESHOLD && ucForward == 1){
                //Stop turn for 1s and move forward
                stop();
                delay_ms(500);
                quick_turn();
                delay_ms(1000);
                forward();
                ucForward = 1;
        }

        //Use ucData, ucLength to read remote control value
        RMC_ReadRemocon(RMC_CHANNEL, &ucData, &ucLength);

        //If remote control value is different from previous value
        if(ucDataOld != ucData){
                //Move forward when UP key pressed
                if(ucData == RMC_KEY_UP){
                        forward();
                        ucForward = 1;
                }
                //Reverse when DOWN down key pressed
                else if(ucData == RMC_KEY_DOWN){
                        backward();
                        ucForward = 0;
```

```
                        }
                        //Turn lef when LEFT key pressed
                        else if(ucData == RMC_KEY_LEFT){
                                left_turn();
                        }
                        //Turn right when RIGHT key pressed
                        else if(ucData == RMC_KEY_RIGHT){
                                right_turn();
                        }
                        //Lift dump box and return to original position when OK key pressed
                        else if(ucData == RMC_KEY_OK){
                                load_off();
                                delay_ms(1000);
                                load_on();
                        }
                        //Stop when STOP key pressed
                        else if(ucData == RMC_KEY_STOP){
                                stop();
                                ucForward = 0;
                        }
                }

                //Save current value in another variable
                ucDataOld = ucData;
        }

        //Convert integer value to fit function format
        return 1;
}
```

Example Description

Refer to Chapter 3 Controlling HekuleX for explanation about TIMER0.h and herkulex which are related to libherkulex.a .lib–herkulex .a is one of the header files which is included from Car.c using #include statement. Refer to Chapter 12 (Reading remote control singal) for information concerning libirremocon.a and related irremocon.h.

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

DDRX, PORTX register, and other registers required for AVR control are defined in the avr/io.h. avr/interrupt.h header file contains functions for using interrupt.

```
#include <herkulex.h>
#include <TIMER0.h>
#include <irremocon.h>
```

herkulex.h header file must be always inlcuded. Header includes declarations and structures for creating and sending packets, and defined functions. TIMER0.h is also included to set Delay using gucTimerTick. irremocon.h must also be included since it contains functions for initializing remote control receiver module and reading remote control data..

```
#define RMC_CHANNEL                    0x61
```

Depending on the remote control setut, channel value can range from 0x61 to 0x6A. In this example channel value will be defult value 0x61 which is set by using pressing 0 button + OK button.

```
#define RMC_KEY_LEFT                   13
                        …
#define RMC_KEY_STOP                   18
```

Declare remote control key and applicable key value. Up/Down/L/R. OK, and red Stop keys will be used. Once the key and the data values are declared there will be no need to look up the value every time it is needed.

```
#define PSD_THRESHOLD                  133
```

Standard value which is used as comaprison against read PSD to judge whether obstacle is located at front. According to information from H/W fundamentals and PSD outpt graph in Chapter 6, PSD voltage is approximately 0.65V at distance of 20cm. 0.65V when coverted to ADC value 0.65/5*1024=133.12. 133 was set as standard 133. If an obstacle is within 20cm distance, output voltage will be greater than 0.65V. ADC value greater than 133 means there is an obstacle within 20cm distance.

```
#define INVERSE_TURN_MASK              0x4000
```

There is forward turn and reverse turn when using I_JOG and S_JOG to control ther servo velocity. In reverse turn, 14th bit of uiValue which is a structure of stJog has to be set to 1. Therefore, mask is declared as shown above and Bitwise OR operation is performed on the mask and speed value and substitued in uiValue.

```
void delay_ms(int iDelay){
        while(iDelay>0){
                if(iDelay>500){
                        gucTimerTick=250;
                        iDelay -= 500;
                        while(gucTimerTick>0);
                }
                else{
                        gucTimerTick=iDelay/2;
                        iDelay = 0;
                        while(gucTimerTick>0);
                }
        }
}
```

Delay funtion usingTIMER0. Has iDelayTick as mediating variable and expresses delay time in (ms)unit. gucTimerTick is a global variable declared in TIMER0.h, normally maintained as 0, when non 0 value is substituted, value decreases by 1 every 2ms due to TIMER) overflow service routine defined in libherkulex.a. For example, to wait 10ms, input 5 in gucTimerTick and wait until gucTimerTick becomes 0. This is how delay_ms function is operates. gucTimerTick is a unsigned char, in other words, 1byte variable without sign. Therefore it can only have values between 0~255. 255 in gucTimerTick is equal to 510ms. gucTimerTick can only operate 0.5 ms with single substitution. Therefore, function cuts delay value iDelay and substitutes gucTimerTick in500ms segments. If iDelay is larger than 500, 250 is sustituted in gucTimerTick and iDelay is changed to value with 500 deducted and using while(gucTimerTick〉0); delay is repeated continuously as long as gucTimerTick is 〉 0 and exits while statement when gucTimerTick becomes 0.When iDelay is 500 or smaller, iDelay/2 is substituted in gucTimerTick and sets iDelay value to 0 and waits until the gucTimerTick becomes 0 using the while statement.This process is surrounded by while(iDelay〉0) and repeats as long as iDelay is 〉 0. While statement ends when iDelay becomes0 and delay_ms function ends.

```
void stop(void){
…
}
```

Next function is stop() function . This function sends I_JOG packet to HerkuleX to stop the dump truck.

```
DrsPacket stSendPacket;
unsigned char i;
```

Declare variable. stSendPacket is a location where I_JOG to be sent will be saved. i is a declared variable whic is used to repeat the content of the for statement by same number of times as number of servo motors to be controlled.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 7 * CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
```

Section for setting up packet size, ID, and CMD. Following I_JOG packet structure, packet size is standard 7 byte (MIN_PACKET_SIZE)+ 5byte/motor * number of motors. This function will control 7 motors with ID 0~6 and the packet size will be 42.Since it's an I_JOG , ID is BROADCAST_ID(0xFE) and CMD is CMD_I_JOG(0x05).

```
for(i=0;i<7;i++){
        stSendPacket.unData.stIJogData.stIJog[i].stJog.reserved = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucStopFlag = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedGreen = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedBlue = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucLedRed = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucJogInvalid = 0;
        stSendPacket.unData.stIJogData.stIJog[i].stSet.reserved = 0;
        stSendPacket.unData.stIJogData.stIJog[i].ucId = i;
}
```

As Jog command has many parameters, codes can become very long if parameters are set for each individual motor. Therefore, for statement is used to setup the parameters with same value all at once. Also, ucId is set to i to make Motor ID same as the stJog array index.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 0;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 1;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 20;
                                ...
stSendPacket.unData.stIJogData.stIJog[6].stJog.uiValue = 970;
stSendPacket.unData.stIJogData.stIJog[6].stSet.ucMode = 0;
stSendPacket.unData.stIJogData.stIJog[6].ucPlayTime = 100;
```

Setup control mode, speed/position value, and run time for each motor from 0–6. Set wheel servos 0, 2, 4, 5 to speed control mode and speed value to 0. Set steering servos1, 3 to direction or position control mode and position to 789 and 235 to face forward direction. Set dump box servo 6 to direction or position control mode and position to 970 which is the default position.

```
hkIx_SendPacket(stSendPacket);
```

Use hkIx_SendPacket() function to send the packet composed above to HerkuleX.

```
void load_on(void){
...
}
```

Next function is load_on() . This function sends dump box (servo 6) to original position.

```
DrsPacket stSendPacket;

stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
```

Declare stSendPacket structure for saving packet to be sent. Setup packet size, ID, and CMD. Only servo ID 6 will be controlled. Packet size is standard 7byte (MIN_PACKET_SIZE) + 5bye =12.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 970;
                                    …
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
                                    …
stSendPacket.unData.stIJogData.stIJog[0].ucId = 6;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 100;

hklx_SendPacket(stSendPacket);
```

seteup following stJog uiValue to 970 (default position), stSet ucMode to 0(position control), ucId to 6, runtime to 100(approximately 1.12s) and send packet. Dump box servo will move to default positon.

```
void load_off(void){
…
}
```

Next function is load_off. This function moves dump box (servo 6) to unload position. Content of this function is similar to load_on.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 700;
```

Difference between load_off function and load_on . Only difference is the servo 6 position. When servo is sent to positio 700, dum box will be lifted to unload position.

```
void left_turn(void){
…
}
```

Function to move steering servos (1, 3) to turn the wheels towards left turn position.

```
DrsPacket stSendPacket;

stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2 * CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
```

Declare stSendPacket structure for saving packets to be send and setup packet size, ID, and CMD. Servos 1 & 3 will be controlled.Packet size is standard 7byte (MIN_PACKET_SIZE)+ 5*2byte = 17.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 689;
…
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
…
stSendPacket.unData.stIJogData.stIJog[0].ucId = 1;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 135;
…
stSendPacket.unData.stIJogData.stIJog[1].stSet.ucMode = 0;
…
stSendPacket.unData.stIJogData.stIJog[1].ucId = 3;
stSendPacket.unData.stIJogData.stIJog[1].ucPlayTime = 50;

hklx_SendPacket(stSendPacket);
```

Setup parameters to turn servo 1 and 3 to left turn position. Setup stJog uiValue value to 689, 135. stSet ucMode to 0(position control), ucId to 1 & 3, runtime to 50(approximately 0.56s). Send packet.

```
void right_turn(void){
…
}
```

Function to move steering servos(1, 3) to right turn position. Content of this function is similar to left_turn() function.

```
SendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 889;
                                …
stSendPacket.unData.stIJogData.stIJog[1].stJog.uiValue = 335;
```

Difference between the right_turn() function & left_turn() function. The only differenc is the position of the servos 1 & 3. Set the sevo position to 889, 335.

```
void forward(void){
…
}
```

This function makes the truch move forward. Send I_JOG pcaket, control servo speed for wheel servos (0, 2, 4, 5) to forward direction. Control servo position for steering servos(1, 3) to face forward.

```
DrsPacket stSendPacket;
unsigned char i;
```

Declare variable. stSendPacket is location where I_JOG packet will be saved. i is a declared variable which is used to repeat the content of the for statement by same number of times as number of servo motors to be controlled.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 6 * CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
```

Section for setting up packet size, ID, and CMD. Following I_JOG packet structure, packet size is standard 7 byte (MIN_PACKET_SIZE)+ 5byte/motor * number of motors. This function will control 6 motors with ID 0~5 and the packet size will be 37.

```
for(i=0;i<6;i++){
…
}
```

For statement is used to setup the parameters with same value all at once. Also, ucId is set to i to make Motor ID same as the stJog array index.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = 800;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 1;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 10;

stSendPacket.unData.stIJogData.stIJog[5].stJog.uiValue = (800 | INVERSE_TURN_MASK);
stSendPacket.unData.stIJogData.stIJog[5].stSet.ucMode = 1;
stSendPacket.unData.stIJogData.stIJog[5].ucPlayTime = 10;
```

Setup control mode, speed/position value, and run time for each motor from 0–5. Set wheel servos 0, 2, 4, 5 to speed control mode. Speed values for left servos 0,4 are forward 800, values for right servos 2,5 are reverse 800. These values will make all ther wheels to rotate forward direction. When setting up servo 2, 5 stJog.uiValue, reason for using result of Bitwise OR operation on 800 and INVERSE_TURN_MASK is to show reverse rotation. Steering servos 1,3 will be positioned at 789, 235 to face forward direction.

```
hklx_SendPacket(stSendPacket);
```

Declare Dhklx_SendPacket() function to send the composed packet above to HerkuleX.

```
void backward(void){
…
}
```

This function is used to reverse the truck. Send I_JOG pcaket, control servo speed for wheel servos (0, 2, 4, 5) to reverse direction. Control servo position for steering servos(1, 3) to face forward. This function is similar to forward() function.

```
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = (800 | INVERSE_TURN_MASK);
                                        …
stSendPacket.unData.stIJogData.stIJog[2].stJog.uiValue = 800;
                                        …
stSendPacket.unData.stIJogData.stIJog[4].stJog.uiValue = (800 | INVERSE_TURN_MASK);
                                        …
stSendPacket.unData.stIJogData.stIJog[5].stJog.uiValue = 800;
```

The difference between the backward() function and forward() function. Set forward/reverse to opposite of forward movement so that all wheels will rotate in reverse direction.

```
void guide turn(void){
…
}
```

quick_turn() function is used when obstacle in encountered.This packe will make the left wheel turn forward and the right wheel turn in reverse to turn the truck to the left from the same position.

```
DrsPacket stSendPacket;
unsigned char i;

stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 4 * CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
```

Declare stSendPacket for saving packet to be sent. Declare i variable for use in for statement, Setup packet size, ID, CMD. Servos 0, 2, 4, 5 will be controlled. Packet size is standard 7byte(MIN_PACKET_SIZE)+ 5*4byte = 27.

```
for(i=0;i<4;i++){
        stSendPacket.unData.stIJogData.stIJog[i].stJog.uiValue = 900;
                                ...
        stSendPacket.unData.stIJogData.stIJog[i].stSet.ucMode = 1;
                                ...
        stSendPacket.unData.stIJogData.stIJog[i].ucPlayTime = 10;
}
```

Use for statement to setup parameters with same values all at once. In this function, For statement will setup the values all at once since all servos have the same control mode, speed value, and runtime. Except, declaring ucId as i will be deleted since servo IDs are not all in sequrence.

```
stSendPacket.unData.stIJogData.stIJog[0].ucId = 0;
stSendPacket.unData.stIJogData.stIJog[1].ucId = 2;
stSendPacket.unData.stIJogData.stIJog[2].ucId = 4;
stSendPacket.unData.stIJogData.stIJog[3].ucId = 5;

hkIx_SendPacket(stSendPacket);
```

Input subject servo ID in ucId and send packet. Dump truck will turn to the left when this packet is sent..

Description about main function.

```
DrsPacket stSendPacket;
unsigned char ucData, ucDataOld, ucLength, ucForward;
int iADC;
```

Declare varialble to be used in main function. Declare variable to be used when packet will be sent as DrsPacket structure type.ucData is current remote control value read from while loop. ucDataOld is previous remote control value read from the while loop. At the end of the whil loop, ucDataOld is updated with ucData value. These two variables are used to chage the timer/counte2 value when remote control value changes due to button press or release. variable ucLength is used to read the length of the remote control button presse. Variable ucForward is used to show that robot is currently moving forward. 1 means robot is moving forward. iADC is a 16bit integer variable for saving ADC value.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

Setup Port C pins 0~6 to output and output value to 1.All LEDs will be turned off.

```
ADMUX=0b00000110;
ADCSRA=0b10010111;
```

Register setup to allow ADC to read analog input from ADC6 in single pole mode. Looking at ADMUX first., DRC circuit standard voltage and ATmega128 operational voltage 5V are connected to AREF pin together. REFS1:0 bit is set to 0b00, ADLAR is set to 0 since left alignment will not be used. Set MUX4:0 bit to 0b00110 since ADC6 port will read with single pole mode. In ADCSRA, set ADEN bit t 1 to activate ADC first, and since coversion will not begin yet and free running will not be used, set ADSC and ADFR to 0. In case ADIF is set to 1, write 1 to initialize to 0, set ADIE to 0 since interrupt will not be sued. Last, to divide ATmega128의 16MHz clock frequency to 50~200kHz suitable for ADC, set ADSP2:0 bit to 0b111 and divide by 128 to supply 125kHz to ADC.

Refer to Chapter 6 for more detailed information on ADC register.

```
cli();
hklx_Init(115200);
RMC_Init();
sei();
```

To prevent error casued by an interrupt when interrupt setup is being changed, use Cli() command to deactivate all interrupts. Next use hklx_Init() function to intialize timer/counter 0 and USART0 which are used for HerkuleX control. use baud rate 1152000 as parameter. Use RMC_Init() function to initalize external interrupt #4 pin and timer/counter 1 which are used for receiving remote control signal.After initialization, use sei() command to activate all interrupts.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_RAM_WRITE;
stSendPacket.unData.stRWData.ucAddress = 52;
stSendPacket.unData.stRWData.ucLen = 1;
stSendPacket.unData.stRWData.ucData[0] = 0x60;

hklx_SendPacket(stSendPacket);
```

Motor torque has to be enabled for motor to move on it's own. above codes are for constructing and sending RAM_WRITE packet to enable torque. Packet size is standard MIN_PACKET_SIZE(7byte)+address+length+data(1 byte). ID is BROADCAST_ID (254) since packet has to be sent to all servos.. CMD is CMD_RAM_WRITE(0x03), Set ucAddress, ucLen, ucData[0] values to 52, 1, 0x60. Next, call hklx_SendPacket() function to send the packet.

```
ucDataOld = RMC_ERROR_DATA;
ucForward = 0;
```

As explained previously, ucDataOld variable contains remote control value read from the while statement previously. But since while statement has not been entered yet and remote control value has not been read, save default value RMC_ERROR_DATA in ucDataOld. Also, since truck is at stop, Initialize ucForward variable which shows whether the truck is moving to 0.

```
ADCSRA |= (1<<ADSC);
while(!(ADCSRA & (1<<ADIF)));
iADC = ADCL | (ADCH<<8);
ADCSRA |= (1<<ADIF);
```

First section of the while statement continuously repeat ADC conversion.Fist, ADCSRA ADSC bit is set to 1.and then ADC coversion will start. Using while(!(ADCSRA & (1<<ADIF))); command. program will wait as long as ADCSRA ADIF is 0.When ADC conversion ends and ADIF bit becomes 1, program will exit while statement and proceed to next step. After ADC conversion is complete and ADCL and ADCH is updated, ADCL and ADCH values are read and saved in iADC. ADCH belongs to top 1byte of ADC value. To save 10bit date in iADC, perform 8bit (1byte)to the left <<(shift left) operation and |(bitwise OR) operation on ADCL and save in iADC.

Since ADC conversion complete interrupt is not used, ADIF bit is not set to 0 automatically. Therefore, Interrupt flag has to be initalized back to 0 after ADC value is read. Write 1 to ADCSRA ADIF bit to initialize this bit to 0.

```
if(iADC > PSD_THRESHOLD && ucForward == 1){
        stop();
        delay_ms(500);
        quick_turn();
        delay_ms(1000);
        forward();
        ucForward = 1;
}
```

If iADC value is greater than PSD_THRESHOLD and obstacle is determined to exist within 20cm, if statement will be entered. stop() functions is called to stop and delay for 0.5s. Next, quick_turn() function is called to send right turn package and wait 1s for right turn to complete. Next, forward() function is called to move forward. Netxt, ucForward is set to 1 to show truck is moving forward.

```
RMC_ReadRemocon(RMC_CHANNEL, &ucData, &ucLength);
```

Use RMC_ReadRemocon() function to read remote control data. Input desired remote control channel RMC_CHANNEL in the first mediating variable. In the 2nd and 3rd mediating variable, enter ucData and ucLength adress to save data and button press length from RMC_CHANNEL .

```
if(ucDataOld != ucData){
…
}
```

Execute if statement if the Remote control value (ucDataOld) read from forward while loop and recently read remote control value (ucData) is different. This is so that command will be sent only when remote control key is pressed.

```
If(ucData == RMC_KEY_UP){
        forward();
        ucForward = 1;
}
else if(ucData == RMC_KEY_DOWN){
        backward();
        ucForward = 0;
}
else if(ucData == RMC_KEY_LEFT){
        left_turn();
}
else if(ucData == RMC_KEY_RIGHT){
        right_turn();
}
else if(ucData == RMC_KEY_OK){
        load_off();
        delay_ms(1000);
        load_on();
}
else if(ucData == RMC_KEY_STOP){
        stop();
        ucForward = 0;
}
```

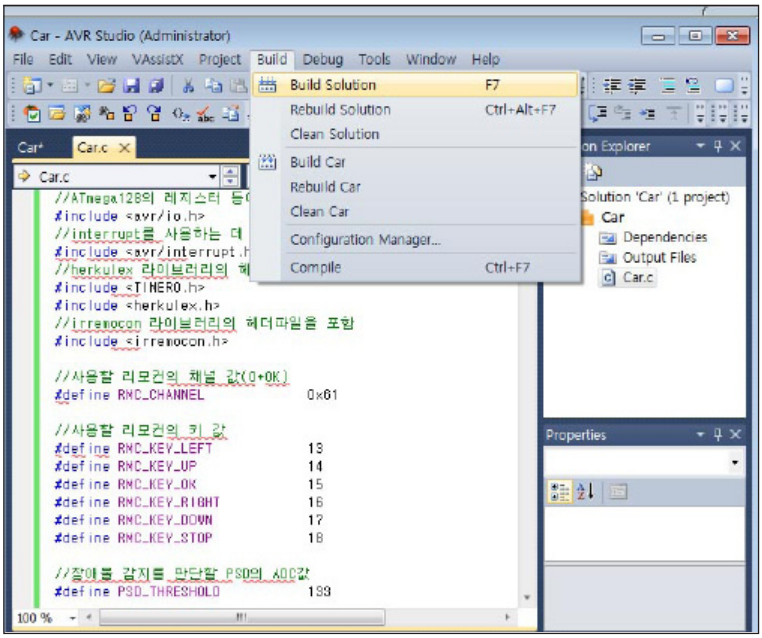
When ucData is equal to RMC_KEY_UP (UP key was presseed), forward function is called to move truck forward and ucForward is changed to 1. Other else if statement perform the same function . When DOWN key is pressed, backward function is called. Left Key presee will call left_turn function. Right key press will call right_turn function. When OK key is pressed, load_off function will be called to lift the dump box and 1s later load_on function will be called to return the dump box to original position. When Stop key is pressed, stop function will be called. During reverse movement and stop, ucForward will be set to 0 so that truck will not react to an obstacle detected by PSD during reverse and stop.

```
ucDataOld = ucData;
```

Everytime while loop ends, current ucData will be entered in ucDataOld to be compared in the next loop.

Run

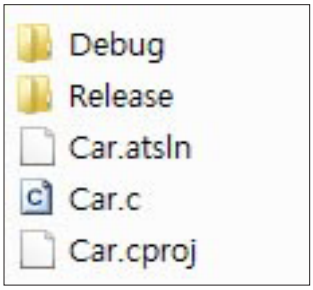
Connect to DRC to dump truck robot, download firmware and run. Use remote control keys to control the truck



01 Build

Select Build menu and then click Build Solution to build.

Successful compilation without error will be shown by
1 succeded 0 failed comment.



02 Debug Folder

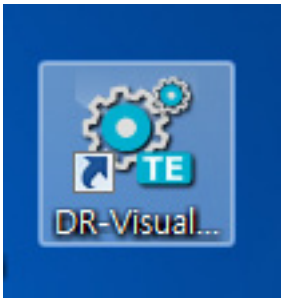
After bulid, debug folder will be created within the soruce folder . Click debug folder to view the folder content



03 hex Check

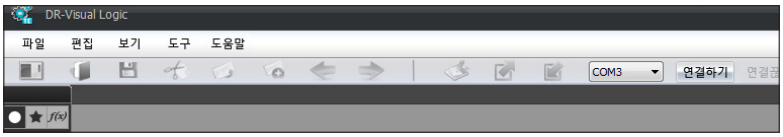
Check the hex file within the Debug folder. Hex file will be ported to the controller.

Start restoring the firmware from DR-Visual Logic.



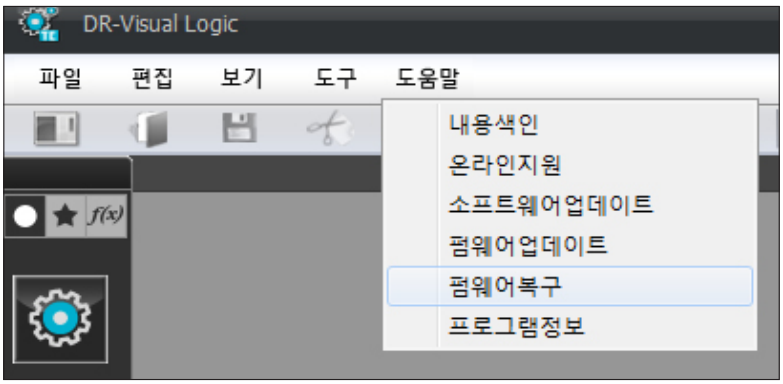
04 Visual Logic

Run DR-Visual Logic



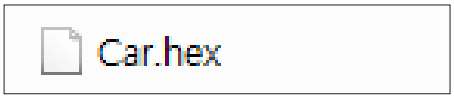
05 Connect

Select COM port and click Connect.



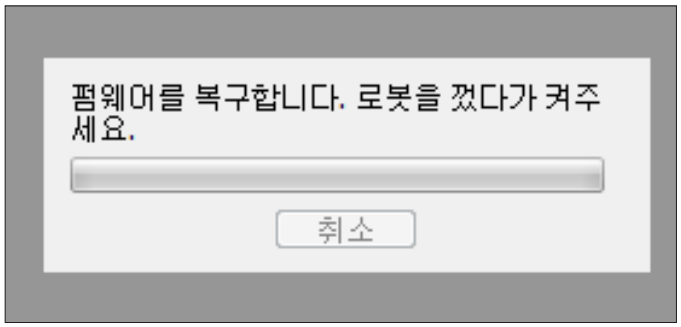
06 Restore Firmware

Click Restore Firmware in the Help Menu.



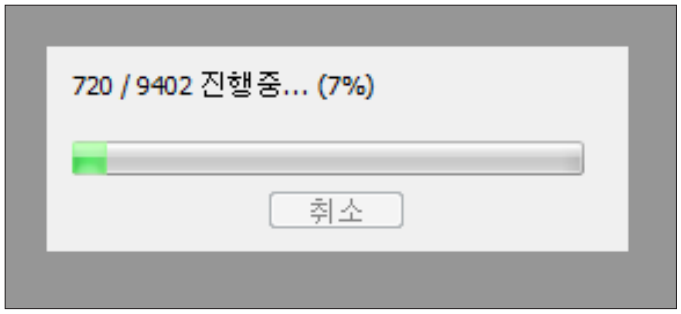
07 Load

Load Hex file from Debug folder



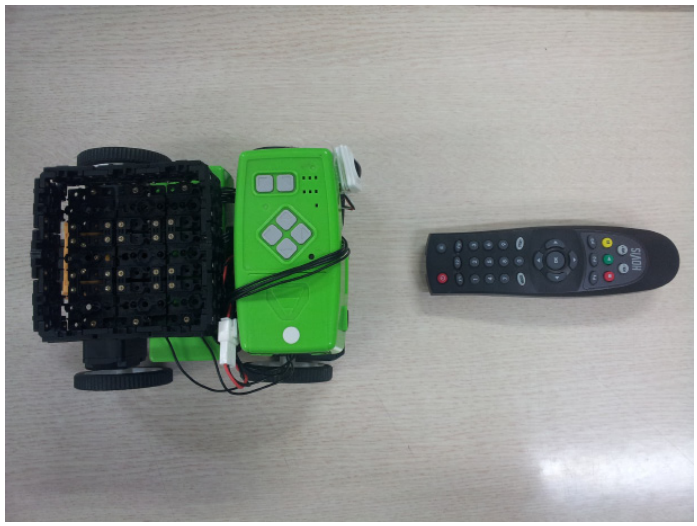
08 Firmware Restoration

Message asking to restart the robot. Turn the controller power button off and back on.



09 Start Restoration

Firmware restoration in progress.



10 Run

Connect to DRC to dump truck robot, download firmware and run. Use remote control keys to control the truck.

Learning With Robot AVR C Programming



ATmega128 Microprocessr Installed Robot Controller AVR C Programming

PART 01. Learning HOVIS DRC and ATmega128

PART 02. Learning Visual Logic related C language grammar

PART 03. Learning AVR robot controller firmware

PART 04. Applying to the robot

Dongbu Robot

Dongbut Robot Co., Ltd www.dongburobot.com

HQ : 11th Fl Bucheon Techno Prk Bldg, 401, Yakdae-dong, Wonmi-gu, Bucheon-si, Gyeonggi-do, S Korea
TEL : 032-329-5551 (내선 103), FAX : 032-329-5569, E-MAIL : robotsales@dongbu.com
Factory : 6 Gil 27, 4 San dan, Jiksan-eup, Sebouk-gu, Cheonan-ski, Chungcheonnam-do, S Korea
TEL : 041-590-1700, FAX : 041-590-1701

93550
9 788998 873097
ISBN 978-89-98873-09-7