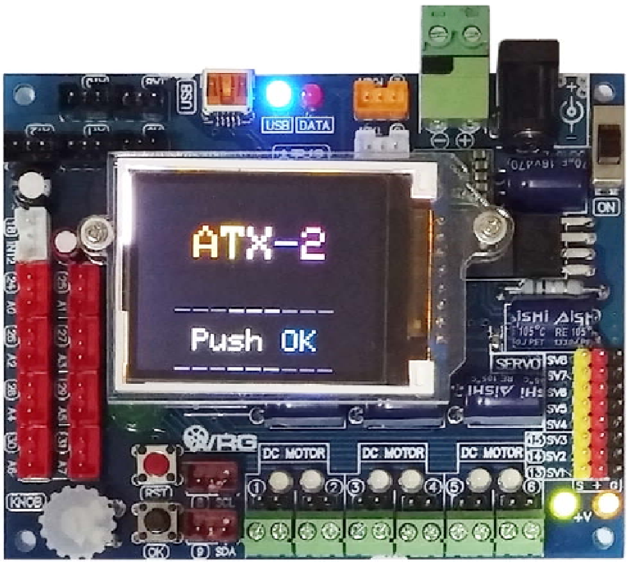
Robo-Creator 2
creative robotics kit



ATX2

The ultimate controller board

Activity book

inex
INNOVATIVE EXPERIMENT

Robo-Creator 2
creative robotics kit

ATX2 : The ultimate controller board activity book

Who should use this handbook?

1. Students and other people who are interested in applying to microcontrollers for testing working process of automatic system or people who are fascinated in learning and examining the microcontrollers in new approaches such as using an autonomous robot as a form of an interactive media.

2. Academic institutes such as schools, colleges and universities, where provide electronic subjects or electronic and computing engineering departments.

3. Lecturers and teachers who would like to study and prepare lesson plans for microcontroller courses, including applied science which focuses on integrating electronics, microcontrollers, computer programming and scientific examination in high school education, vocational education, and bachelor's degree.

Published and distributed by

Innovative Experiment Co.,Ltd.

108 Soi Sukumvit 101/2 Sukumvit Road, Bangna, Bangkok 10260 THAILAND

www.inex.co.th/www.inexglobal.com

Details and illustrations used in this handbook are thoroughly and carefully to provide the most accurate and comprehensive information under the conditions and time we have before publishing. Innovative Experiment Co.,Ltd. shall not be responsible for any damages whatsoever resulting from the information of this book as constant revisions and updates will be published after this edition.

Clarification from writer/ complier team

All Illustrations and Technical information found in this handbook are in our best interest to simplify working processes and equipment principles so that it could be easily understood by any user interested in robotics.

Therefore, The translation from THAI language to English and the usage of technical terms may not follow the provision of the Royal Academy where they are many words not described officially. Our team would be allowed to produce new technical terms.

The main reason of this explanation comes from data collection of equipment in embedded computer system and robotic technology. Thai language is quite hard to translate into English and thus our writer team gathered the required data and investigated to make sure that the understanding in working processes has the limited error.

When we compose the information in Thai, many technical terms have complicated meanings. Definition of vocabulary occurred from the practice coordinated with linguistic meaning. If there are any errors or mistakes shown, the team of writer will accept and if we get explanation or suggestion from any expert, we will clarify and improve those errors as soon as possible.

In order to develop the academic media especially with new technological knowledge, it will be able to proceed continually under the participation of experts in all fields.

Innovative Experiment Co.,Ltd.

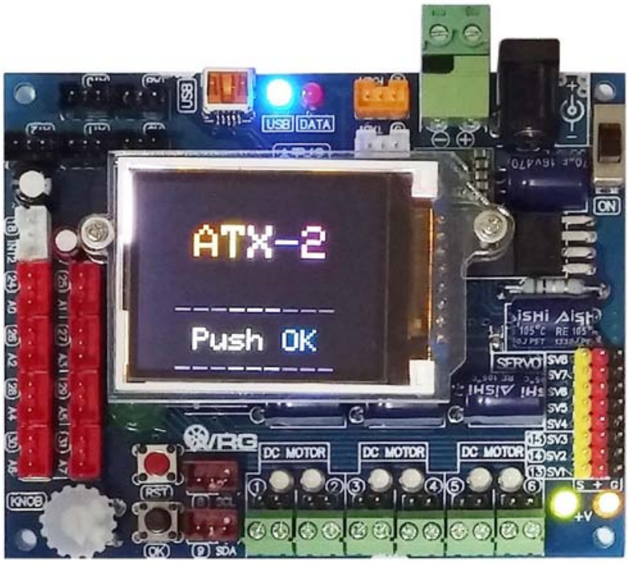
Table of contents

Chapter 1

ATX2 controller board overview

ATX2 is one important part of the Robo-Creator2 robot kit. This chapter introduces about technical information of the ATX2 controller board. It is base knowledge for using to make the autonomous robot such as Sumo-BOT, Fire fighting robot, Line tracking robot, All-terrain robot. By connecting with wireless communication module such as RF module, Bluetooth module and WiFi module we can adapt to control robot wirelessly by using smartphone, tablet and computer.

Moreover this ATX2 controller board is recommended for making the robot for World Robot Games competition.



Programming development of the ATX2 controller board uses C/C++ language in the open source software are named **Arduino** (www.arduino.cc). Free download the customize Arduino software for the ATX2 and Robo-Creator2 at ***www.inexglobal.com***.

ATX2 controller boad is next generation of INEX's ATX board that have developed since 2012. The new version ATX2 board is able to drive 6 of DC motors and 8 of Servo motors together. It has many ports for interfacing with both digital and analog sensors, basic digital outpuit port and using the data communication via 2 lined bus system called I2C bus and the standard UART serial bus. The improved feature of new version is the Color Graphic LCD (GLCD). It helps user to monitor the status, shows text and simple graphic in color.

1.1 ATX2 controller board features

In the figure 1-1, it is shown components of the ATX2 controller board and there are significant technical features as follows:

● Main microcontroller is Atmel's ATmeg644. It features 8-ch 10-bit Analog to Digital Converter, 128-KByte Flash memory , 4-KByte EEPROM, 4-KByte RAM. Operated with 16MHz clock from external crystal

●Define all ports compatible with Arduino I/O standard hardware. The number of port are 0 to 31. All ports available with 3-pin of 2.00mm. header. (+5V, Signal and GND)

● 13 programmable port in JST connector type. Includes Digital I/O port (2, 3, 8, 9, 18, 24 to 31), A/D port (8 : port 24/A0 to 31/A7), Two-wire interface or TWI (8/SCL and 9/SDA) and UART serial port communication (2/RxD1 and 3/TxD1). Both TWI and UART ports can config to digital input/output port for more I/O applications.

● Extension analog input (A8 to A12) and One variable resistor is labeled KNOB and OK button for ADC experiments. Only these inputs are analog input only.

● All analog supports 0 to +5Vdc input. The converter resolution is 10-bit. The result value is 0 to 1,023 range.

● One of button switch "OK" for simple digital input experiment.

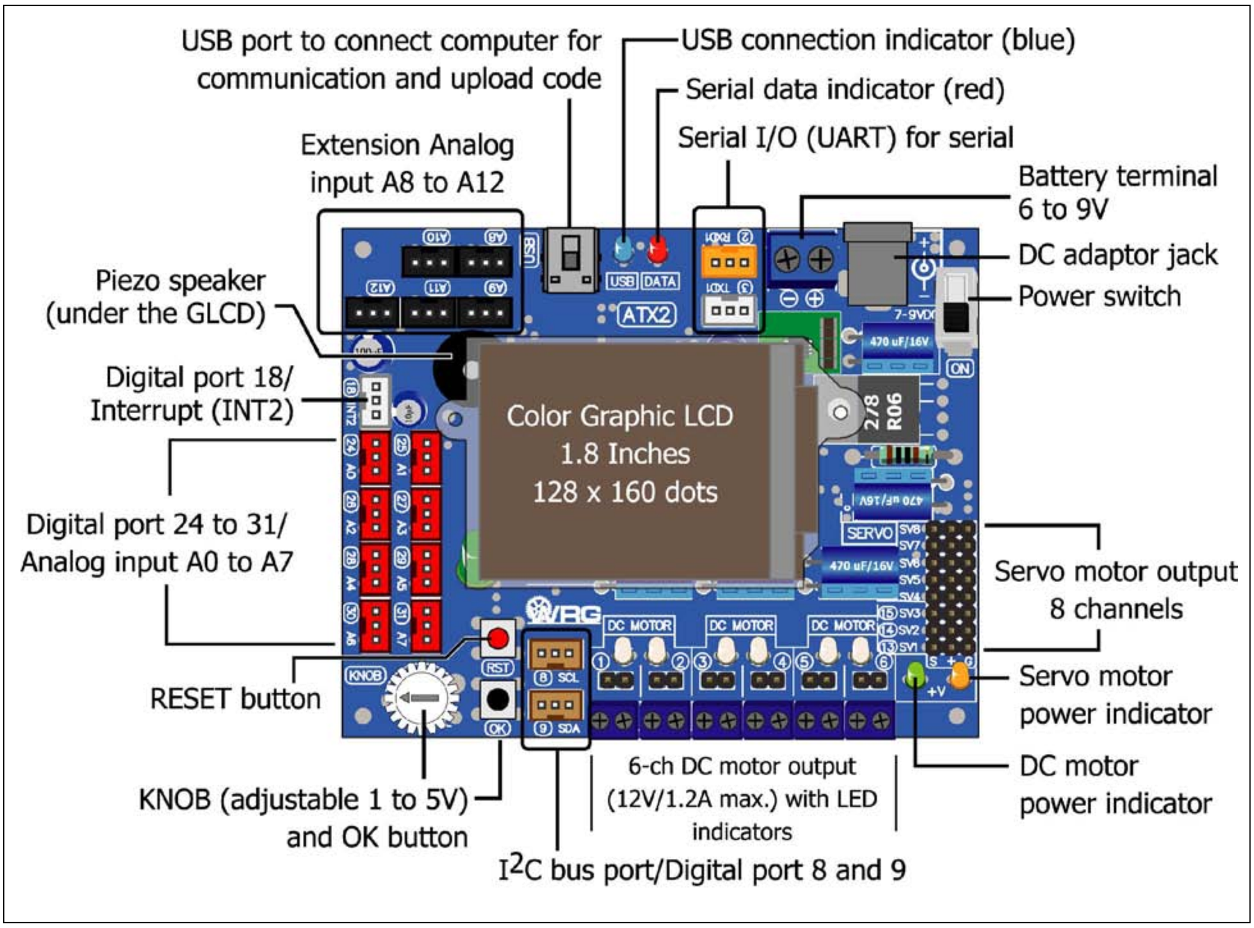

USB port to connect computer for communication and upload code
USB connection indicator (blue)
Serial data indicator (red)
Serial I/O (UART) for serial
Extension Analog input A8 to A12
Battery terminal 6 to 9V
DC adaptor jack
Power switch
Piezo speaker (under the GLCD)
Digital port 18/ Interrupt (INT2)
Color Graphic LCD 1.8 Inches 128 x 160 dots
Digital port 24 to 31/ Analog input A0 to A7
Servo motor output 8 channels
Servo motor power indicator
RESET button
DC motor power indicator
KNOB (adjustable 1 to 5V) and OK button
6-ch DC motor output (12V/1.2A max.) with LED indicators
I2C bus port/Digital port 8 and 9

Figure 1-1 : ATX2 controller board layout

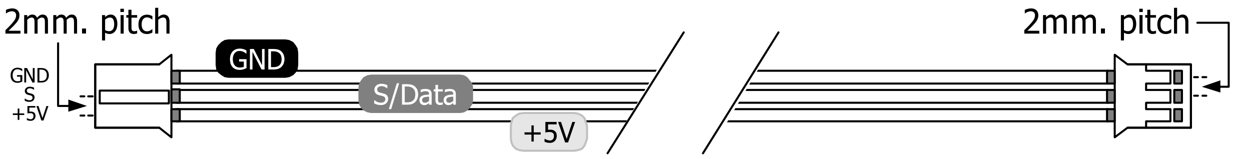
- One piezo speaker
- UART port (RxD1 and TxD1) for interfacing serial module device such as Pixy camera module, servo controller board (Parallax servo controller, ZX-SERVO16i), Real-time clock (ZX-17 : serial real-time clock module)
- 128x160 dots color GLCD. It only supports line-art , color background and text with 21 characters 16 lines (no support the image file).
- 6-ch of DC motor driver with LED indicator. Support 4.5 to 9V motor 1.2A (max)
- 8-ch of Servo motor outputs
- Upload with computer via USB port
- 2 of power inputs ; DC adaptor jack and 2-pin terminal block for battery
- Power switch and RESET switch available
- Requires +6 to 9V 500mA supply voltage in normal operation (no motor driving) and/or at least 1500mA for robotics application.
- +5V switching regualator on-board with polarity protection circuit

1.2 ATX2 cable information

For using the ATX2 controller board requires 2 kinds of cable. One is miniB-USB cable for interfacing with computer and JST3AA-8 cable for connecting with any sensor board and input/output devices.

1.2.1 JST3AA-8 cable

This is an INEX standard cable, 3-wires combined with 2mm. The JST connector is at each end. 8 inches (20cm.) in length. The wire assignment is shown in the diagram below.

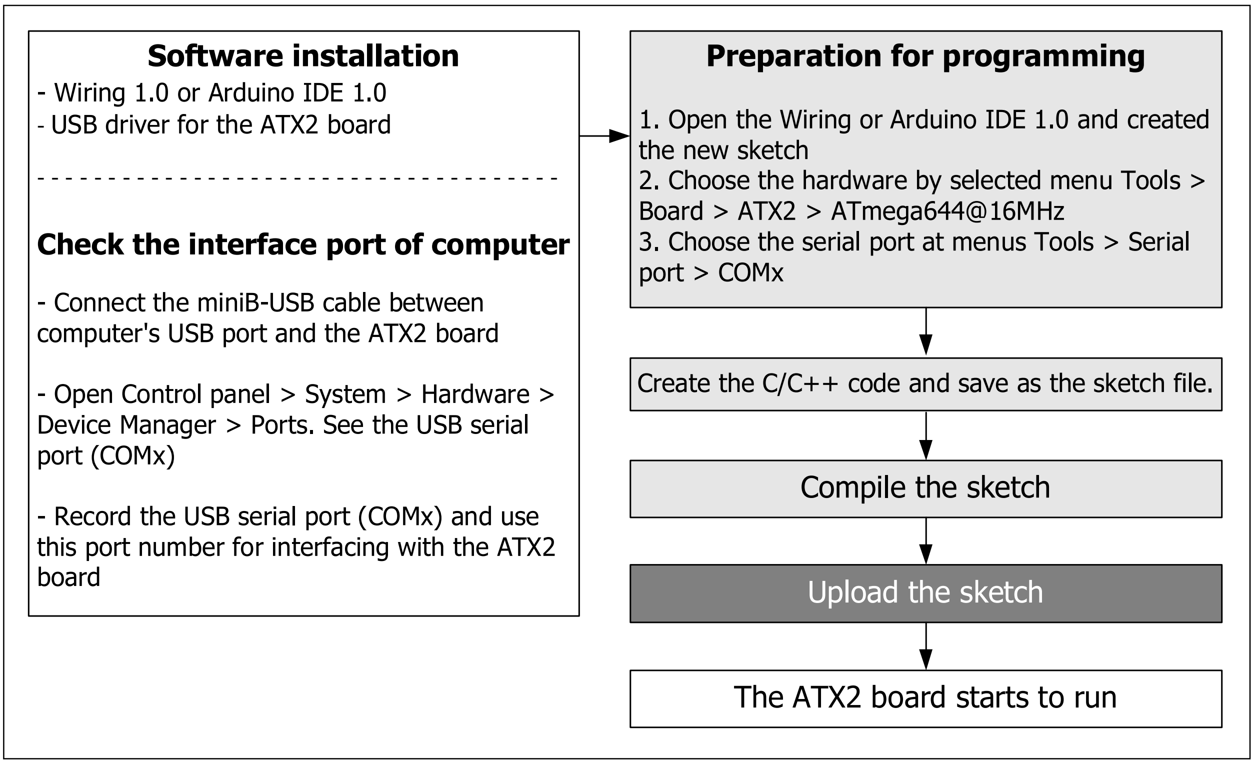


1.2.2 Standard USB-miniB cable

This is used to connect between the computer's USB port and the ATX2 controller board. Its length is 1.5 metres approximation.

1.3 How to develop program with ATX2 controller board

The programming development for the ATX2 controller board and Robo-Creator2 robot kit is the process by outlining the diagram in figure 1-2. The suitable software tools are Arduino IDE1.0.7 (customize version for Robo-Creator2 robot kit and ATX2 board). Free download from www.inexglobal.com.



Software installation

- Wiring 1.0 or Arduino IDE 1.0
- USB driver for the ATX2 board

Check the interface port of computer

- Connect the miniB-USB cable between computer's USB port and the ATX2 board

- Open Control panel > System > Hardware > Device Manager > Ports. See the USB serial port (COMx)

- Record the USB serial port (COMx) and use this port number for interfacing with the ATX2 board

Preparation for programming

1. Open the Wiring or Arduino IDE 1.0 and created the new sketch
2. Choose the hardware by selected menu Tools > Board > ATX2 > ATmega644@16MHz
3. Choose the serial port at menus Tools > Serial port > COMx

Create the C/C++ code and save as the sketch file.

Compile the sketch

Upload the sketch

The ATX2 board starts to run

Figure 1-2 : Programming development diagram of ATX2 controller board and All robot activities of Robo-Creator2 robot kit

Chapter 2

Development tools : Arduino IDE

Robo-Creator2 robot kit and ATX2 controller board support the operation controller program which can be developed from Assembly, BASIC or C programming languages. For here, we will use C/C++ programming language with the open-source software called Arduino, which is the name of a development project of a small control system in order to apply the software and hardware together.

We focus on concrete utilization as well as the connection of devices with electronic system so that the system can work according to the statement written correctly, collectively, Physical computing or the computer system which concentrates on physical signal connection, connecting external sensor devices or controlling display of LEDs, light, and sound, etc.

2.1 Arduino introduction

The official website of Arduino here is ***www.arduino.cc***. At this website, there is data of both hardware and software allowed to download with free of charge. Also, it is the open-source project to give an opportunity to developers who will be able to join the project and expand the project freely.

Arduino started in 2005 as a project for students at the Interaction Design Institute Ivrea in Ivrea, Italy. The founder is Massimo Banzi. The name "Arduino" comes from a bar in Ivrea, where some of the founders of the project used to meet. The bar itself was named after Arduino, Margrave of Ivrea and King of Italy from 1002 to 1014.

A hardware thesis was contributed for a wiring design by Colombian student Hernando Barragan. After the Wiring platform was complete, researchers worked to make it lighter, less expensive, and available to the open source community. The school eventually closed, but the researchers, including David Cuartielles, promoted the idea.

However for Robo-Creator2 robot kit and ATX2 controller board program developent will be use customized version of Arduino IDE that developed by Innovative Experiment Co.,Ltd. (INEX - www.inexglobal.com). It is Arduino 1.0.7. This version includes the ATX2 library that suitable for ATX2 controller board and Robo-Creator2 robot kit activities. Free download at www.inexglobal.com.

2.2 Supported operating system

The software for the program development is Arduino Development Environment or sometimes called Arduino IDE and it can work with these operating systems or platforms.

- Mac OS X 10.4 or higher (both models using Powerpc and Intel CPU)
- Windows XP, Windows Vista, 7 and 8/8.1 or higher
- Linux, both Fedora Core and Debian (including Ubuntu as well)
- Other platforms which support the operation of Java 1.4 up

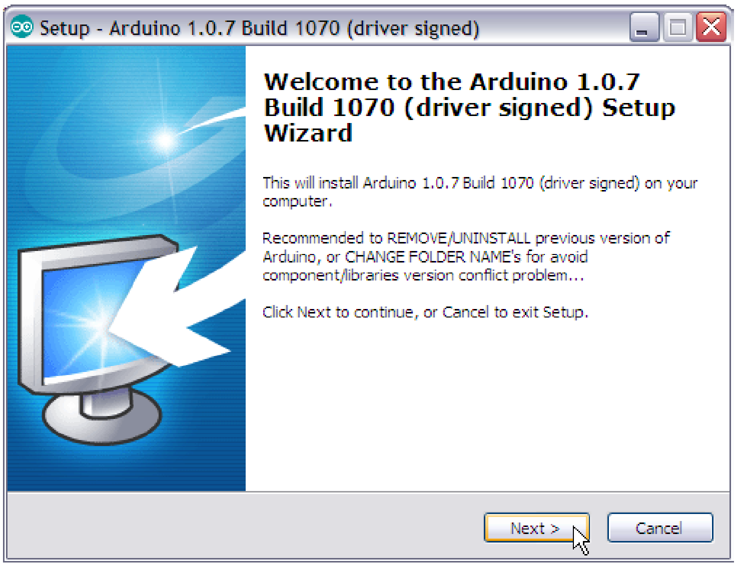
2.3 Introduction to Arduino 1.0.7 IDE

Arduino 1.0.7 is the software for developing C/C++ programming language in order to create a program controlling ATmega644 microcontroller on ATX2 controller board and then use the program in Robo-Creator2 kit as well.

In the kit, tools used in development of the program are contained completely in the format of IDE (Integrated Development Environment), either the text editor for coding or C complier. Uploading the sketch and the window of serial monitor for receiving and sending serial information to ATX2 controller board and robot kit. Arduino is designed to use easily and the C/C++ language is for programming which can work on the operating system of Windows XP up, Linux and MAC OSX and installation files for each platform are separated.

2.3.1 Software installation

(1) Insert the CD Rom, come with Robo-Creator2 robot kit. Click the file named ***Arduino1.0.7.1070_ReleaseSetup150123.exe*** (the number of the installation file may be changeable) and then the window of welcoming to the Arduino setup will appear.



Setup - Arduino 1.0.7 Build 1070 (driver signed)

Welcome to the Arduino 1.0.7 Build 1070 (driver signed) Setup Wizard

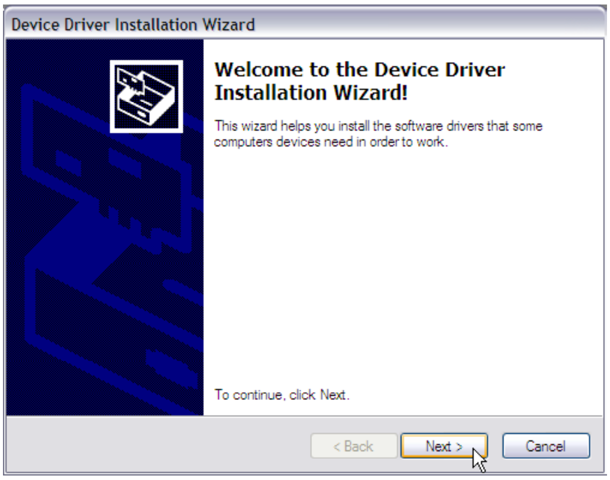
This will install Arduino 1.0.7 Build 1070 (driver signed) on your computer.

Recommended to REMOVE/UNINSTALL previous version of Arduino, or CHANGE FOLDER NAME's for avoid component/libraries version conflict problem...

Click Next to continue, or Cancel to exit Setup.

(2) Next, click to agree in each step of the setup as installation of other applications of Windows until completion.

(3) Installing the Arduino 1.0.7 software by using the CD rom is bundled with Robo-Creator2 robot kit is the setup of both Arduino software and USB driver to connect with ATX2 controller board in the same time. After Arduino IDE is installed, the driver installation window will appear. Click on **Next** button to install the first USB driver. Wait until installation finish. Click on **Finish** button following the picture below.

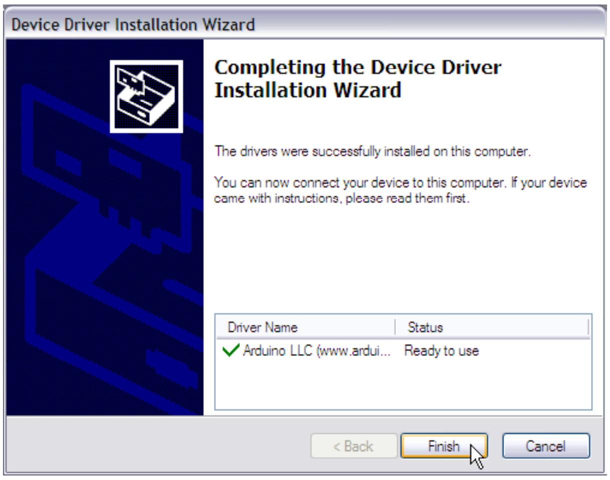


Device Driver Installation Wizard

Welcome to the Device Driver Installation Wizard!

This wizard helps you install the software drivers that some computers devices need in order to work.

To continue, click Next.



Device Driver Installation Wizard

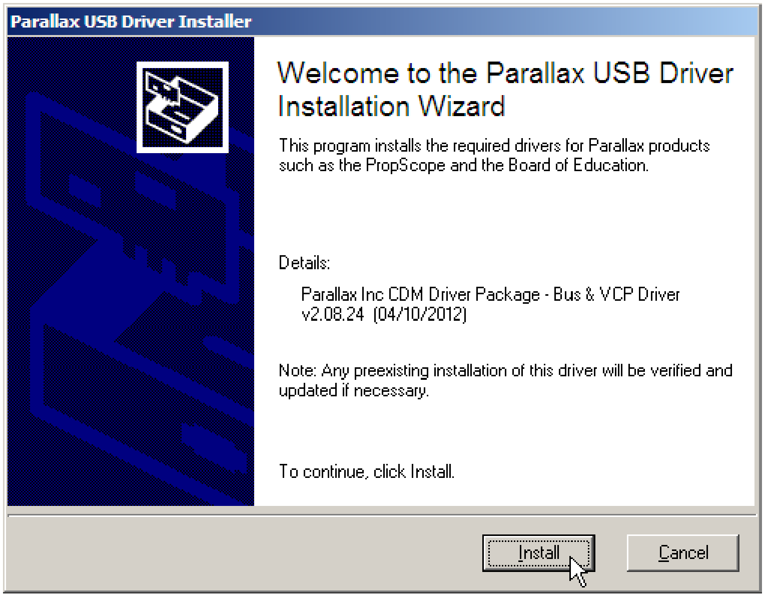
Completing the Device Driver Installation Wizard

The drivers were successfully installed on this computer.

You can now connect your device to this computer. If your device came with instructions, please read them first.

Driver Name	Status
✓ Arduino LLC (www.ardui...	Ready to use

(4) Next, the second USB driver will be install. Click on the **Install** button to start the driver installation. Wait until the installation is completed and finished.



Parallax USB Driver Installer

Welcome to the Parallax USB Driver Installation Wizard

This program installs the required drivers for Parallax products such as the PropScope and the Board of Education.

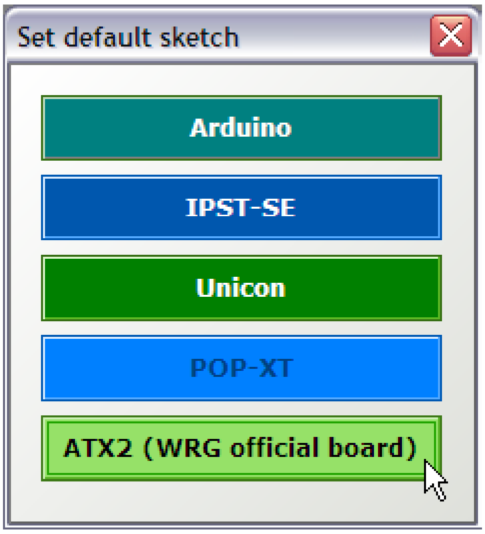
Details:

Parallax Inc CDM Driver Package - Bus & VCP Driver
v2.08.24 (04/10/2012)

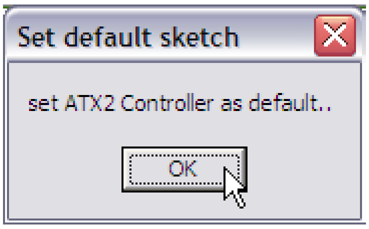
Note: Any preexisting installation of this driver will be verified and updated if necessary.

To continue, click Install.

(4) The hardware console window is appeared. This window will help programmer to prepare suitable library to support the chosen hardware. Select the ATX2 for ATX2 controller board and Robo-Creator2 robot kit and Click on the OK button again to confirmation.



Set default sketch



Set default sketch

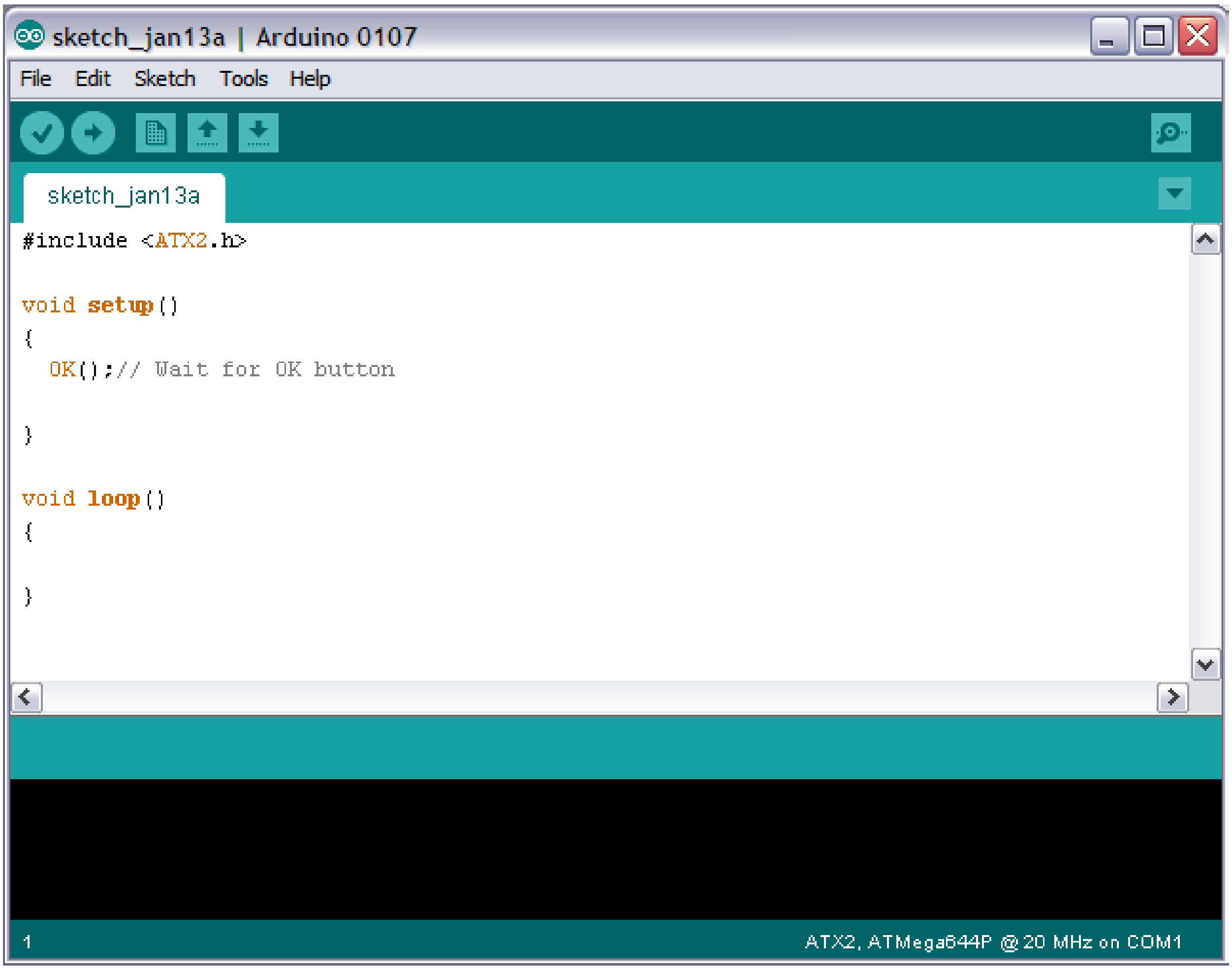
set ATX2 Controller as default..

(5) The Arduino IDE will be start.

AN OPEN PROJECT WRITTEN, DEBUGGED AND SUPPORTED BY MASSIMO BANZI, DAVID CUARTIELLES, TOM IGOE, GIANLUCA MARTINO AND DAVID MELLIS

BASED ON PROCESSING BY CASEY REAS AND BEN FRY

(6) The main window of Arduino IDE will appeared. The default template will be loaded. User can start easy and reduce the error from missing the main library.



```
#include <ATX2.h>

void setup()
{
  OK();// Wait for OK button

}

void loop()
{

}
```

```
1                                        ATX2, ATMega644P @ 20 MHz on COM1
```

(6.1) See the first line of sketch. Default setting will include ATX2.h the main library at the first line

(6.2) The name of chosen hardware will show at the bottom of IDE; ***ATX2.ATmega644@20MHz on COMx*** (x is any COM port or USB serial port number).

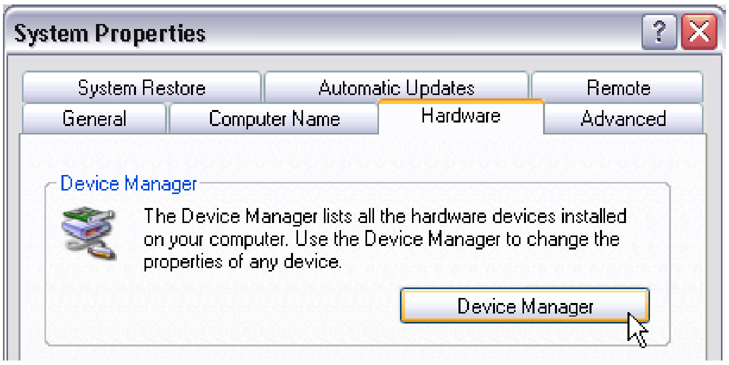
2.3.2 Checking the USB Serial port of the ATX2 controller board

(1) Plug the USB cable connecting ATX2 controller board with the USB port of the computer. Turn on and wait for the blue LED at the position of USB on the circuit board is on.

(2) Click the **START** button and go to the **Control Panel**.

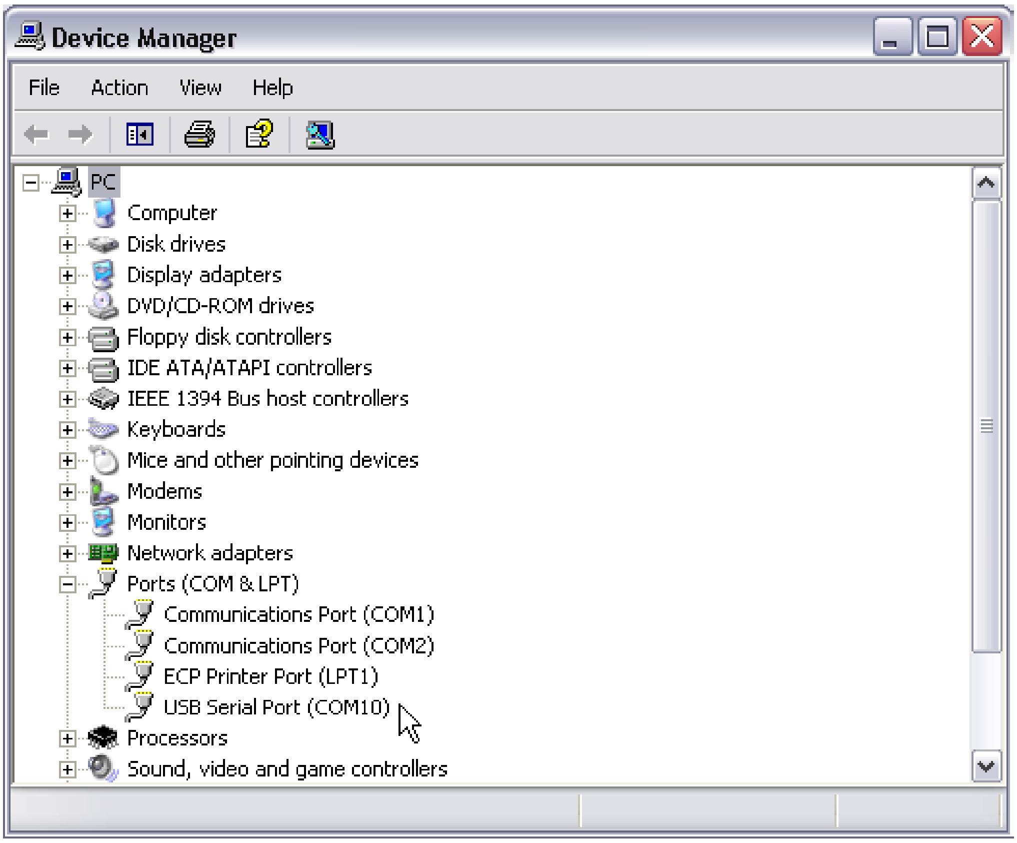
(3) Then double-click the **System**

(4) Go to the tab of **Hardware** and click on the **Device Manager** button



The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.

(5) Check the hardware listing at **Port.** You should see **USB Serial port** . Check the position. Normally it is COM3 or higher (for example; COM10). You must use this COM port with the Wiring IDE software.

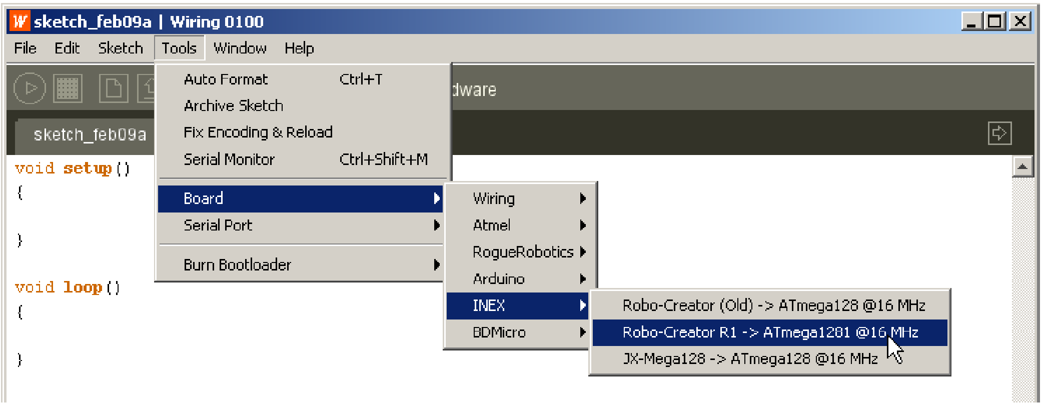


- PC
  - Computer
  - Disk drives
  - Display adapters
  - DVD/CD-ROM drives
  - Floppy disk controllers
  - IDE ATA/ATAPI controllers
  - IEEE 1394 Bus host controllers
  - Keyboards
  - Mice and other pointing devices
  - Modems
  - Monitors
  - Network adapters
  - Ports (COM & LPT)
    - Communications Port (COM1)
    - Communications Port (COM2)
    - ECP Printer Port (LPT1)
    - USB Serial Port (COM10)
  - Processors
  - Sound, video and game controllers

2.3.3 ATX2 with Arduino IDE interface

(1) Open **Arduino IDE**. Wait for a while. The main window of Arduino will appear.

(2) Choose the suitable hardware by select menu **Tools > Board > ATX2; ATmega644 @16MHz**


sketch_feb09a | Wiring 0100
File Edit Sketch Tools Window Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Burn Bootloader
Wiring
Atmel
RogueRobotics
Arduino
INEX
BDMicro
Robo-Creator (Old) -> ATmega128 @16 MHz
Robo-Creator R1 -> ATmega1281 @16 MHz
JX-Mega128 -> ATmega128 @16 MHz
dware
sketch_feb09a

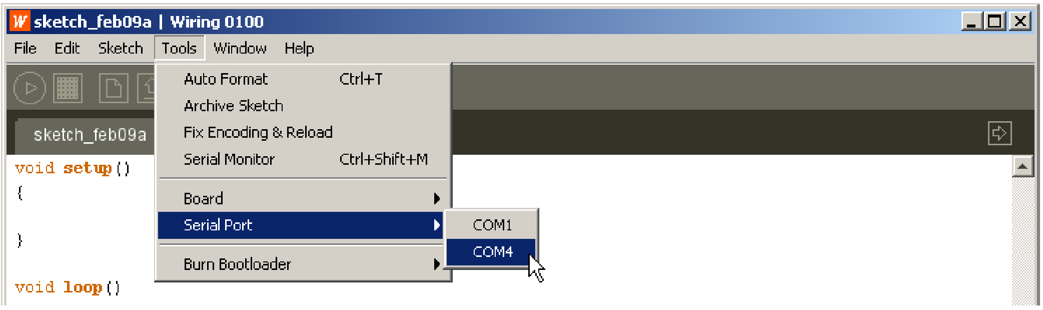
```
void setup()
{

}

void loop()
{

}
```

(3) Select menu **Tools > Serial Port** to choose the USB serial port of ATX2 board. It is **COM3** (for example).



```
void setup()
{

}

void loop()
```

Must do this step for every new connection of the ATX2 board with Arduino IDE 1.0.7

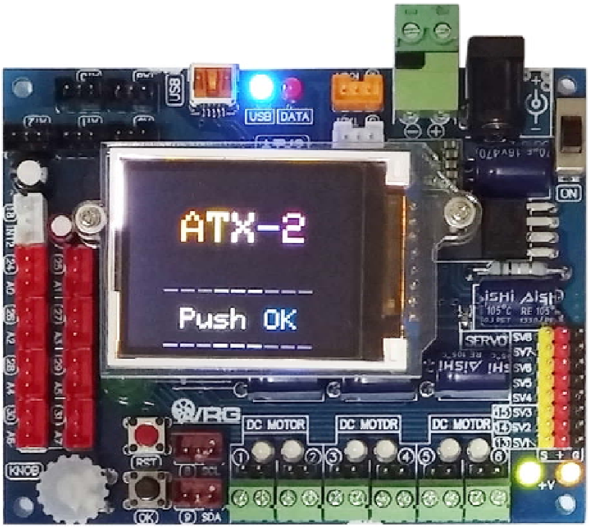
Now the ATX2 controller board is ready for interfacing and code development with the Arduino IDE 1.0.7 (ATX2 customized version)

(4) Type this code below and save as atx2_hello.ino

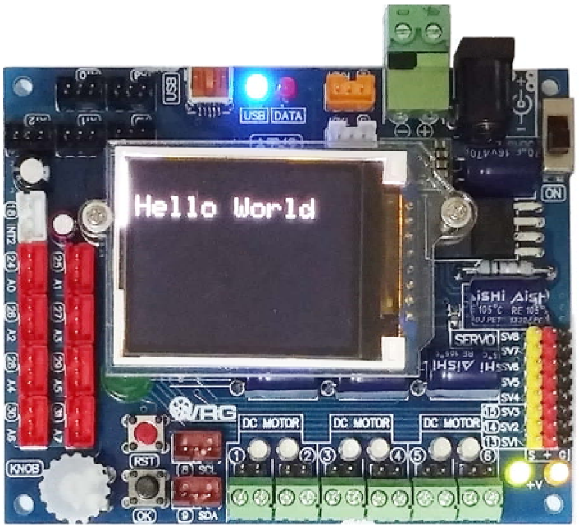
```
#include <ATX2.h>
void setup()
{
   OK();                         // Wait for OK button
   glcdClear();                  // Clear screen
}
void loop()
{
   glcd(1,0,"Hello World");      // Show message
}
```

(5) Upload this sketch to the ATX2 board and run it.

The ATX2 board displays the welcome menu and wait for pressing the **OK** button. After press the **OK** button. ATX2's screen will display message Hello World at the first line of screen.



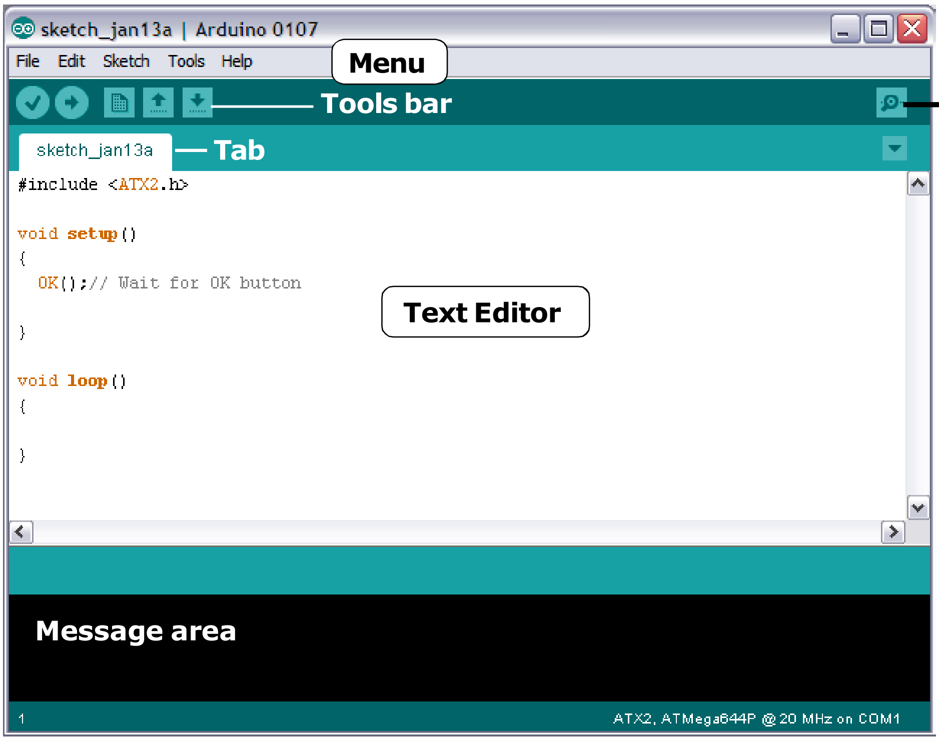
After upload and run. Wait for pressing OK button



After pressing the OK button

2.4 Arduino environment

After starting Arduino IDE 1.0.7, the main window will appear as shwon in the figure 2-1. The Arduino includes the environments as follows.


sketch_jan13a | Arduino 0107
File Edit Sketch Tools Help
Menu
Tools bar
sketch_jan13a
Tab
Text Editor
Message area
1
ATX2, ATMega644P @ 20 MHz on COM1

```
#include <ATX2.h>

void setup()
{
  OK();// Wait for OK button

}

void loop()
{

}
```

Serial Monitor
Click to open the other window for transmitting and receiving the serial data between Aruino hardware and computer

Figure 2-1 Arduino 1.0 IDE environment

● **Menu :** Select the operation command

● **Toolbar :** Includes all most command button

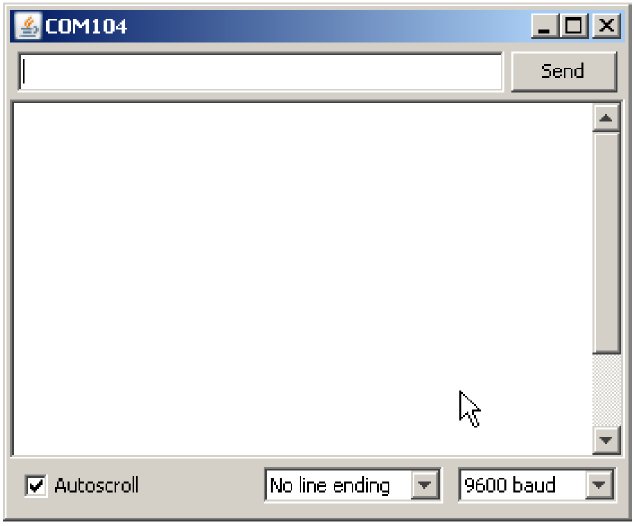
● **Tabs** : Allows you to manage sketches with more than one file (each of which appears in its own tab).

● **Text editor** : Text editor area for creating the sketch.

● **Message area :** Shows the program operation status such as compiling result.

● **Text area :** The space demonstrates compiling information and Serial data Terminal window if enable.

● **Serial Monitor** This button is on the top right corner. Click this button to open the serial communications and display information. Requires the Arduino hardware must connect with the COM port always.



Serial Monitor is a important for displaying the serial data on the computer. Arduino has this command ; **Serial.print** to display data. For sending data from computer to Arduino is very easy. Only type the data on the text box and click on the **Send** button at the top right corner of Serial monitor window. For using Mcintosh computer or Linux computer; the Arduinoi hardware will need be reset when the Serial monitor is opened every time.

2.4.1 Menu bar

2.4.1.1 File

The Arduino calls the code as **Sketch.** This menu contains many commands like open, save and close the sketch as follows :

● **New** : Creates a new sketch, named is the current date format "sketch_ YYMMDDa".

Open : Open the exist sketch.

Sketchbook : Opne the latest sketch file

● **Example** : Open the example sketch.

● **Save** : Save the current sketch

● **Save as** : Save the current sketch as another name.

● **Upload to I/O board** : Uploads your code to the Arduino I/O board (POP-XT). Make sure to save or verify your sketch before uploading it.

● **Page setup** : Set up the page layout for the current sketch

● **Print** : Print the current sketch code to the printer

● **Preference** : Set some preference of Arduino environment

● **Quit** : Exit the Arduino IDE

2.4.1.2 Edit

The Edit menu provides a series of commands for editing the Arduino files.

● **Undo** : Reverses the last command or the last entry typed.

● **Redo** : Reverses the action of the last Undo command. This option is only available, if there has already been an Undo action.

● **Cut** : Removes and copies selected text to the clipboard.

● **Copy** : Copies selected text to the clipboard.

● **Paste** : Inserts the contents of the clipboard at the location of the cursor, and replaces any selected text.

● **Select All** : Selects all of the text in the file which is currently open in the text editor.

● **Find** : Finds an occurance of a text string within the file open in the text editor and gives the option to replace it with a different text.

● **Find Next** : Finds the next occurance of a text string within the file open in the text editor.

2.4.1.3 Sketch

This menu provides a series of commands for compile the code and manage library.

- **Verify/Compile** : Verify and compiles the code
- **Stop** : Stops current activity.
- **Add file** : Opens a file navigator. Select a code files to add it to the sketches "data" directory.
- **Import Library** : Import the addition library.
- **Show Sketch folder** : Opens the directory for the current sketch.

2.4.1.4 Tools

This menu provides commands about tools for developing the Arduino sketch and setting up the Arduino hardware.

- **Auto Format** : Attempts to format the code into a more human-readable layout. Auto Format was previously called Beautify.
- **Archive Sketch** : Compress the current sketch to the zip file.
- **Export Folder** : Open the folder that contain the curretn sketch.
- **Board** : Choose the Arduino hardware. For POP-BOT XT, choose ***POP-XT***
- **Serial Port** :Allows to select which serial port to use as default for uploading code to the Arduino I/O Board or monitor data coming from it. The data coming from the Arduino I/O Board is printed in character format in the text area region of the console.

2.4.1.5 Help

This menu contains many information in HTML format for supporting the Arduino user.

- **Getting Start** : Opens the How to start Arduino.
- **Troubleshooting** : Suggest the solution when meet the problem in Arduino.
- **Environment** : Describe about Arduino environments
- **Reference** : Opens the reference in the default Web browser. Includes reference for the language, programming environment, libraries, and a language comparison.
- **Frequently Asked Question** : See the popular question and answer
- **Visit www.arduino.cc** : Opens the web browser to the Arduino homepage.
- **About Arduino** : Opens a concise information panel about the software.

2.4.2 Tools bar

✓ **Verify/Compile** : Checks your code for errors.

➡ **Upload to I/O Board** : Uploads your code to the Arduino I/O board (POP-168 module). Make sure to save or verify your sketch before uploading it.

▦ **New** : Creates a new sketch.

⬆ **Open** : Presents a menu of all the sketches in your sketchbook.

⬇ **Save** : Saves your sketch.

🔍 **Serial Monitor** : Open the Serial Monitor window to get and show data via USB port of Arduino hardware.

Chapter 3

Arduino programming

Programming for ATX2 controller board and Robo-Creator2 robot kit are done by using the Arduino programming language (based on C++) with version 1.0 or higher, in which the language of Arduino itself is developed from the open-source project framework for microcontrollers called Wiring and the language of Arduino is grouped into two parts as follows:

1. Language structure, variables, and constants
2. Functions

Arduino language is based on the C/C++ languages so that when programming for Arduino (including ATX2 controller board) one can use the functions and libraries which already exist for the C language. This makes it convenient for those who do not understand the intricacies of microcontrollers to write command programs for them.

This chapter will mainly describe the structure of Arduino. For a complete text written about the function of the C programming language behind Arduino, see details in help menu of Arduino IDE or their website; www. arduino.cc

3.1 Program structure of Arduino

An Arduino program is composed of two sections including

```
void setup ( )
```

and

```
void loop ( )
```

The function of **`setup ()`** : when a program runs, it will make a statement for this function only once to define a default value for operation.

The function of **`loop ()`** is a procedure in the program that contains statements which are repeated.

Normally, this procedure is performed to specify working modes of the various pins in the serial communication bus, and so on. The loop contains the program code which is executed, such as reading input values, processing, displaying output etc. Any configuration of default values, or components such as variables, need to be declared at the beginning of the program before reaching the function part. In addition, you should consider whether to use small letters or capital letters for variables, and name the functions meaningfully.

3.1.1 setup () function description

This function is written at the beginning of a program and executes just once when the program starts. It is used to declare variables, operating modes of various pins, or the initialization of libraries, etc.

Example 3-1

```
int buttonPin = 3;
void setup()
{
   beginSerial(9600);
   pinMode(buttonPin, INPUT);
}
void loop()
{
   if (digitalRead(buttonPin) == HIGH)
      serialWrite('H');
   else
      serialWrite('L');
   delay(1000);
}
```

While the standard C programming language is written on AVR GCC (a kind of C programming language using the GCC compiler for the AVR microcontroller) will be coded as follows.

```
int main(void)
{
   init();
   setup();
   for (;;)
   loop();
   return ;
}
```

compliance with `void setup()`

compliance with `void loop()`

3.1.2 loop () function description

After coding the function of setup ( ), which has already defined the default values and initial state of the program, the next part is the loop( ) function. The program will repeatedly execute this function until otherwise instructed to do so. Inside this function, there are user programs which read values from ports, perform compilations, and control output through various pins in order to control the performance of board.

Example 3-2

```
int buttonPin = 3;       // Declare pin 3 to buttonPin variable
void setup()
{
   beginSerial(9600);
   pinMode(buttonPin, INPUT);
}
// A loop which detects the pressing of a switch
// that declared with buttonPin variable.
void loop()
{
   if (digitalRead(buttonPin) == HIGH)
      serialWrite('H');
   else
      serialWrite('L');
   delay(1000);
}
```

3.2 Operational control statements

3.2.1 If

Used to test for evaluating conditions during program operation, for example, if an input value is greater than a certain value, what should be done in that case. The syntax is shown as follows.

```
if (someVariable > 50)
{
   // do something here
}
```

The program will check whether the value of someVariable is greater than 50 or not. If so, what will have to do but if it is not, just skip to the function of this section.

The function of this command is to test certain conditions, which are written in brackets. If the condition is true, follow an instruction in braces but if the condition is false, leave out the function of this section.

The conditional test which is in brackets will have to use these comparison operators as follows.

x == y (x equal to y)

x != y (x not equal to y)

x < y (x less than y)

x > y (x greater than y)

x <= y (x less than or equal to y)

x >= y (x greater than or equal to y)

Techniques for programming

In the comparison of variables, you should use the operator ==, e.g. if (x==10). Do not use =, such as if (x=10), because a statement like this assigns a value of ten(10) to the variable x.

Moreover, If statements can be applied to control intersection of the program and the If...else statements can be used as well.

3.2.2 If…else

Applied to test for determining conditions of the program operation and they are better than normal **`if`** statements. The process is that if a condition is true, what to do but if it is false, what to do so. For example, If a readable value of analog input is greater than 500, what to do but if the value is greater than or equal to 500, do another one else. Writing statements as follows:

Example 3-3

```
if (pinFiveInput < 500)
{
   // A statement for a kind of operation if a value of pinFiveInput
   //is less than 500
}
else
{
   // A statement for another type of operation if a value of pinFiveInput
   // is greater than or equal to 500
}
```

After the **`if`**…**`else`** statement, able to be followed by the if command. Therefore, syntax becomes **`if`**…**`else...if`**. This is the test of conditions. When it is true, meet the demand as the following example.

Example 3-4

```
if (pinFiveInput < 500)
{
   // The statement for run a function, since pinFivelnput is less than 500
}
else if (pinFiveInput >= 1000)
{
   // The statement for run another function
   // because pinFivelnput is greater than or equal to 1000
}
else
{
   // The statement to set the next step in case that
   // pinFiveInput is not less than 500 and greater than or equal to
   // 1000 (that is there will be response of the statement in this
   // subprogram when a variable has a value between 501 to 999
}
```

After the **`else`** statement, able to be followed by unlimited if statements (or capable of putting the **`switch case`** statement instead of **`if...else...if`** statement for testing a lot of conditions)

When the **`if...else...`** statement has been made, it is important to determine that if the examination does not match with any condition, what to do by specifying at the last **`else`** statement.

3.2.3 For ()

`For ()` is used to instruct a statement in braces in order to repeat operation following number of cycles required. This statement is very useful for any function that needs to repeat and know the exact number of repetition round. It is often applied with Array variables in collection of readable values from many analog input pins that have pin numbers in sequence.

The syntax of `For ()` command is divided into three parts as follows :

```
for (initialization; condition; increment)
{
   //statement(s);
}
```

Beginning with **`initialization`** used to assign a default value of loop control variables. In each work cycle, a **`condition`** is checked. If the condition is true, a statement in braces is responded and a variable value is increased or decreased according to the order in **`increment`**. Repeating the test until the condition is false.

Example 3-5

```
for (int i=1; i <= 8; i++)
{
   // The statement for function by using a value of variable i and
   // repeating the function until the value of variable i is
   // greather than 8 ;
}
```

`For` statement of C language is more flexible than whose other computer languages are. It can ignore some parts or all three parts of the for statement. However, semicolon is still necessary.

3.2.4 Switch-case

The statement is used to examine conditions in the determination of program operation. If a tested variable matches with a condition, the run will follow the determined process.

Syntax

```
switch (var)
{
   case 1:
   // The statement for operation when the variable value is equal to 1
   break;
   case 2:
   // The statement for operation when the variable value is equal to 2
   break;
   default:
   // If the value of variable is neither 1 nor 2, comply with this order.
}
```

Parameters

`var` - a variable which requires to test matching with which condition.

`default` - make a statement follow the end if it does not match with any condition,

`break` - a statement to stop running and it should be written appended cases. If it is not written, the program will perform repeatedly according to a condition.

3.2.5 While

It is a kind of loop statements in order to follow the instruction written in braces continually until a `while()` statement is false. A loop statement has to change a value of a variable, such as to increasing variable value, or has external conditions, such as ability to start and stop reading values from a sensor, respectively. Otherwise, the condition in the braces of `while ()` is always true, it causes the loop statement works endlessly.

Syntax

```
while(expression)
{
   // statement(s);
}
```

Parameters

`expression` is a conditional test statement (true or false)

Example 3-6

```
var = 0;
while(var < 200)
{
   // The statement for operation by repeating the operation in
   // total of 200 rounds
   var++;
}
```

3.3 Arithmetic Operators

There are 5 operators consisted of + (addition), - (substraction), * (multiplicaiton), and % (modulo reduction, remainder from integer division).

3.3.1 Arithmetic Operators, addition, substraction, multiplication, and division

They are used to calculate the sum, difference, the product, and quotient of binary operators to give answers that include the same type as both binary operatands, such as 9/4, which shows the answer as 2 because both 9 and 4 are integer variables (int).

Moreover, arithmetic operators may cause overflow if a result is larger than storing in that type of variables. If operands are different in types, a result will be as large as the type of biggest variable (such as 9/4 =2 or 9/4.0 = 2.25).

Syntax

```
result = value1 + value2;
result = value1 - value2;
result = value1 * value2;
result = value1 / value2;
```

Parameters

`value1` - the value of any variables or constant

`value2` - the value of any variable or any constant

Example 3-7

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Techniques for programming

You should :

- Choose the size of variables that is large enough to store the most result value of calculation.

- Know that what value of variables will return the value back and how it back, for example, (0 to 1) or (0 to -32768)

- Use float variables in the calculation of fraction but the difference should be concerned. For example, large variables can cause slow calculation.

- Change types of temporary variables by **`cast`** operators, such as `(int)-myfloat`, while a program is running

3.3.2 Modulus operator : %

The operator is able to compute remainder of 2 integers but not to apply for floating-point variables (float).

Syntax

```
result = value1 % value2;
```

Parameters

`value1` - a variable of byte, char, int or long type

`value2` - a variable of byte, char, int or long type

Result

Remainder from performing integer division is integer data.

Example 3-8

```
x = 7 % 5;   // x now contains 2
x = 9 % 5;   // x now contains 4
x = 5 % 5;   // x now contains 0
x = 4 % 5;   // x now contains 4
```

This modulus operator is applied for operation which demands events occurring at regular intervals or causes the memory, which stores array variables, to roll over.

Example 3-9

```
// Examine the value of a detector for 10 times per a operating cycle
void loop()
{
   i++;
   if ((i % 10) == 0)
   // Divide a value of i by 10 and then check
   // if the remainder of division is 0 or not
   {
      x = analogRead(sensPin);  // Read from the detector for 10 times
   }
}
```

*In this example, the `%` statement is used to assign the cycle of operation in which the program repeats performing to read value till the result of modulo division of the **`i % 10`** statement is equal to 0 and this occurs when i = 10 only.*

3.4 Comparison operators

Comparison operators go with the **`if ()`** statement to test conditions or compare variable values by writing expression inside the mark of **`()`**.

`x == y` `(x equal to y)`

`x != y` `(x not equal to y)`

`x < y` `(x less than y)`

`x > y` `(x greater than y)`

`x <= y` `(x less than or equal to y)`

`x >= y` `(x greater than or equal to y)`

3.5 Logical operators

Performed a for comparison of **`if ()`** statements and there are 3 types; **`&&`**, **`||`**, and **`!`**.

3.5.1 && (logic and)

Let a value is true when the comparison result of both sides is true.

Example 3-10

```
if (x > 0 && x < 5)
{
   // ...
}
```

The value is true when x is greater than 0 and less than 5 (a value of 1 to 4).

3.5.2 && (logic or)

The value is true when the comparison result shows that one of variables is true or both variables are true.

Example 3-11

```
if (x > 0 || y > 0)
{
   // ...
}
```

It shows the result is true when the value of x or y is greater than 0.

3.5.3 ! (used to convert the result to be contrary)

The value is true when the comparison result is false.

Example 3-12

```
if (!x)
{
   // ...
}
```

The result is true if x is false (such as x = 0, the result is true)

3.5.4 Caution

Be careful for programming. If you want to use logical operators, you have to write the mark of **&&**. If you write a mark of **&**, Bitwise **AND** operators and variables will show different results.

As well in application of logic or writing the sign of **||** (two vertical bars next to each other), if write the sign of **|** (one vertical bar) this means Bitwise **OR** operators and variables.

A bitwise **NOT** operator is different from a conversion operator (**!**) so you should choose the right one to apply.

Example 3-13

```
if (a >= 10 && a <= 20){}

// The result of performance is true when the value of a is between 10
// to 20.
```

3.6 Bitwise operators

Level operators will bring bit of variables to process and they are useful to solve problems of programming widely.

There are 6 level operators of C language (including Arduino) consisted of **&** (bitwise AND), **|** (OR), **^** (Exclusive OR), **~** (NOT), **<<** (shift bits to the left) and **>>** (shift bits to the right)

3.6.1 Bitwise AND operators (&)

AND in bitwise statements of C programming language can be represented by using the mark of **&** in and you must write between expressions or integer variables. The work will take data of each bit of both variables to operate with AND logic. The rules are as follows.

```
0  0  1  1     Operand1
0  1  0  1     Operand2
──────────
0  0  0  1     Returned result
```

If both inputs are '1', both outputs are '1'. Other cases, outputs are '0' as the following example. In checking, a pair of operators should be in the vertical.

In Arduino, int variables have the size of 16 bits so when using **AND** bitwise operators, there is logical action in conjunction with all 16- bit data as the example in the following program.

Example 3-14

```
int a =  92;    // equal to 0000000001011100 binary
int b = 101;    // equal to 0000000001100101 binary
int c = a & b;  // result is 0000000001000100 binary or 68 demical
```

In this example, all 16-bit data of variable a and b are performed by ***AND*** *logic and then the result of all 16 bits will be stored at variable c, which the value is 01000100 in binary number or 68 in decimal number.*

It is popular to use bitwise **AND** operators to select desired bit-data (maybe one bit or many bits) from int variables in which this selection of some bits is called masking.

3.6.2 Bitwise OR operator (I)

A bitwise **OR** statement of C language is able to be written by using one mark of **|** by writing between expressions or integer variables. The process is to bring data of each bit of both variables to operate with OR logic. The rules are as follows.

If any input or both are '1', so the output is '1'. In case that both input are '0' so the output is '0' as following example.

```
0  0  1  1      Operand1
0  1  0  1      Operand2
──────────
0  1  1  1      Returned result
```

Example 3-15

The program shows application of bitwise OR operators.

```
int a =  92;    // equal to 0000000001011100 binary
int b = 101;    // equal to 0000000001100101 binary
int c = a | b;  // The result is 0000000001111101 binary or 125 decimal
```

Example 3-16

The program demonstrates application of bitwise AND and OR operators.

The example of a program, which uses bitwise AND and OR operators, are called by programmers as Read-Modify-Write on a port. For 8-bit microcontroller, values of reading or writing to the port have the size of 8 bits and show input at all 8 pins. Writing values sending to a port is done only one time to cover all 8 bits.

The variable named PORTD is the value used instead of the status of digital pin numbers 0, 1, 2, 3, 4, 5, 6, 7. If any bit has a value as 1, this makes that pin have logic value as HIGH (don't forget to specify that port pin to work as output with pinMode command ( )) Therefore, if configuration of PORTD is determined = B00110001; this is to design pin 2, 3, and 7 as HIGH. In this case, it doesn't need to change the status values of pin 0 and 1. Normally, hardware of Arduino is used in the serial communication. If you convert a value, this will affect to the serial communication.

Algorithm for the program is as follows

● Read a value from PORTD and then clean up the value at only the controlled bit (using a bitwise AND operator).

● Take the modified PORTD value from above to add with the bit from above (OR operators).

Write the program as follows :

```
int i;                        // counter variable
int j;
void setup()
{
   DDRD = DDRD | B11111100;   // determine the direction of the
                              // port pin 2 to 7 with the value of
                              // 11111100
   Serial.begin(9600);
}
void loop()
{
   for (i=0; i<64; i++)
   {
      PORTD = PORTD & B00000011; // determine data to pin 2 to 7
      j = (i << 2);
      PORTD = PORTD | j;
      Serial.println(PORTD, BIN);
         // show the value of PORTD at the serial monitor window
      delay(100);
   }
}
```

3.6.3 Bitwise Exclusive OR statement (^)

It is a special operator which is not usually used in C/C ++ language. The **`exclusive OR`** operator (or **`XOR`**) is written by using the symbol of **^** and this operator has operation resembled to a bitwise **`OR`** operator but they are different when both inputs are '1', output is '0'. The demonstration of work is as follows.

```
0  0  1  1     Operand1
0  1  0  1     Operand2
__________
0  1  1  0     Returned result
```

Or it is also said that bitwise **`XOR`** operators give the output as '0' when both inputs are the same and give the output as '1' when both inputs are different.

<u>Example 3-17</u>

```
int x = 12;       // Binary number is 1100
int y = 10;       // Binary number is 1010
int z = x ^ y;    // Result is 0110 as binary or 6 as decimal
```

Bitwise **`XOR`** operators are much used in value swap of some bits of int variables, such as changing from '0' to be '1' or from '1' to '0'.

When using bitwise **`XOR`** operators, if a bit of mask is '1' so this affects the swop of bit value. Otherwise, if a value of mask is '1' the bit value remains the same. The following example is a program shown the instruction that logic of the digital pin 5 is switched all the time.

<u>Example 3-18</u>

```
void setup()
{
   DDRD = DDRD | B00100000;    // Assign pin 5 as output
}
void loop()
{
   PORTD = PORTD ^ B00100000;     // Invert logic at pin 5
   delay(100);
}
```

3.6.4 Bitwise NOT operator (~)

Bitwise **`NOT`** will write by using symbol **~**. This operator will be performed with only one operand on the right by switching all bits to become opposite values, i.e. from ‘0’ to be ‘1’ and from ‘1’ to be ‘0’ as the example.

```
0  1     Operand1
——————
1  0     ~ Operand1

int a = 103;    // binary: 0000000001100111
int b = ~a;     // binary: 1111111110011000
```

When operated already, this makes the value of variable be as -104 (decimal) in which the answer is negative because of the most important bit (the bit on the far left) of an int variable. An int variable is the bit which notifies whether the number is positive or negative. The value is ‘1’, meant that this value is negative. A computer will store the number value as both positive and negative according to 2’s complement system.

Declaration of int variables, which has the same meaning as the signed int, should be aware that the value of the variable may be negative.

3.6.5 Left shift (<<) and Right shift (>>)

In C/C++ programming language, there is an operator shifts bits to the left **<<** and shifts bits to the right **>>**. This operator will instruct to slide bits of operands written on the left to the left or the right in accordance with the amount of bits stated on the right of the operator.

Syntax

```
variable << number_of_bits
variable >> number_of_bits
```

Parameter

`variable` - an integer variable which has the amount of bits less than or equal to 32 bits (or a byte, int or long variable).

Example 3-19

```
int a = 5;        //equal to 0000000000000101 binary
int b = a << 3;   // the result is 0000000000101000 binary or 40
int c = b >> 3;   // the result is 0000000000000101 binary or 5 decimal
```

Example 3-20

When instructing to slide variable x to the left following the amount of y bit (x << y), the bit data on the far left of x with the amount of y will be disappeared because it is moved to the left hand.

```
int a = 5;        //equal to 0000000000000101 binary
int b = a << 14;  // the result is 0100000000000000 binary
```

Moving bits to the left will affect the value of the variable on the left of the operator is multiplied by two and power the number of bits that shift to the left as follows.

When you instruct to slide variable x to the right with the amount of y bit (x >> y), the difference depends on types of variables. If x is an int variable, the sum can appear both positive and negative. Wherewith, the bit on the far left will be sign bit and if it is a negative, the bit on the far left will be 1. After instructing to move a bit to the right, the program will bring the value of sign bit to add up to the bit on the far left. This phenomenon is called sign extension as the example below.

Example 3-21

```
int x = -16     // equal to 1111111111110000 binary
int y = x >> 3  // shift bits of x variable to the right for 3 times
                // the result is 1111111111111110 binary
```

If you would like to slide the bit to the right and then put 0 at the bit on the far left (which happens with a case that a variable is unsigned int), you can do by changing types of temporary variables (typecast) in order to convert variable x to unsigned int temporarily as the following example.

Example 3-22

```
int x = -16                  // Equal to 1111111111110000 binary
int y = unsigned (x) >> 3    // Shift bits of x variable (not concerned
                             // about the sign) to the right for 3 times
                             // The result is 0001111111111110 binary
```

After being aware of the sign extension, you will use the operator to move bits to the right for dividing the variable value by 2 and power any number. For example

Example 3-23

```
int x = 1000;
int y = x >> 3;    // Divide the value of 1000 by 8, from 2^3
                   // The result is y = 125
```

3.7 Syntax of Arduino

3.7.1 ; (semicolon)

Used to end a statement

Example 3-24

```
int a = 13;
```

Forgetting to end a statement line in a semicolon will result in the complier error. Wherewith, a complier may complain that it cannot find the semicolon or notify as other mistakes. In case that a line which is noticed some mistakes but cannot be found any mistake so you should check a previous line.

3.7.2 { } (Curly-braces)

Curly braces are a major part of C programming language and used to determine operation in each period. An opening brace {must always be followed by a closing brace} or it is said that parentheses must be balanced. In Arduino IDE software that is used for programming will have an ability to check the balance of curly braces so users just click at a parenthesis and its logical companion will be highlighted.

For beginning programmers and programmers coming to C language from BASIC language might be confused with the application of brackets. In fact, curly braces can be compared with the RETURN statement in subroutine (function) or replace the ENDIF statement in comparison, and the NEXT statements in a FOR loop.

Because the use of the curly braces is so varied, it is good that you should type a closing bracket immediately after typing the opening bracket. Next, press ENTER button in between two braces to start a new line and write a desired statement. If you can follow this suggestion, your braces will never become unbalanced.

Unbalanced braces can make mistakes while a programme is compiled. If a programme is large, it is hard to track down any mistake. Location of each bracket influences to the syntax of a program. Therefore, moving a brace one or two lines will affect the meaning of a program.

The main use of curly braces

Functions

```
void myfunction(datatype argument)
{
    statements(s)
}
```

Loops

```
while (boolean expression)
{
   statement(s)
}
do
{
   statement(s)
} while (boolean expression);
for (initialisation; termination condition; incrementing expr)
{
   statement(s)
}
```

Conditional statements

```
if (boolean expression)
{
   statement(s)
}
else if (boolean expression)
{
   statement(s)
}
else
{
   statement(s)
}
```

3.7.3 // (single line comment) and /*…* (multi-line comment)

Comments are part of a program and a programmer will write more information to inform how the program operates. They are ignored by complier, and not exported to the processor, so they are very useful to investigate the program later or inform colleagues or others what this line is used for. There two kinds of comments in C lanague, including

(1) Single line comments write 2 slashes (//) in front of a line.

(2) Multi-line comments write a slash (/) paired with asterism to cover text of comments, such as `/*blabla*/`.

3.7.4 # define

This statement is used a lot in determination of constants for a program. Defined constants don't take up any memory space of microcontroller. When it is at the compile time, the compiler will replace characters with the assigned value. Arduino will use the same # define statement as C language does.

Syntax

`#define constantName value` (Note : the # is necessary)

Example 3-25

```
#define ledPin 3
// Determination to let variable ledPin equal to constant 3
```

There is no semicolon after the # define statement.

3.7.5 # include

It instructs to gather other files and our program file and then compile the program later.

Syntax

```
#include <file>
#include "file"
```

Example 3-26

```
#include <stdio.h>
#include "ATX2.h"
```

The first line will take file stdio.th to fuse with a developing program by searching the file from a location where stores the file system of Arduino. Normally, it is a standard file come up with Arduino.

The second line instructs to gather file ATX2.h and a developing program by looking for address of C language files. Generally, they are files users build.

3.8 Variables

Variables are several characters determined in a program and used to store various informative values, such as readable values from a sensor connected with analog port pins of Arduino. There are many types of variables.

3.8.1 char : character type variable

The variable takes up 1 byte (8 bits) and it is provided for storing character values. Characters in C language will be given inside single quotes, like this, 'A' (for text, which composed of many characters together, will be written in double quotes, such as "ABD"). You can do arithmetic on characters, in case that you will apply ASCII values of the characters, e.g. 'A'+1 has the value of 66 because the ASCII code value of character A is equal to 65.

Syntax

```
char sign = ' ';
```

Example 3-27

```
char var = 'x';
```

var is a name of a character variable you desired.

x is a desired value to assign to that variable and here is one letter.

3.8.2 byte : a variable of numeric type 8 bit or 1 byte

A byte variable is used to store a numeric value with the size of 8 bits and the value can be between 0 to 255.

Example 3-28

```
byte b = B10010;
// Show the value of b in the form of a binary number
// (equal to 18 of a decimal number)
```

3.8.3 int : a variable of integer type

It is abbreviated from integer, which means an integer number. The int is a basic variable for preserving numbers. One variable has the size of 2 bytes and stores a value from -32,768 to 32,767, from -215 (minimum value) and 215-1 (maximum value). In storage of negative numbers uses a technique called 2's complement and the maximum bit sometimes called sign bit. If a value is '1', it is shown the value is negative.

Syntax

```
int var = val;
```

Parameters

`var` - a name of a desired variable of int type.

`val` - a desired value to define to a variable.

Example 3-29

```
int ledPin = 13;  // Assign variable ledPin to have a value = 13
```

When a variable is greater than a maximum capacity, the value will roll over to a minimum capacity, however, when a value is less than a minimum capacity, the value will roll over to a maximum capacity as the following example.

Example 3-30

```
int x
x = -32,768;
x = x - 1;          // When you follow the instruction,
                    // the value of x will change -32,768 into 32,767.
x = 32,767;
x = x + 1;          // When the statement is done,
                    // the value of x will change from 32,767 to -32,768
```

3.8.4 unsigned int : a variable of unsigned integer type

This type variable is similar to an int variable but will store only positive integers by storing the value 0 to 65,535 (216-1).

Syntax

```
unsigned int var = val;
```

Parameters

`var` - the name of the desired int variable.

`val` - the desired value to assign to a variable.

Example 3-31

```
unsigned int ledPin = 13;
// Determine ledPin variable to have the value equal to 13 of unsigned type
```

When a variable has a maximum capacity, it will roll over back to a minimum capacity later. Additionally, when a variable has a minimum capacity, it will roll over and become a maximum capacity when there is value reduction again. As the example

Example 3-32

```
unsigned int x
x = 0;
x = x - 1;      // After executed, the x value change from 0 to 65535.
x = x + 1;      // After executed, the x value change from 65535 into 0
```

3.8.5 long : a variable of 32-bit integer type

The variable stores integer values and extends its capacity from an int variable. A long variable uses a memory space of 32 bits (4 bytes) and is capable of storing values from -2,147,483,648 through 2,147,483,647.

Syntax

```
long var = val;
```

Parameters

var - the long variable name.

val - the value that assign to the variable.

Example 3-33

```
long time;  // Specify a time variable to be the long type
```

3.8.6 unsigned long : a variable of unsigned 32-bit integer type

A variable stores positive integers. One variable uses a memory space of 32 bits (4 bytes) and stores a value in range of 0 to 4,294,967,295 or 232-1.

Syntax

```
unsigned long var = val;
```

Parameter

var - the name of unsigned long variable.

val - the value that assign to the variable.

Example 3-34

```
unsigned long time;   // Determine a time variable to be unsigned long type
```

3.8.7 float : float-point variable

Float is a variable for storing a value of decimal number values. It is popular to be used to keep an analog signal value or a continuous value. Because this variable gives values more precise than an int variable does, a float variable can store in a range of 4.4028235 x 10^{38} through -4.4028235 x 10^{38} and one variable will cover 32-bit memory space (4 bytes).

In a mathematical calculation with the float variable will be slower than a calculation of int variable so you should avoid calculating with float variables when doing loop statement with the highest speed of a time function because it must be very precise. Some programmers will convert decimal numbers to integer numbers before a calculation for faster operation.

Syntax

```
float var = val;
```

Parameters

`var` - the float variable name

`val` - the value that determine to the variable

Example 3-35

```
float myfloat;
float sensorCalbrate = 1.117;
```

Example 3-36

```
int x;
int y;
float z;
x = 1;
y = x / 2;              // y is equal to 0 no storage of values of
                        // remainder from division
z = (float)x / 2.0;     // z is equal to 0.5
```

When there is a usage of a float variable, numbers that are operated with this float variable will be decimals as well. From the example is that number 2 calculated with float variable x, so the number 2 is written as 2.0.

3.8.8 double : Double precision floating-point variable

Double has the size of 8 bytes and the highest stored value is 1.7976931348623157 x 10^{38} . In Arduino, there is limited capacity so it doesn’t use this type of variables.

3.8.9 string : variable of text type

String is a variable that stores text and it is usually array of a char variable in C language.

Example 3-37 : Example of declaration of string variables

```
char Str1[15];
char Str2[8] = {'a','r','d','u','i','n','o'};
char Str3[8] = {'a','r','d','u','i','n','o','\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

- *Str1 is a declaration of a string variable in which a default value is not defined.*

- *Str2 announces a string variable along with defining a value of each character for a text. If it is not completed following the announced number, a complier will add null string until completing (from the example, 8 characters are declared but the text has only 7 characters so a complier fill one null string).*

- *Str3 states a string variable together with determining a value to give a text and close the end with a closing character which is\0.*

- *Str4 declares a string variable together with a determining a variable value in quotes. From the example, it doesn't assign the size of a variable, so a complier will assign the size according to a number of letters.*

- *Str5 proclaims a string variable along with specifying a variable value in a quotation mark and the size of the variable, from the example, is 8 letters.*

- *Str6 reveals a string variable and defines the size reserving for another text that is longer than this.*

3.8.9.1 Addition of characters which inform null termination

A string variable in C language specifies that the last character is an indicator of a null string. The indicator is 0. In determination of size of variables (value in square brackets), the size is equal to the amount of characters +1, as shown in the variable Str2 and Str3 in the example 4-37. In the text of ***Arduino***, there are 7 characters, declaration must specify as [8].

In announcement of a string variable, you should keep a space for storing a character, which notifies the null termination. Otherwise, a complier will warn a mistake happening. In the example, variable Str1 and Str6 can store messages with a maximum of 14 characters.

3.8.9.2 Single and double quotes

Normally, a string variable is determined inside quotes, such as "ABC". For a char variable, it is defined inside single quotation marks 'A'.

3.8.10 Array variable

An array is a variable containing multiple variables which are stored in variables of the same name.

Each variable is referred by an index number written inside square brackets. An array variable of Arduino will be cited according to C language. It may look complicated but if using an array variable directly, it is easy to understand.

Example 3-38 : Example of declaration of array

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = “hello”;
```

● *A program developer is able to declare an array without determination of a* `myInts` *variable.*

● *Announcement of variable* `myPins` *is a declaration of an array variable without specifying its size.*

● *In declaration of an array, a program developer can announce and determine a size of an array in the same time, as the example of declaration of variable* `mySensVals` *to proclaim a size and specify values.*

● *The last example is the declaration of a* `message` *variable, which is a char variable, and there are 5 characters, including hello. However, determination of a variable size will have to preserve a space for a character that informs null termination, so the index value is specified as 6.*

3.8.10.1 Usability of array variables

The usability of arrays can be done by typing a variable name together with specifying an index value inside square brackets. An index value of an array variable starts with value 0, so the value of a mySensVals variable is as follows.

```
mySensVals[0] == 2, mySensVals[1] == 4 ,
```

Determination of a value for an array variable can do as follows

```
mySensVals[0] = 10;
```

Calling a member value of an array value is done as this example

```
x = mySensVals[4];
```

3.8.10.2 Arrays and for loop instructions

Generally, application of an array variable is found in a for statement. An index value of an array variable is given from using a value of a loop variable of the for statement as the following example.

Example 3-39

```
int i;
for (i = 0; i < 5; i = i + 1)
{
   Serial.println(myPins[i]);     // Showing a member value of an array
                                  // variable at a serial monitor window
}
```

*The completed program example of application of array variables are available in the example of KnightRider in the topic of Tutorials at the website **www. arduino.cc**.*

3.9 Scope of variables

Variables in C language used in Arduino have a specification called scope. This is different from BASIC language that all variables have the same status and the status is called global.

3.9.1 Local and global variables

Global variables are variables all functions in a program knowed by declaring the variables outside functions. Local variables are variables proclaimed inside braces of functions and known only in functions.

When programs are larger and more complex, usage of local variables is very useful because only functions can apply the variables. This helps to protect mistakes when a function modifies variable values used by other functions.

Example 3-40

```
int gPWMval;           // All functions can see this variable.
void setup()
{}
void loop()
{
   int i;          // Variable i will be seen and performed inside only
                   // a loop function.
   float f;        // Variable f will be seen and applied only inside a
                   // loop function.
}
```

3.9.2 Static variables

A static variable is a reserved word and used when declaring variables, which have scope of use only inside functions. It is different from a local variable that a local variable will be created and deleted every time to run functions. For a static variable, when a function finishes, it is still appeared (not deleted). This is a preservation of a variable value during function performs.

A declared variable as static will build and define a value at the first time to run a function.

3.10 Constants

Constants are a group of characters or text preconfigured, so a complier of Arduino will recognise these constants and it is not neccesary to notify or determine constants.

3.10.1 HIGH, LOW : used to assign logical values

In reading and writing values for digital port pins, two possible values are HIGH or LOW.

HIGH is determination of values for digital pins and has voltage equal to +5V. If you can read a value equal to +3V or greater, a microcontroller will read the value as LOW. A constant of LOW is “0” or compared to logic as false.

LOW, which is configuration of digital pins, has voltage equal to 0V. If you can read a value as +2V or less, a microcontroller will read the value as LOW and a constant of LOW is “0” or compared to logic as false.

3.10.2 INPUT, OUTPUT : Determination of direction of digital port pins

Digital port pins can serve for 2 types including being input and output:

When defined as INPUT means to assign that port pin as an input pin.

When specified as OUTPUT means to assign that port pin as an output pin.

Arduino-Note

Assignment of integer constants to be in various number bases of Arduino

An integer constant is a number you write in a program of Arduino directly, such as 123, and normally, these numbers are decimal numbers. If you would like to specify them into other number radix systems, you will have to use special marks to specify. For example

Base	**Example**
10 (decimal)	123
2 (binary)	B1111011
8 (octal)	0173
16 (hexadecimal)	0x7B

Decimal is a decimal number used in daily life

Example : 101 = 101. It is from $(1* 10^2) + (0 * 10^1) + (1 * 10^0) = 100 + 0 + 1 = 101$

Binary is a binary number in which a number in each position can be only 0 or 1.

Example : B101 = 5 in decimal. It is from $(1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 4 + 0 + 1 = 5$

Binary numbers can be used less than 8 bits (not greater than 1 byte) and have values from 0 to (B0) 255 (B11111111).

Octal is an octal number and a number in each position has a value from 0 to 7 only.

Example : 0101 = 65 in decimal. It is from $(1 * 8^2) + (0 * 8^1) + (1 * 8^0) = 64 + 0 +1 = 65$

Caution in configuration of constants is to not put 0 in the front, otherwise, a compiler will translate a meaning incorrectly that a number value is octal.

Hexadecimal (Hex) is a hexadecimal number. A number in each position is worth from 0 to 9 and character A is 10 and B is 11 until F which is equal to 15.

Example : 0x101 = 257 in decimal. It is from $(1 * 16^2) + (0 * 16^1) + (1 * 16^0) = 256 + 0 + 1 = 257$

3.11 Other operators related to variables

3.11.1 cast : changing kinds of variables temporarily

`Cast` is an operator that instructs changing of a type of variables into another type and controls calculation of a variable value to become a new kind.

Syntax

```
(type)variable
```

`type` is a type of any variables (such as int, float, long)

`variable` is any variable or constant

Example 3-41

```
int i;
float f;
f = 4.6;
i = (int) f;
```

In the change of a variable type from float into int, a remainder of an obtained value will be cut out, so (int) 4.6 becomes 4.

3.11.2 sizeof : notifying the size of variables

`Sizeof` is used to identify the amount of byte of variables you are interested and it can be both a normal variable and an array.

Syntax

written into two patterns as follows

```
sizeof(variable)
sizeof variable
```

Therefore; **`Variable`** is a normal variable or an array variable (int, float, long) user would like to know the size.

Example 3-42

Sizeof operators are very useful in management with array variables (including string variables).

The following example will type text out through a serial port, each character a time.

```
char myStr[] = “this is a test”;
int i;
void setup()
{
   Serial.begin(9600);
}
void loop()
{
   for (i = 0; i < sizeof(myStr) - 1; i++)
   {
      Serial.print(i, DEC);
      Serial.print(“ = “);
      Serial.println(myStr[i], BYTE);
   }
}
```

3.12 Reserved words of Arduino

A reserved word is a constant, a variable, and a function defined as a part of of C language of Arduino. Don’t take these words to denominate variables. They are shown as follows :

# Constants

```
HIGH
LOW
INPUT
OUTPUT
SERIAL
DISPLAY
PI
HALF_PI
TWO_PI
LSBFIRST
MSBFIRST
CHANGE
FALLING
RISING
false
true
null
```

# Literal Constants

```
GLCD_RED
GLCD_GREEN
GLCD_BLUE
GLCD_YELLOW
GLCD_BLACK
GLCD_WHITE
GLCD_SKY
GLCD_MAGENTA
```

# Port Constants

```
DDRB
PINB
PORTB
DDRC
PINC
PORTC
DDRD
PIND
PORTD
```

# Names

```
popxt
ATX2
```

# Datatypes

```
boolean
byte
char
class
default
do
double
int
long
private
protected
public
return
short
signed
static
switch
throw
try
unsigned
void
```

# Methods/Fucntions

```
sw_ok
sw_ok_press
analog
knob
glcd
glcdChar
glcdString
glcdMode
glcdGetMode
glcdFlip
glcdGetFlip
colorRGB
setTextColor
setTextBackgroundColor
setTextSize
getTextColor
getTextBackgroundColor
getTextSize
glcdFillScreen
glcdClear
glcdPixel
glcdRect
glcdFillRect
glcdLine
glcdCircle
```

```
glcdFillCircle
glcdArc
getdist
in
out
motor
motor_stop
fd
bk
fd2
bk2
tl
tr
sl
sr
servo
sound
beep
uart_set_baud
uart_get_baud
uart_putc
uart_puts
uart
uart_available
uart_getkey
uart1_set_baud
uart1_get_baud
uart1_putc
uart1_puts
uart1
uart1_available
uart1_getkey
uart1_flush
```

```
# Other
abs
acos
+=
+
[]
asin
=
atan
atan2
&
&=
|
|=
boolean
byte
case
ceil
char
char
class
,
//
?:
constrain
cos
{}
-
default
delay
delayMicroseconds
/
/**
.
else
==
exp
false
float
```

```
float
floor
for
<
<=
HALF_PI
if
++
!=
int
<<
<
<=
log
&&
!
||
^
^=
loop
max
millis
min
-
%
/*
*
new
null
()
PII
return
>>
;
Serial
Setup
sin
sq
sqrt
```

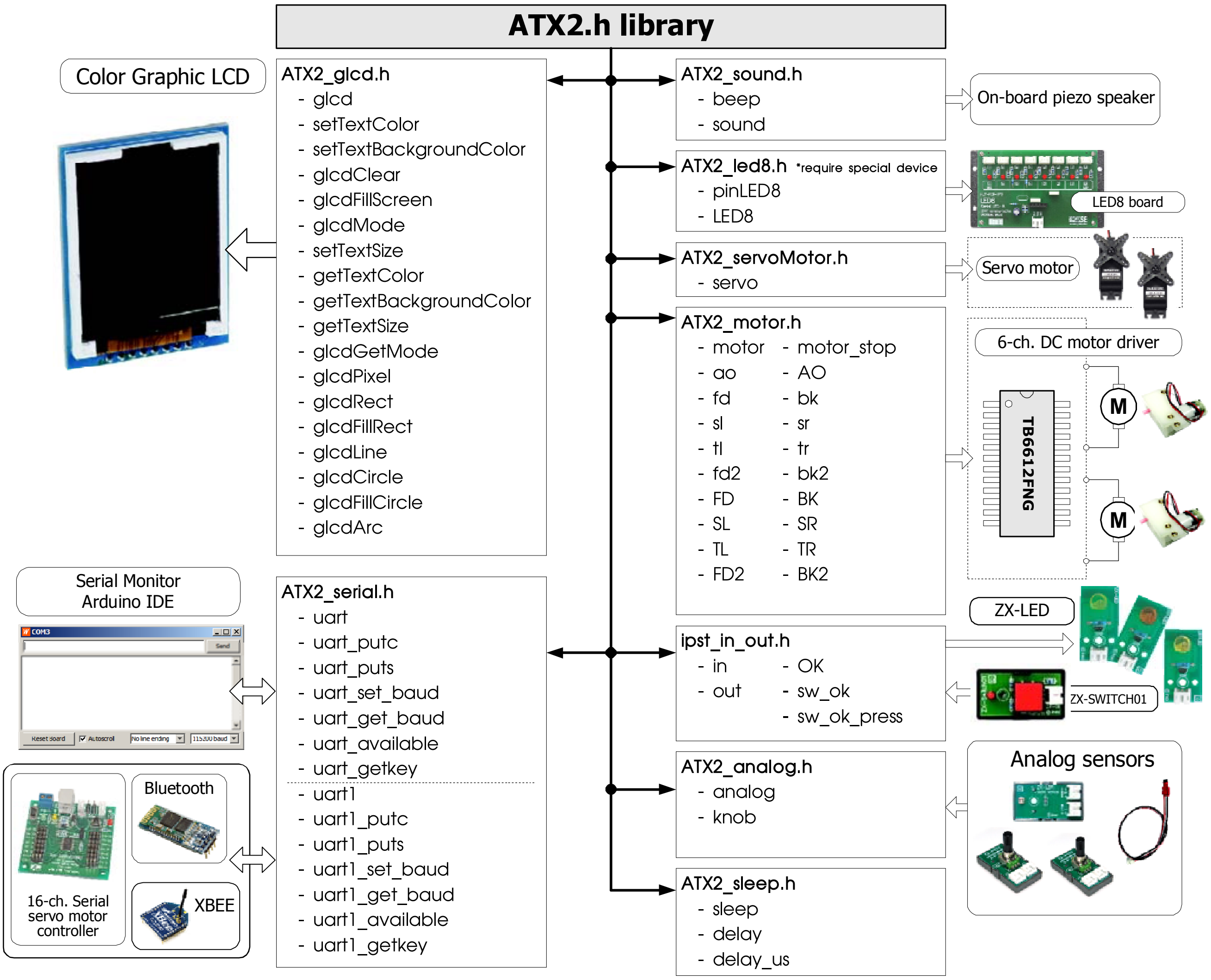
```
=
switch
tan
this
true
TWO_PI
void
while
Serial
begin
read
print
write
println
available
digitalWrite
digitalRead
pinMode
analogRead
analogWrite
attachInterrupts
detachInterrupts
beginSerial
serialWrite
serialRead
serialAvailable
printString
printInteger
printByte
printHex
printOctal
printBinary
printNewline
pulseIn
shiftOut
OK
```

Chapter 4

ATX2 library

The development of C/C++ language programming for the ATX2 controller board is performed under the support of library file; ATX2.h. It helps to reduce the complexity of programming to control the hardware.

The sturcture of the ATX2.h library file is shown below.



ATX2.h library

ATX2_glcd.h
- glcd
- setTextColor
- setTextBackgroundColor
- glcdClear
- glcdFillScreen
- glcdMode
- setTextSize
- getTextColor
- getTextBackgroundColor
- getTextSize
- glcdGetMode
- glcdPixel
- glcdRect
- glcdFillRect
- glcdLine
- glcdCircle
- glcdFillCircle
- glcdArc

ATX2_serial.h
- uart
- uart_putc
- uart_puts
- uart_set_baud
- uart_get_baud
- uart_available
- uart_getkey
- uart1
- uart1_putc
- uart1_puts
- uart1_set_baud
- uart1_get_baud
- uart1_available
- uart1_getkey

ATX2_sound.h
- beep
- sound

ATX2_led8.h *require special device
- pinLED8
- LED8

ATX2_servoMotor.h
- servo

ATX2_motor.h

- motor	- motor_stop
- ao	- AO
- fd	- bk
- sl	- sr
- tl	- tr
- fd2	- bk2
- FD	- BK
- SL	- SR
- TL	- TR
- FD2	- BK2

ipst_in_out.h

- in	- OK
- out	- sw_ok
	- sw_ok_press

ATX2_analog.h
- analog
- knob

ATX2_sleep.h
- sleep
- delay
- delay_us

4.1 ATX2.h library

To run the instructions for ATX2 and Robo-Creator2 program development; developers have to include ATX2.h mainly library file at the beginning of the C/C++ code with this command :

```
#include <ATX2.h>
```

to declare the compiler know all statements of the ipst.h library.

The ipst.h library consist of many sub-library. Includes :

● **ATX2_glcd.h** contains the functions and statements of the display message, number and line art graphic on the color graphic LCD of IPST-SE board (not support the image file).

● **ATX2_sleep.h** contains the function and statements of delayed time.

● **ATX2_in_out.h** contains the functions and statements of reading digital input port and sending digital data to digital output port.

● **ATX2_analog.h** contains the functions and statements of reading analog input. Also included KNOB button and OK switch value.

● **ATX2_sound.h** contains the functions and statements of sound generation.

● **ATX2_motor.h** contains the functions and statements of DC motor driving.

● **ATX2_servoMotor.h** contains the functions and statements of servo motor control. This library work with fixed microcontroller port.

● **ATX2_serial.h** contains the functions and statements of serial data communication via USB and TxD1/RxD1 of the IPST-SE controller board.

● **ATX2_led8.h** contains the functions and statements of interfacing with the LED8 board to control the LED operations.

● **ATX2_SPI.h** contains the functions and statements of interfacing with the SPI (Serial Peripheral Interface) devices.

● **ATX2_XIO.h** contains the functions and statements of interfacing with the on-board extension input/output controller.

● **ATX2_i2c.h** contains the functions and statements of interfacing with the 2-wire bus devices such as I2C bus.

● **ATX2_gp2d120.h** contains the functions and statements of interfacing with the GP2D120/GP2Y0A41 Infrared distance sensor.

4.2 Library description

This topic describes all sub-libraries of ATX2.h library file. Includes important functions and commands.

4.2.1 The delay time library

This library file has functions for time delaying. Important functions of this library file are consisted of :

4.2.1.1 sleep and delay

sleep and **delay** are same operation. They delayed time in millisecond unit.

Syntax

```
void sleep(unsigned int ms)
void delay(unsigned int ms)
```

Parameter

ms - Set the delay time in millsecond unit. Range is 0 to 65,535.

Example 4-1

```
sleep(20);          // Delay 20 millisecond
delay(1000);        // Delay 1 second
```

4.2.1.2 delay_us

It is delay time function in microsecond unit.

Syntax

```
void delay_us(unsigned int us)
```

Parameter

us - Set the delay time in microsecond unit. Range is 0 to 65,535.

Example 4-2

```
delay_us(100);      // Delay 100 microsecond
```

4.2.2 Sound library

This library file has functions for sound generator to drive a small piezo speaker. Important functions of this library file are consisted of :

4.2.2.1 beep

It is "beep" signal generated function forthe piezo speaker on the ATX2 controller board. Beep signal frequency should be select 4 frequencies (250, 506, 762 and 1,274Hz) with duration time 100 millisecond. ATX2 board will drive the sound signal to on-board piezo speaker.

Syntax

```
void beep(int beepType)
```

Parameter

`beepType` - beep frequency

0 or () : 1,274Hz

1 : 762Hz

2 : 506JHz

3 : 250Hz

Example 4-3

```
beep(1); // Generate the signal 762Hz with 100ms to  ATX2 speaker.
```

4.2.2.2 sound

It is sound generated function for the piezo speaker on the ATX2 controller board. This function sets the frequency, duration time and output pin. ATX2 board will drive the sound signal to on-board piezo speaker.

Syntax

```
void sound(int freq, int time)
```

Parameter

`freq` - Frequency value. It is 0 to 32,767 in Hz unit.

`time` - Duration time. It is 0 to 32,767 in millisecond unit.

Example 4-4

```
sound(1200,500);
// Generate the signal 1200Hz with 500ms to ATX2 on-speaker
```

4.2.3 Digital input/output port library

This library file has functions for reading and sending the digital value to any digital input/output port of ATX2 controller board.

Important functions of this library file are consisted of :

4.2.3.1 in

Read data from the specific digital port

Syntax

```
char  in(x)
```

Parameter

x - Digital pin of ATX2 board. It is 0 to 31. Pin 2, 3, 8, 9, 18, 24 to 31 are recommended.

Return value

0 or 1

Example 4-5

```
char x;        // Declare x variable for keeping reading input data
x = in(18);    // Read pin 18 and store data to x variable.
```

4.2.3.2 out

Write or send the data to the specific digital port

Syntax

```
out(char  _bit,char  _dat)
```

Parameter

_bit - Set digital pin of ATX2 board. It is 0 to 31. Pin 2, 3, 8, 9, 18, 24 to 31 are recommended.

_dat - Set data output as 0 of 1.

Example 4-6

```
out(24,1);    // Out oin 17 with "1"
out(25,0);    // Out pin 18 with "0"
```

4.2.4 Analog input library

This library file supports all instructions for reading the analog input port (A0 to A7), Extended analog input port (A8 to A12), KNOB button and OK switch of the ATX2 controller board.

Important functions of this library file are consisted of :

4.2.4.1 analog

This gets digital data from the analog to digital converter module of any analog port; A0 to A12 of the ATX2 controller board.

Syntax

```
unsigned int analog(unsigned char channel)
```

Parameter

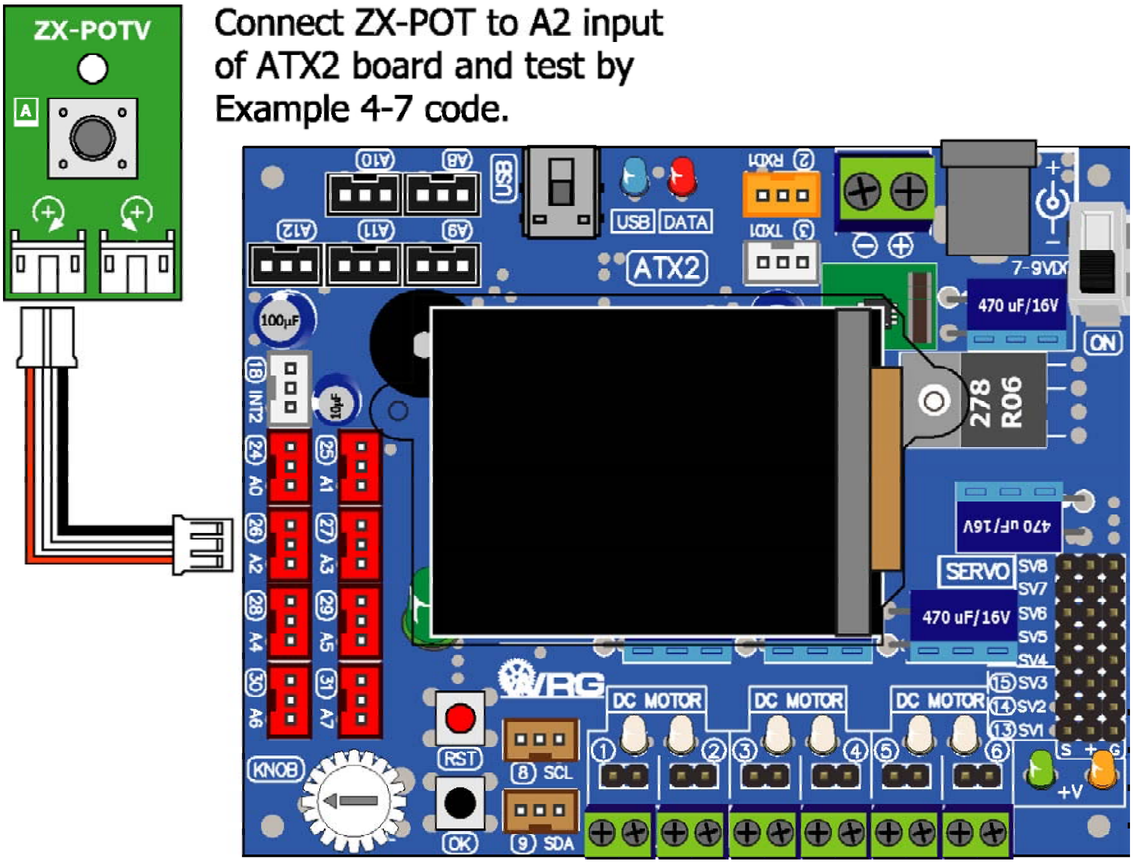
`channel` - Analog input port. It is 0 to 12 (means A0 to A12)

Return value

Digital data from analog to digital converter module. The value is 0 to 1,023 (in decimal). It represents 0 to +5Vdc.

Example 4-7

```
#include <ATX2.h>            // Include main library
int val=0;                   // Declare stored variable
void setup()
{
   OK();                     // Wait for OK
   glcdClear();
   setTextSize(2);           // Set text size as 2x
}
void loop()
{
   glcd(1,2,"AnalogTest");  // Show message on GLCD
   val  = analog(2);         // Read A2 input and store to val variable
   setTextSize(3);           // Set text size as 3x
   glcd(2,2,"%d   ",val);    // Show A2 value
   setTextSize(2);           // Set text size back to 2x
}
```



Connect ZX-POT to A2 input of ATX2 board and test by Example 4-7 code.

Illustration shows how to connect the ZX-POT (simple analog sensor) to the ATX2 controller board for analog function testing.

After uploading, ATX2 will read analog value from A2 input and shows the digital data that converted by A/D converter within microcontroller. The value is 0 to 1,023 from 10-bit A/D converter resolution.

4.3.4.2 knob( )

Read the **KNOB** button data of the ATX2 board.

Syntax

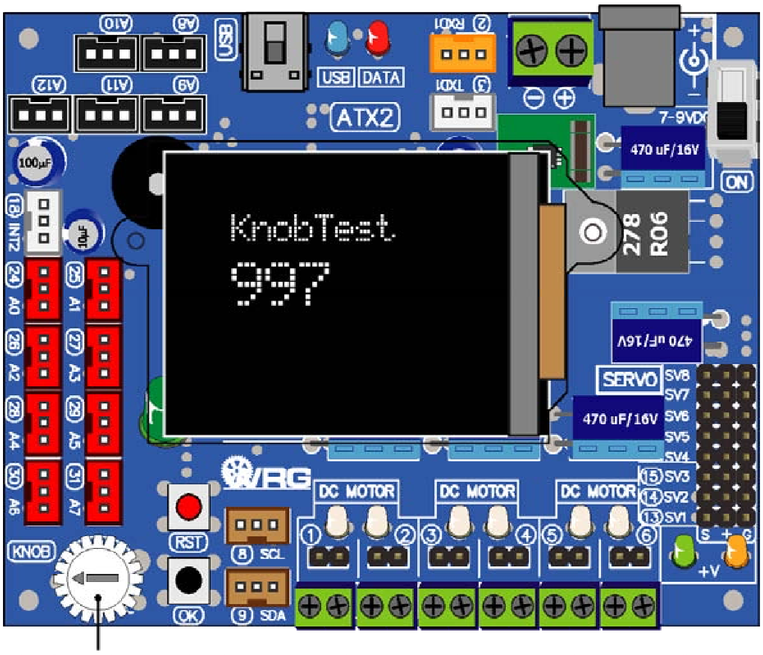
```
unsigned int knob()
```

Retuen value

Digital data from analog to digital converter module of **KNOB** button. It is 0 to 1,000

Example 4-8

```
#include <ATX2.h>                 // Include main library
void setup()
{
   OK();                          // Wait for OK button
   glcdClear();
   setTextSize(2);                // Set text size as 2x
}
void loop()
{
   glcd(1,2," KnobTest");         // Show message on GLCD
   setTextSize(3);                // Set text size as 3x
   glcd(2,2,"%d   ",knob());      // Show KNOB button value on GLCD
   setTextSize(2);                // Set text size back to 2x
}
```



Adjust KNOB button from left to right. Value is 0 to 1000.

4.3.4.3 sw_OK()

Read status of the **OK** button on the ATX2 controller board.

Syntax

```
unsigned char sw_ok()
```

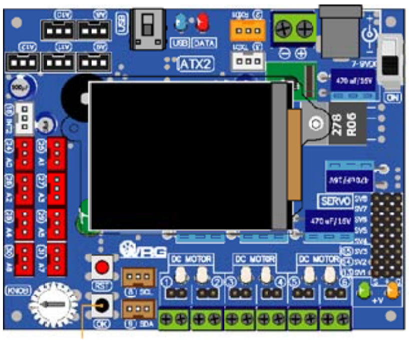
Retun value

1 (true) when the switch is pressed

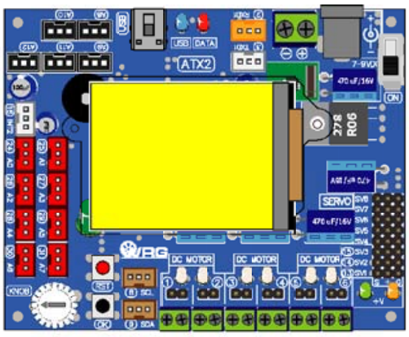
0 (false) no press the switch

Example 4-9

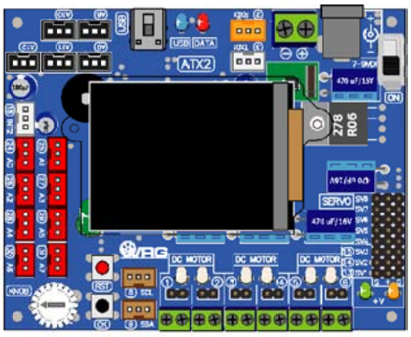
```
#include <ATX2.h>                    // Include main library
void setup()
{
   glcdClear();
}
void loop()
{
   if (sw_OK())                      // Check OK switch pressing
   {
      glcdFillScreen(GLCD_YELLOW);   // Change background to yellow
      delay(3000);                   // Show time 3 seconds
   }
   glcdClear();   // Clare screen and set backgoround color to black
}
```



Press OK button



After press OK button, ATX2 shows yellow background in 3 seconds.



Then shows black color again. Loop to wait for OK button pressing.....

4.3.4.4 sw_OK_press()

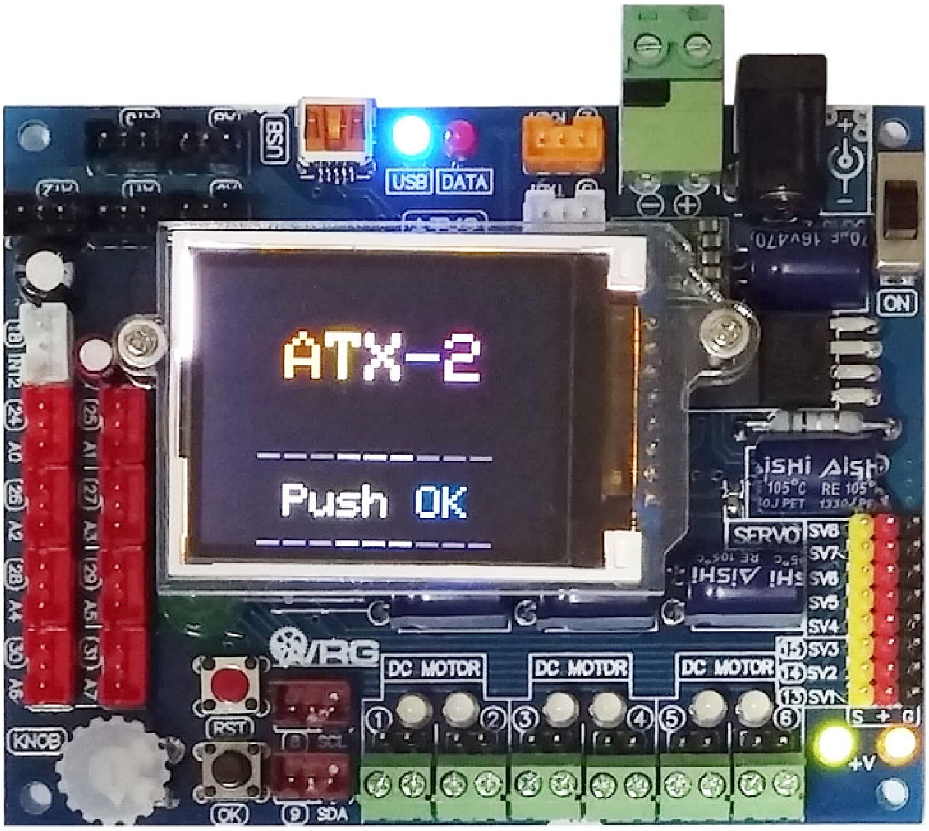
Loop to check the **OK** button pressing function

Example 4-10

```
............
sw_OK_press();       // Wait until the OK button is pressed
.............
```

4.3.4.5 OK()

Initialize the system thenloop to check the **OK** switch pressing and display the welcome message “ATX2   Push   OK”. Only OK characters are blinked.



Example 4-11

```
............
OK();              // Show welcome message and wait OK button is pressed
.............
```

4.3.5 DC motor control library

ATX2 board has 6 of DC motor driver outputs. It can drive 3 to 9V DC motor. Important functions of this library file are consisted of :

4.3.5.1 motor

Drive DC motor function.

Syntax

```
void motor(char _channel,int _power)
```

Parameter

_channel - DC motor output. Includes :

1 to 6 for each channel 1, 2, 3, 4, 5 or 6

12 for channel 1 and 2

34 for channel 3 and 4

56 for channel 5 and 6

100 or ALL or ALL4 for channel 1 to 4

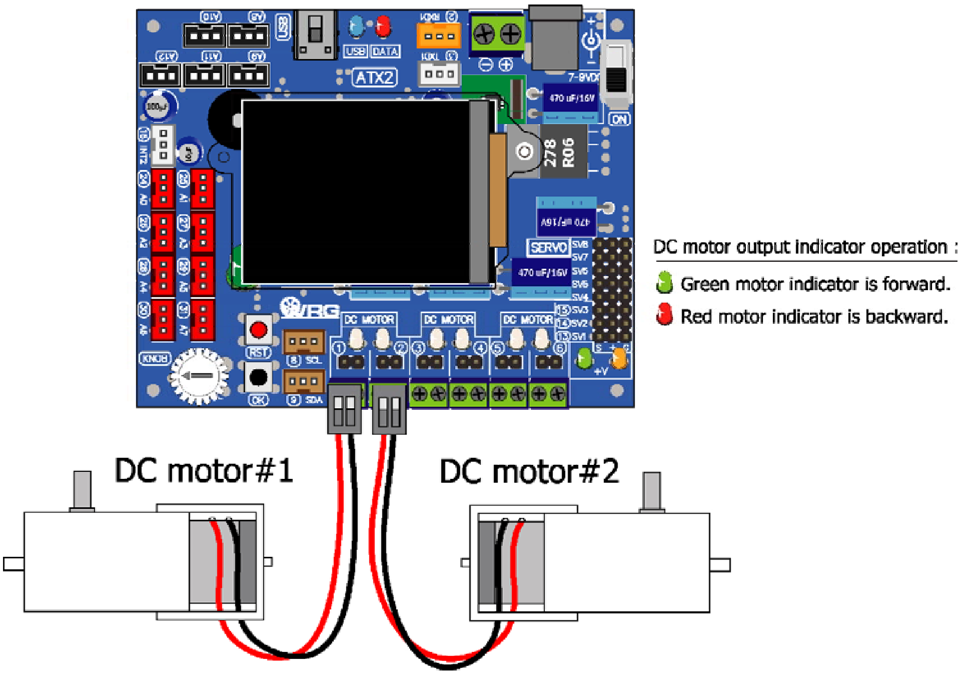
106 or ALL6 for all channels (1 to 6)

_power - Power value. It is -100 to 100

If set **_power** as positive value (1 to 100), motor moves forward

If set **_power** as negative value (-1 to -100), motor moves backward

If set as 0, motor stop but not recommended. Please choose the **motor_stop** function better.



DC motor output indicator operation :

● Green motor indicator is forward.

● Red motor indicator is backward.

Example 4-12

```
#include <ATX2.h>
void setup()
{
   OK();          // Wait for OK
}
void loop()
{
   motor(1,60);   // Drive motor 1 with 60% power
   delay(500);    // Driving time 0.5 second
   motor(1,-60);  // Drive motor 1 backward with 60% power
   delay(500);    // Driving time 0.5 second
}
```

5.4.6.2 motor_stop

Stop motor driving function

Syntax

`void motor_stop(char _channel)`

Parameter

`_channel` - DC motor output. Includes :

1 to 6 for each channel 1, 2, 3, 4, 5 or 6

12 for channel 1 and 2

34 for channel 3 and 4

56 for channel 5 and 6

100 or ALL or ALL4 for channel 1 to 4

106 or ALL6 for all channels (1 to 6)

Example 4-13

```
#include <ATX2.h>        // Include main library
void setup()
{
   OK();                 // Check the OK switch pressing
}
void loop()
{
   motor(1,60);          // Drive motor1 with 60% power
   delay(500);           // Driving time 0.5 second
   motor(1,-60);         // Drive motor1 backward with 60% power
   delay(500);           // Driving time 0.5 second
   if (sw_OK_press())    // Check the OK button pressing
   {
      motor_stop(1);     // If OK button is pressed, stop motor1
      while (1);
   }
}
```

Example 4-14

```
#include <ATX2.h>     // Include main library
void setup()
{
   OK();              // Check the OK switch pressing
}
void loop()
{
   motor(1,50);       // Drive motor1 with 50% power
   motor(2,50);       // Drive motor2 with 50% power
   sleep(3000);       // Driving time 3 seconds
   motor(1,-50);      // Drive motor1 backward with 50% power
   motor(2,-50);      // Drive motor 2 backward with 50% power
   sleep(3000);       // Driving time 3 seconds
   motor_stop(ALL);   // Stop both motors
   sleep(3000);       // Delay 3 seconds
}
```

4.3.6 Servo motor library

This is library for control the servo motor of ATX2 board. It has 8 servo motor outputs. It can drive each output or all in same time.

There is only one function in this library. It is **servo.**

Syntax

```
void servo(unsigned char _ch, int _angle)
```

Parameter

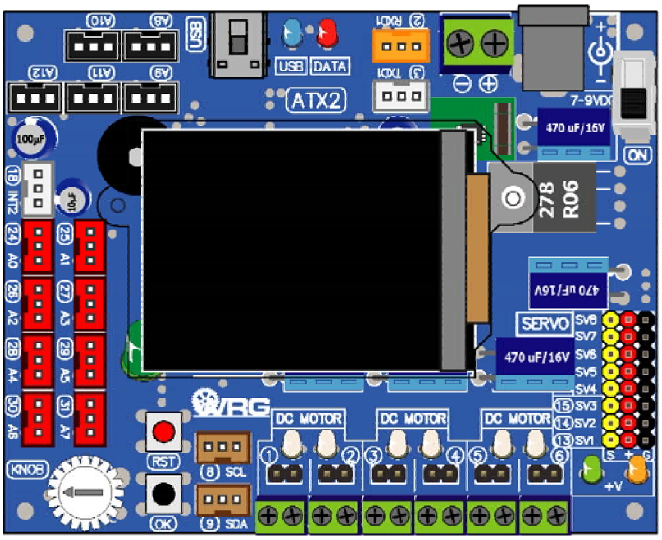
_ch - Servo motor output. It is 1 to 8

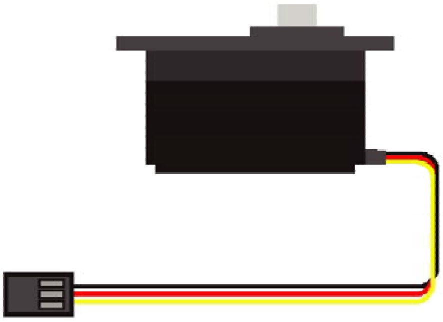
_pos - Servo motor shaft position. It is 0 to 180 and -1.

If set to -1, **it means stop that servo motor output**

Example 4-15

```
#include <ATX2.h>      // Include main library
void setup()
{
   OK();              // Check the OK switch pressing
}
void loop()
{
   servo(0,60);       // Drive servo motor0 to 60 degress position
   sleep(5000);       // Delay 5 seconds
   servo(0,120);      // Drive servo motor0 to 120 degree position
   sleep(5000);       // Delay 5 seconds
}
```





Connect servo motor to Servo output

Yellow/White wire to Yellow pin

Red wire to Red pin

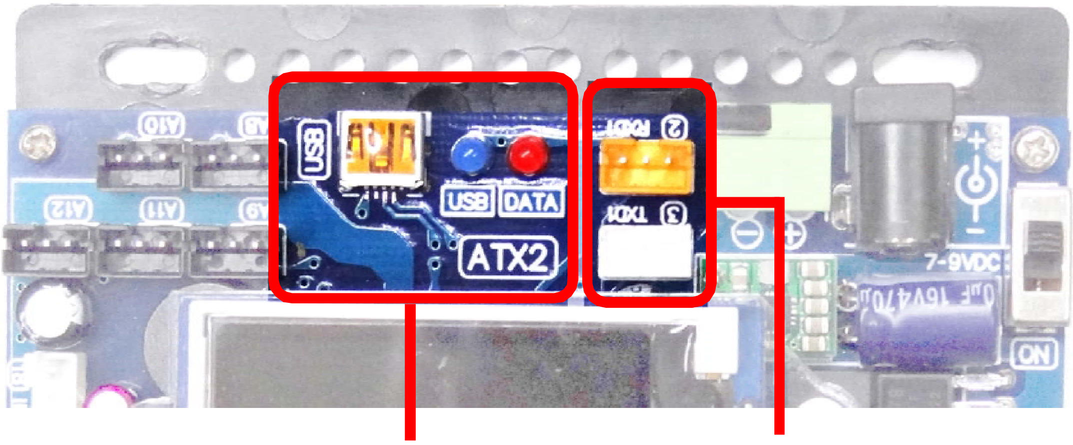
Black wire to Black pin

4.3.7 Serial data communication library

It is library that contains function and statement for supporting the serial data communication with UART of ATX2 board.

4.3.7.1 Hardware connection

ATX2 board has 2 channels for support serial data communication. They are UART0 and UART1. Hardware connection could be done as follows :



UART0 (connect via USB to UART circuit) **UART1 (TxD1/RxD1)**

UART0 (connectd with USB port via USB to UART circuit)

Connect USB cable between ATX2 board and computer's USB port.

UART1

Connect the cable to RXD1 (pin 2) and TXD1 (pin 3) on the ATX2 board

4.3.7.2 uart

This is serial data sending function via UART0 port. The default baudrate of this function is 9,600 bit per second.

Syntax

```
void uart(char *p,...)
```

Parameter

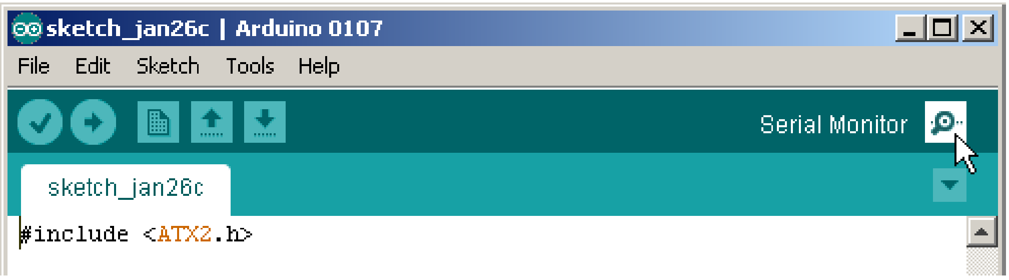
*p - Type of data. Support the special character for setting display method.

Command	*Operation*
%c or %C	Display 1 character
%d or %D	Display the decimal value -32,768 to +32,767
%l or %L	Display the decimal value -2,147,483,648 to +2,147,483,647
%f or %F	Display floating point 3 digits
\r	Set the message left justify of the line
\n	Display message on the new line

Example 4-16

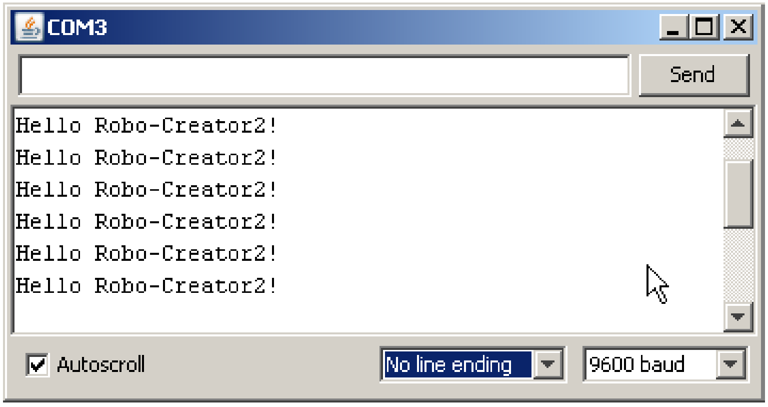
```
#include <ATX2.h> // Include main library
void setup()
{}
void loop()
{
    uart("Hello Robo-Creator2!\r\n");
    // Transmit data to computer with carriage return
    sleep(2000);      // Delay 2 seconds
}
```

For running this sketch, still connect the USB cable after uploading. Open the Serial Monitor by clicking on 🔍 button or choose at menu **Tools > Serial Monitor**



```
#include <ATX2.h>
```

Wait a moment, The Serial Monitor is appeared and shows the messages following the picture below. Click to mark at **Auto Scroll** box and select **No line ending**. Last box is selected to **9600 Baud.**



```
Hello Robo-Creator2!
Hello Robo-Creator2!
Hello Robo-Creator2!
Hello Robo-Creator2!
Hello Robo-Creator2!
Hello Robo-Creator2!
```

☑ Autoscroll   No line ending   9600 baud

4.3.7.3 uart_set_baud

Baud rate setting function of UART0

Syntax

```
void uart_set_baud(unsigned int baud)
```

Parameter

baud - Baud rate value of UART0. It is 2,400 to 115,200

Example 4-17

```
uart_set_baud(4800); // Set baud rate to 4,800 bit per second
```

4.3.7.4 uart_available

This is receiveing data testing function of UART0.

Syntax

```
unsigned char uart_available(void)
```

Return value

- "0" : no data received
- more than 0 : received character

Example 4-18

```
char x =uart_available();
    // Check receiving data of UART0.
    // If x is more than 0, it means now data is received
    // Must read data by using uart_getkey funtion immediately
```

4.3.7.5 uart_getkey

This is data reading function from receiver's buffer of UART0

Syntax

```
char  uart_getkey(void)
```

Return value

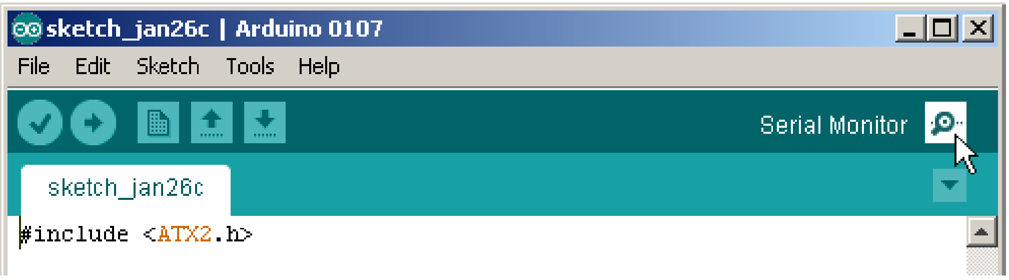
- "0" : no data received
- data : received character in ASCII code

Example 4-19

```
#include <ATX2.h>
void setup()
{
   glcdMode(1);                         // Set display rotation mode 1
   setTextSize(2);                      // Set text size 2x
}
void loop()                             // Main loop
{
   glcdClear();
   if(uart_available())                 // Received data checking
   {
      if(uart_getkey()=='a')            // Key a is pressed ?
      {
         glcd(3,1,"Key a Active!"); // Show message to response
         sleep(2000);                   // Delay 2 seconds
      }
      else
      {
         glcdClear();                   // Clear screen
      }
   }
}
```

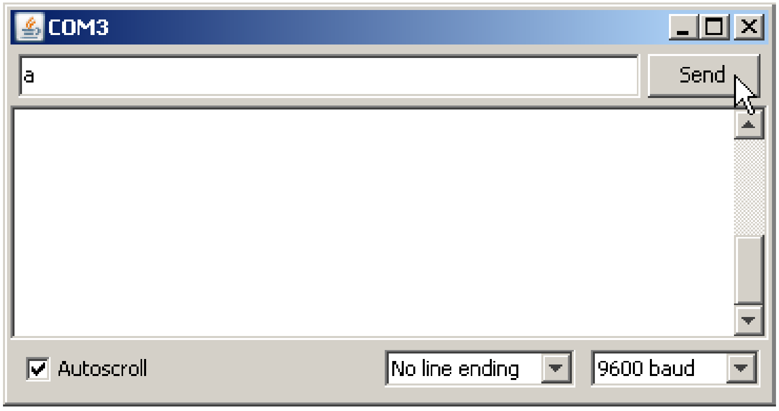
Note Using `uart()` function to send data out from UART0 and `uart_getkey()` function to get character, baud rate is set to 9,600 bit per second, 8-bit data and none parity checking automatically. It is default value. If requres to change baud rate, have to use uart_set_baud funtion.

For running this sketch, still connect the USB cable after uploading. Open the Serial Monitor by clicking on button or choose at menu **Tools > Serial Monitor**



```
#include <ATX2.h>
```

Wait a moment, The Serial Monitor is appeared and shows the messages following the picture below. Click to mark at **Auto Scroll** box and select **No line ending**. Last box is selected to **9600 Baud.**



COM3

a

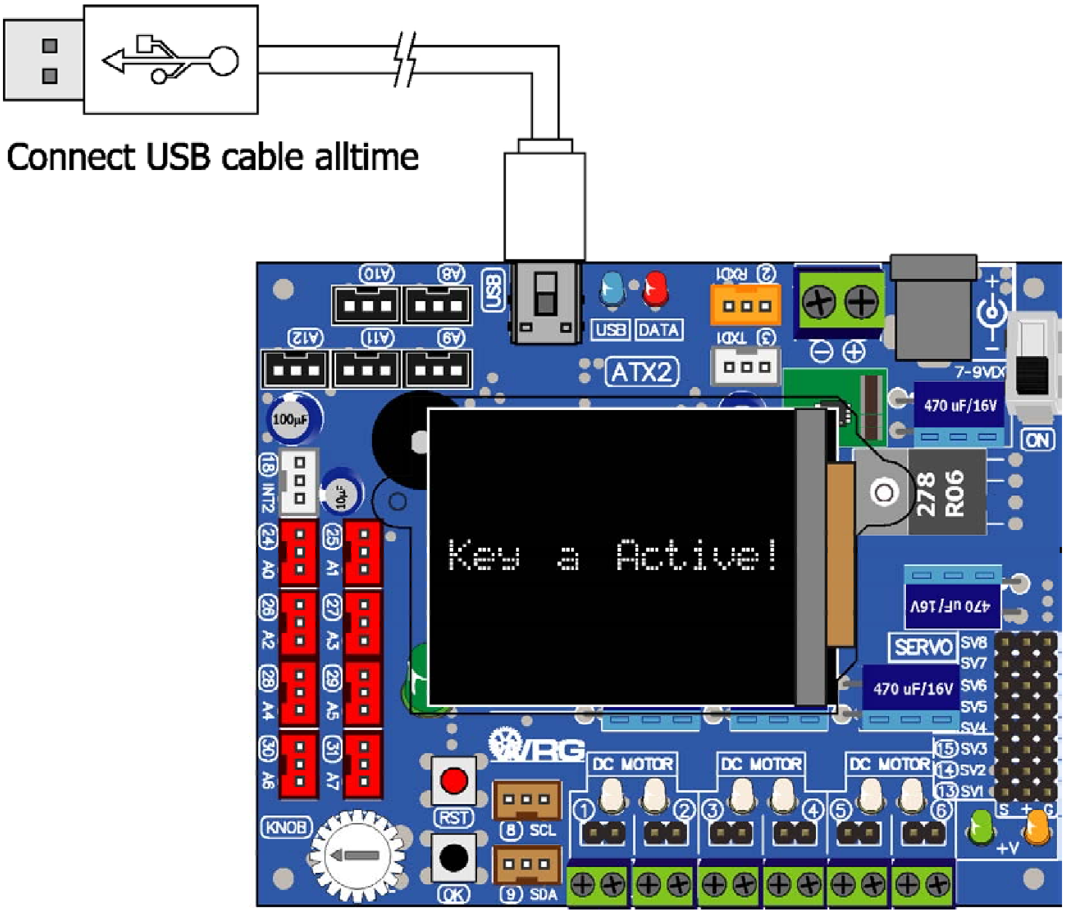
Send

☑ Autoscroll

No line ending

9600 baud

After ATX2 board gets the a character, it shows message Key a Active! *on th color GLCD screen.*



4.3.7.6 uart1

This is serial data sending function via UART1 port (TxD1 andc RxD1). The default baud rate is 9,600 bit per second.

Syntax

```
void uart1(char *p,...)
```

Parameter

`*p` - Type of data. Support the special character for setting display method. See details in **`uart0`** function.

4.3.7.7 uart1_set_baud

This is baud rate setting function for UART1.

Syntax

```
void uart1_set_baud(unsigned int baud)
```

Parameter

baud - Baud rate of UART1. It is 2,400 to 115,200

Example 4-20

```
uart1_set_baud(19200);     // Set baud rate as 19,200 bit per second
```

4.3.7.8 uart1_available

This is receiving data testing function of UART1.

Syntax

```
unsigned char uart1_available(void)
```

Return value

- "0" : no data received
- more than 0 : received character

Example 4-21

```
char x =uart1_available();
// Check receiving data of UART1.
// If x is more than 0, it means now data is received
// Must read data by using uart1_getkey funtion immediately
```

4.3.7.9 uart1_getkey

This is data reading function from receiver's buffer of UART1.

Syntax

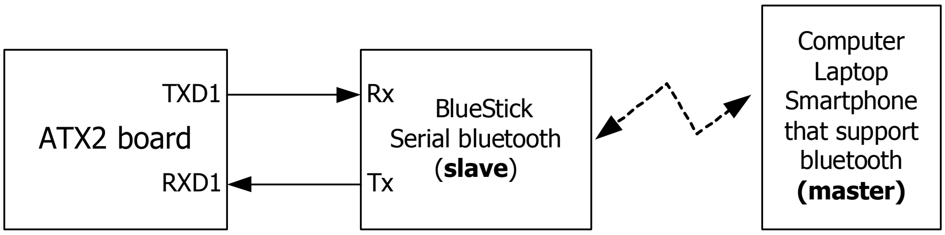
```
char uart1_getkey(void)
```

Return value

- "0" : no data received
- data : received character in ASCII code

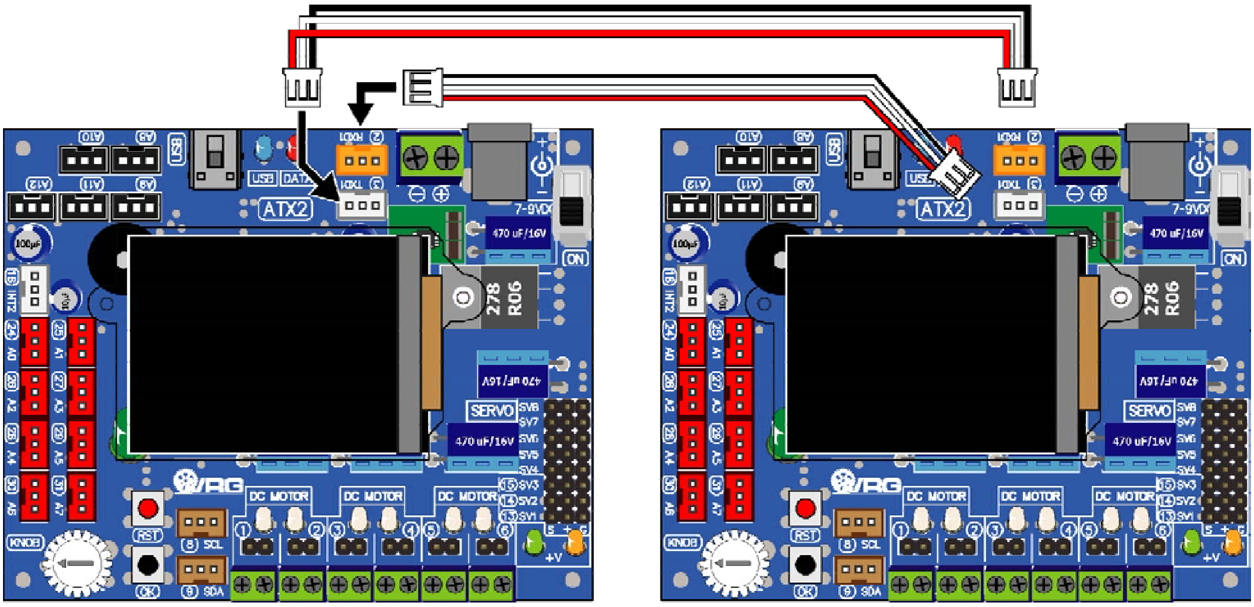
4.3.7.10 When does using UART1 ?

1. Connnect with wirelss devices such as bluetooth, XBEE, WiFi, etc.



Example of the connection diagram between ATX2 board with smartphone/tablet (Android) or computer in wirelessly via bluetooth.

2. Communicate between 2 of ATX2 board



Connect TxD1 (left board) to RxD1 (right board)
and connect RxD1 (left board) to TxD1 (right board)

3. Interface with any serial devices and modules such as ZX-SERVO16i : Serial servo motro controller board, Serial LCD16x2 (SLCD16x2), Serial Real-time clock (ZX-17), Pixy Camera module, RFID reader board, etc.

Chapter 5

ATX2 with GLCD activity

In the development of modern applied science projects that related to programming, must have automatic control systems involved. Including the need to have contact with any sensor. User need to know the value of the sensor measuring or detecting it. One of the most important devices is display device. It is used to display or report the value or data.

ATX2 controller board also available an on-board display device. It is color graphic LCD module. This chapter describes about features, library, how to interface and some of examples.

5.1 Features of GLCD module of the ATX2 board

- 1.8" size and 128 x 160 dots resolution
- Support line art and simple graphic with 65,536 color. Not support the image or photo file.
- Support the standard character (5 x 7 dots) with 21 characters 16 lines maximum.
- Built-in LED back light
- Programming support with ATX2_glcd.h library

5.2 Library file for GLCD executation

The suitable library file for worlking with GLCD module of ATX2 controller board will be installed together with Arduino IDE 1.0.7 installation. The library file ATX2_glcd.h is located in folder C:\Arduino\libraries\ATX2 and executed by ATX2.h main library.

This library must be included at the top of the program with the command #include as follows :

```
#include <ATX2.h>
```

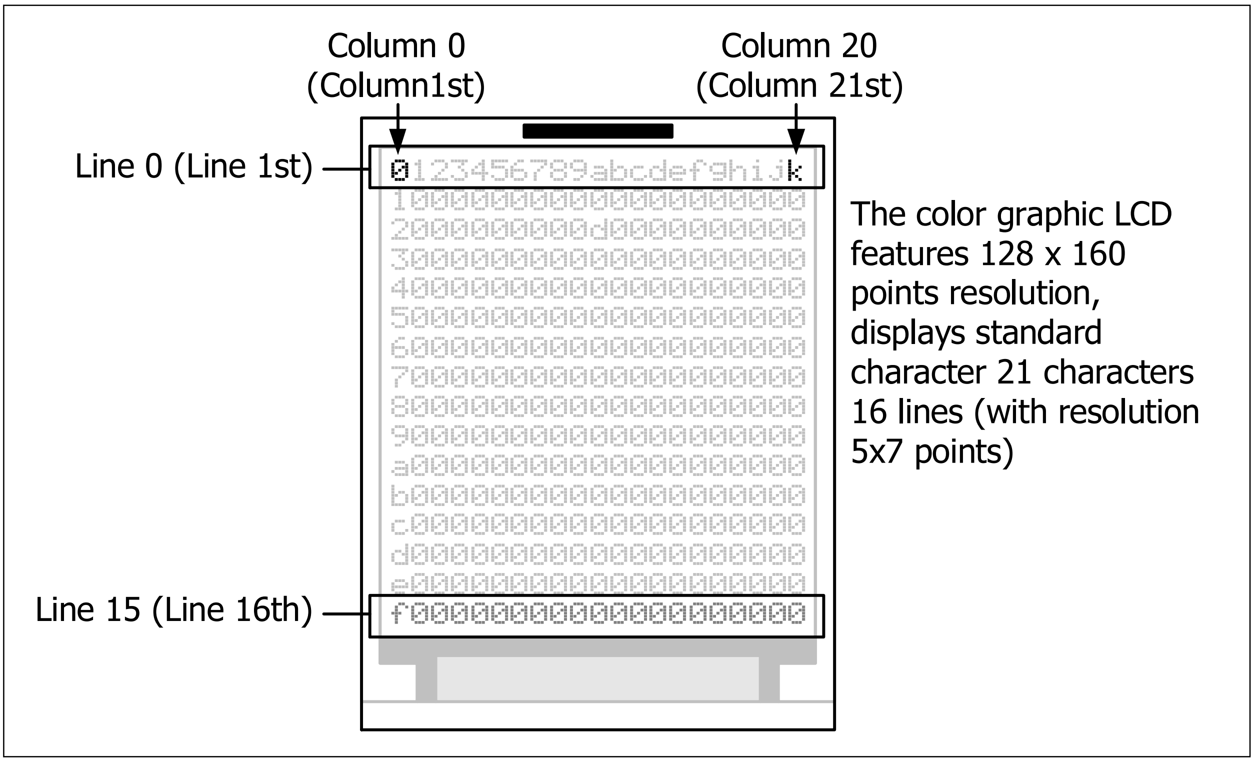

Column 0
(Column1st)
Column 20
(Column 21st)
Line 0 (Line 1st)
Line 15 (Line 16th)
0123456789abcdef9hijk
The color graphic LCD features 128 x 160 points resolution, displays standard character 21 characters 16 lines (with resolution 5x7 points)

Figure 5-1 : Illustated of details and displayed dot location of the GLCD module that used in ATX2 controller board

5.3 ATX2 board's color GLCD function description

5.3.1 glcd

It is the function for display message on the color graphic LCD screen. The default display is 21 characters, 16 lines with smallest (default) font size.

Syntax

```
void glcd(unsigned char x, unsigned char y ,char *p,...)
```

Parameter

x - line number. Value is 0 to 15

y - character position. Value is 0 to 20

***p** - the display message and special character or symbol for determine the display as follows :

%c or %C - display one character

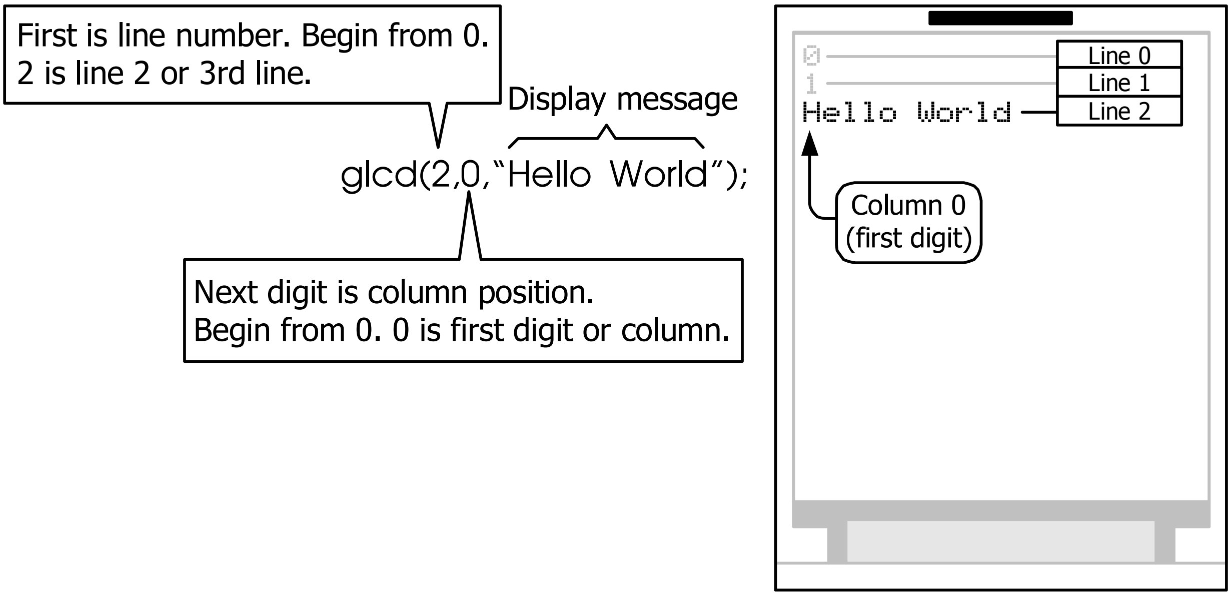
%d or %D - display integer from -32,768 to 32,767

%l or %L - display integer from -2,147,483,648 to 2,147,483,647

%f or %F - display floating point (3-digit maximum)

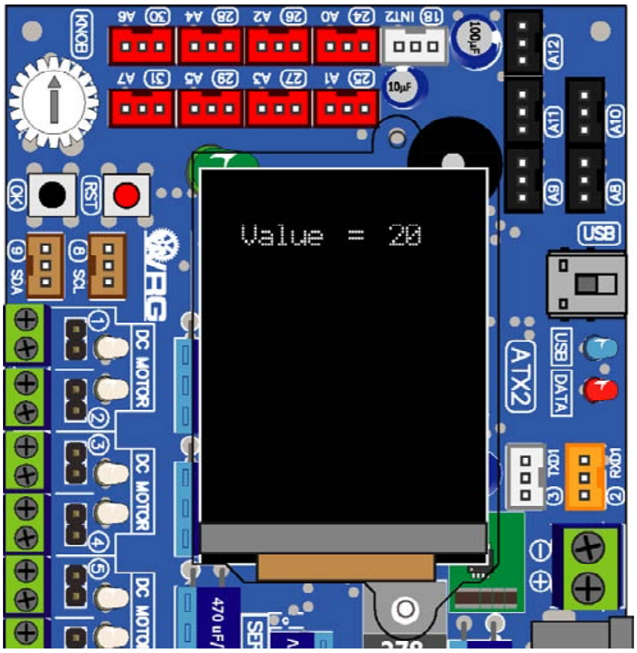
Example 5-1

```
glcd(2,0,"Hello World");
// Show message; Hello World at left end position of line 2
```



Example 5-2

```
int x=20;
glcd(1,2,"Value = %d",x);
// Display both charater and number same line
// Start from column 2 of line 1
```



5.3.2 colorRGB

It is color changing function in RGB (Red Blue Green) format to 16-bit data. It divides 5-bit for Red , 6-bit for Green and last 5-bit for Blue.

Syntax

```
unsigned int colorRGB(uint red,uint green,uint blue)
```

Parameter

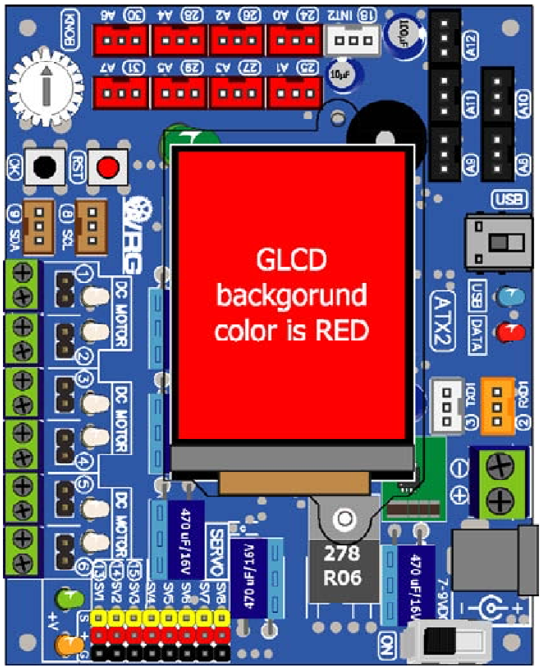
`red` - Red value is between 0 to 31. If applied data is greater than 31, adjsut to 31

`green` - Green value is between 0 to 63. If applied data is greater than 63, adjust to 63

`blue` - Blue value is between 0 to 31. If applied data is greater than 31, adjsut to 31

Example 5-3

```
#include <ATX2.h>
int colors;
void setup()
{
   int colors;
   colors=colorRGB(31,0,0);    // Set 16-bit data of red to colors variable
   glcdFillScreen(colors);     // Fill background color
}
void loop()
{}
```



5.3.3 color[ ]

It is array type variable. It is uesd for set the 18 colors. Developers can also use `color[]` directly or use the color name.

Syntax

```
unsigned int color[]= { GLCD_RED,
  GLCD_GREEN, GLCD_BLUE,
  GLCD_YELLOW, GLCD_BLACK,
  GLCD_WHITE, GLCD_SKY,
  GLCD_MAGENTA, GLCD_ORANGE,
  GLCD_LIME, GLCD_VIOLET
  GLCD_PINK, GLCD_DOLLAR,
  GLCD_BROWN, GLCD_DARKGREEN,
  GLCD_NAVY,GLCD_DARKGREY,
  GLCD_GREY};
```

Parameter

GLCD_RED - Select red

GLCD_GREEN - Select grren

GLCD_BLUE - Select blue

GLCD_YELLOW - Select yellow

GLCD_BLACK - Select black

GLCD_WHITE - Select white

GLCD_SKY- Select sky blue color

GLCD_MAGENTA - Select magenta

GLCD_ORANGE - Select orange

GLCD_LIME - Select lime green

GLCD_VIOLET - Select vilolet

GLCD_PINK - Select pinko

GLCD_DOLLAR - Select black

GLCD_BROWN - Select white

GLCD_DARKGREEN - Select dark green

GLCD_NAVY - Select magenta

GLCD_DARKGREY - Select dark grey color

GLCD_GREY - Select Grey

Example 5-4

```
glcdFillScreen(color[5]);     // Set background color as white
```

Example 5-5

```
glcdFillScreen(GLCD_BLUE);    // Set backgrounbd color as blue
```

5.3.4 glcdSetColorWordBGR

It is factory color data setting function in BGR type. The setting data consists of 5-bit of blue, 6-bit of green and 5-bit of red. There is 2 models of this color GLCD productions. One is set the color data as RGB. Another is RGB.

The default of ATX2_glcd.h is set to BGR. No need to put this function into the sketch.

Syntax

`glcdSetColorWordBGR()`

Example 5-6

```
#include <ATX2.h>
void setup()
{
   glcdSetColorWordBGR();
   // Set color data setting of GLCD to BGR type. Normally not require
}
void loop()
{}
```

5.3.5 glcdSetColorWordRGB( );

It is factory color data setting function in RGB type. The setting data consists of 5-bit of red, 6-bit of green and 5-bit of blue.

Need to put this function into the sketch if GLCD operation after uploading shows incorrect color. It means the current color GLCD is RGB type. Developer have to put this function in the **`setup()`** on the top of sketch.

Syntax

`glcdSetColorWordRGB()`

Example 5-7

```
#include <ATX2.h>
void setup()
{
   glcdSetColorWordBGR();   // Set color data setting to RGB type
}
void loop()
{}
```

5.3.6 setTextColor

This function is used to set the text color that displayed with **`glcd()`** function. The default color is white.

Syntax

```
void setTextColor(unsigned int newColor)
```

Parameter

`newColor` - This is to set the target color. It is 16-bit data or variable data which is defined from the variable **`color[]`**

Example 5-8

```
setTextColor(GLCD_YELLOW); // Set text color as yellow
```

5.3.7 setTextBackgroundColor

It is to set the text background color function. The default color is black. The text background color is not screen background. Setting the screen background color, need to use the **`glcdFillScreen`** function.

Syntax

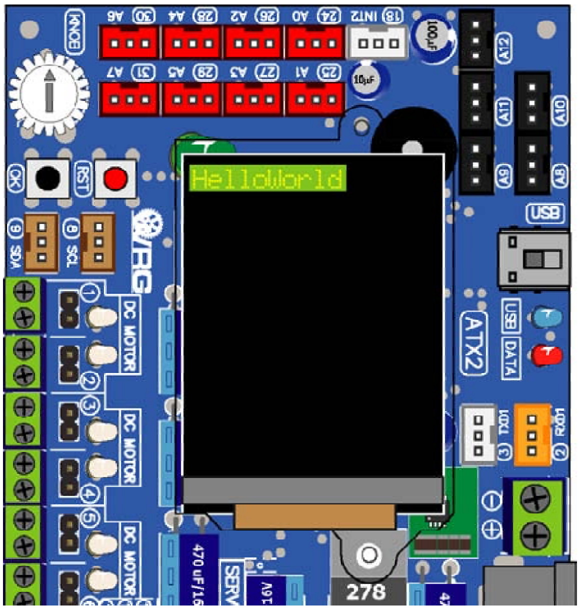
```
void setTextBackgroundColor(unsigned int newColor)
```

Parameter

`newColor` - This is to set the target color. It is 16-bit data or variable data which is defined from the variable **`color[]`**

Example 5-9

```
setTextBackgroundColor(GLCD_GREEN);
// Set text backgorund color as green
```



5.3.8 glcdClear

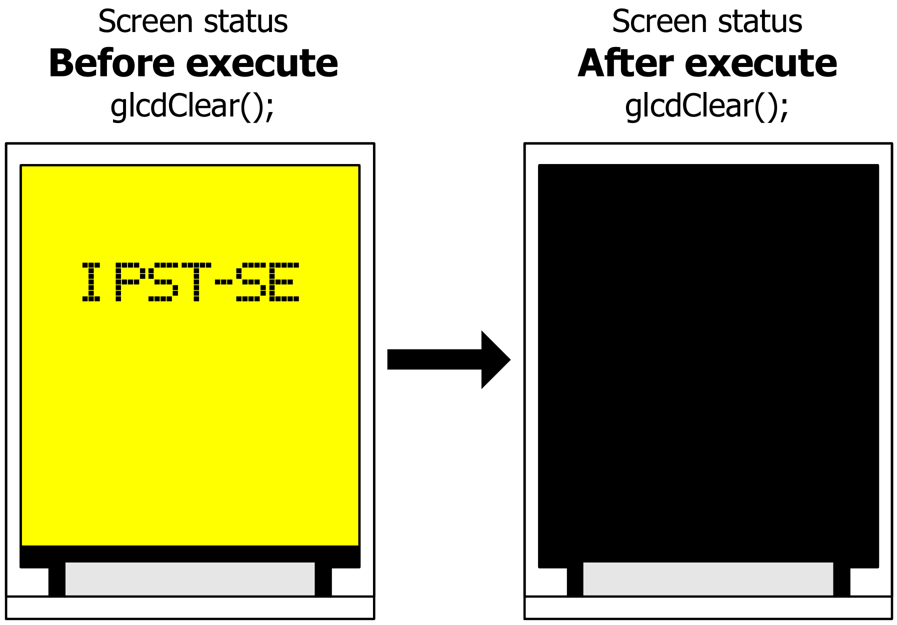
It is clear screen function. The background color will be latest the text backgorund color. If not defined before, the background color will be black

Syntax

```
void  glcdClear()
```

Example 5-10

```
glcdClear();    // Clear screen
```



5.3.9 glcdFillScreen

This will clear the screen and change to the background color function. After executing this function, all contents on scrren will be cleared and it will change the backgtround color to the target color.

Syntax

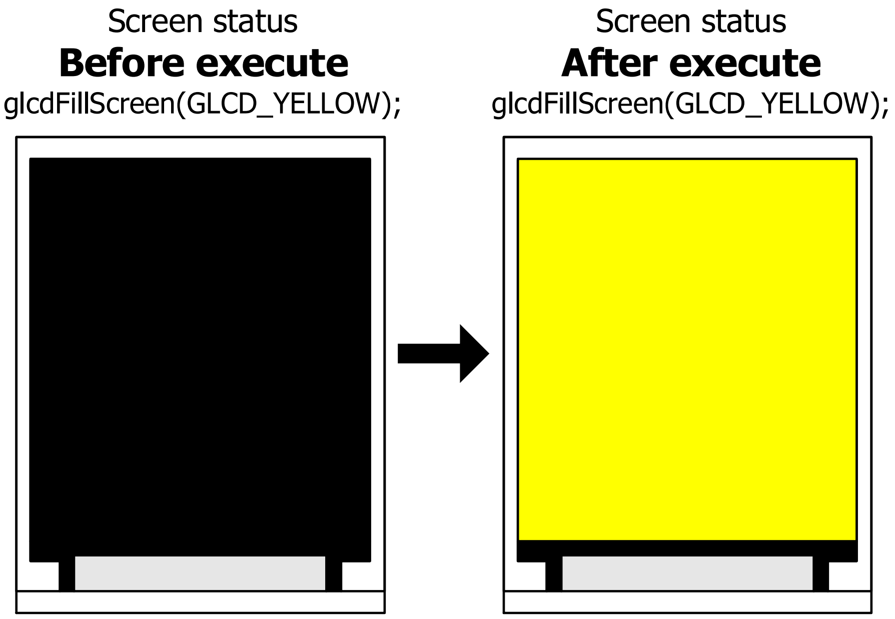
```
void glcdFillScreen(unsigned int color)
```

Parameter

color - This is to set the target color. It is 16-bit data or variable data which is defined from the variable **color[]**

Example 5-11

```
glcdFillScreen(GLCD_YELLOW);  // Fill GLCD screen with yellow
```



Screen status
Before execute
glcdFillScreen(GLCD_YELLOW);

Screen status
After execute
glcdFillScreen(GLCD_YELLOW);

5.3.10 glcdMode

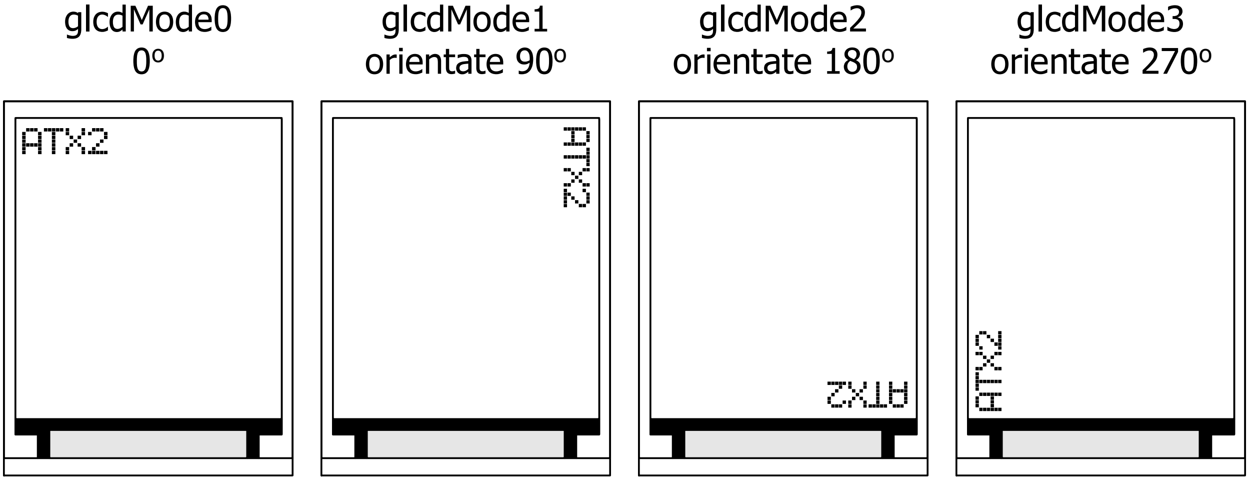
It is for setting the display orientation. There are 4 modes; 0 (0 degree), 1 (oritentate right 90 degrees), 2 (orientate 180 degrees or invert) and 3 (orientate 270 degrees from origin)

Syntax

```
glcdMode(unsigned int modeset)
```

Parameter

modeset - Orientation mode number. It is 0 to 3 for determine 0, 90, 180 and 270 degrees orientation. The default is 0 degree in vertical.



glcdMode0 0°

glcdMode1 orientate 90°

glcdMode2 orientate 180°

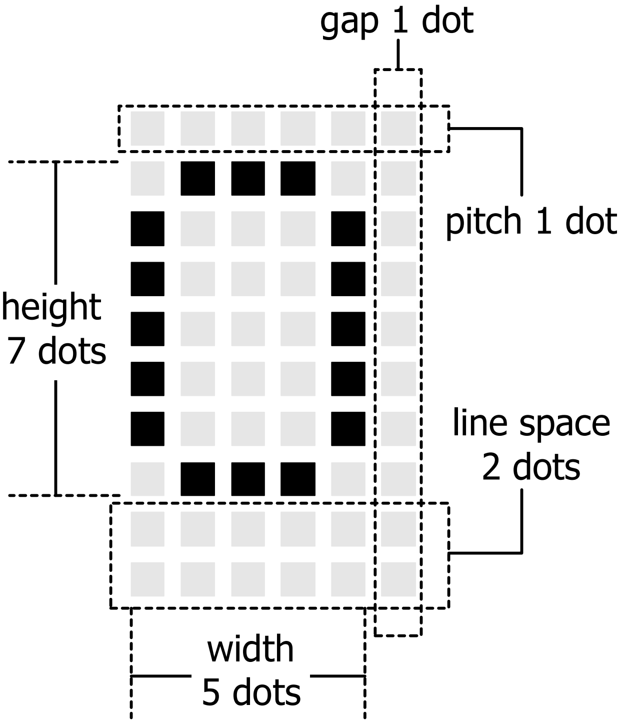
glcdMode3 orientate 270°

Example 5-12

```
#include <ATX2.h>
void setup()
{
   setTextSize(2);       // Set text size as 2x
}
void loop()
{
   glcdClear();          // Clear screen
   glcdMode(0);          // Set display orientation mode 0
   glcd(0,0,"ATX2");     // Show message
   sw_ok_press();        // Wait OK switch pressing
   glcdClear();
   glcdMode(1);          // Set display orientation mode 1
   glcd(0,0,"ATX2");
   sw_ok_press();
   glcdClear();
   glcdMode(2);          // Set display orientation mode 2
   glcd(0,0,"ATX2");
   sw_ok_press();
   glcdClear();
   glcdMode(3);          // Set display orientation mode 3
   glcd(0,0,"ATX2");
   sw_ok_press();
}
```

5.3.11 setTextSize

This function is used to set the text size. The text size is 1x time by default. It requires 6 x 10 dots include character gap. With the default size, this display shows 21 characters 16 lines maximum in vertical.



Syntax

```
setTextSize(unsigned int newSize)
```

Parameter

`newSize` - times number of the default size. It is 1 to 16.

text size 1 contains 21 characters 16 lines

text size 2 contains 10 characters 8 lines

text size 3 contains 7 characters 5 lines

text size 4 contains 5 characters 4 lines

text size 5 contains 4 characters 3 lines

text size 6 and 7 contains 3 characters 2 lines

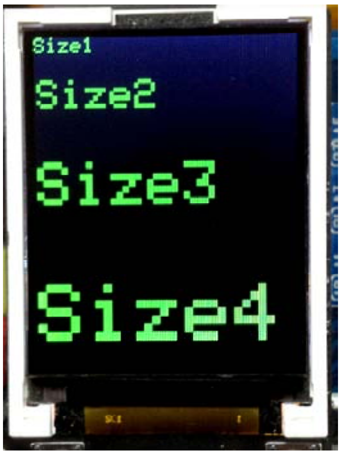
text size 8 contains 2 characters 2 lines

text size 9 and 10 contains 2 characters 1 line

text size 11 to 16 contains only 1 character and 1 line to fit in screen.

Example 5-13

```
#include <ATX2.h>
void setup()
{
   setTextSize(1);              // Set text size as default
   setTextColor(GLCD_GREEN);    // Set text color as green
   glcd(0,0,"Size1");           // Show message
   setTextSize(2);
   glcd(1,0,"Size2");           // Set text size 2x
   setTextSize(3);
   glcd(2,0,"Size3");           // Set text size 3x
   setTextSize(4);
   glcd(3,0,"Size4");           // Set text size 4x
}
void loop()
{}
```



5.3.12 getTextColor

Get the current text color.

Syntax

```
unsigned int getTextColor()
```

Return value

`textColor` - It is 16-bit data. It refer **`colorRGB[]`** function

Example 5-14

```
   unsigned int color;
   color=getTextColor();    // Store current text color data to variable
```

5.3.13 getTextBackgroundColor

Get the current text background color.

Syntax

`unsigned int getTextBackgroundColor()`

Return value

`textColor` - It is 16-bit data. It refer **`colorRGB[]`** function

Example 5-15

```
unsigned int color;
color=getTextBackgroundColor();
// Store current text background color data to variable
```

5.3.14 getTextSize

Get the current text size.

Syntax

`unsigned int getTextSize()`

Return value

`textSize` - The size is times number from the default size. Range is 1 to 16.

Example 5-16

```
unsigned int textSize;
textSize=getTextSize(); // Store current text size value to variable
```

5.3.15 glcdGetMode

Get the current orientation in display mode.

Syntax

`unsigned int glcdGetMode()`

Return value

`mode` - The orientation display mode number. It is 0 to 3. See details in **`glcdMode`** function

Example 5-17

```
unsigned int Mode;
Mode=glcdGetMode();
// Get the current orientation display mode number
```

5.3.16 glcdPixel

This plots the dots on the coordinator of the display. It refers to 128 x 160 dots display.

Syntax

```
void glcdPixel(unsigned int x,unsigned int y,unsigned
   int color)
```

Parameter

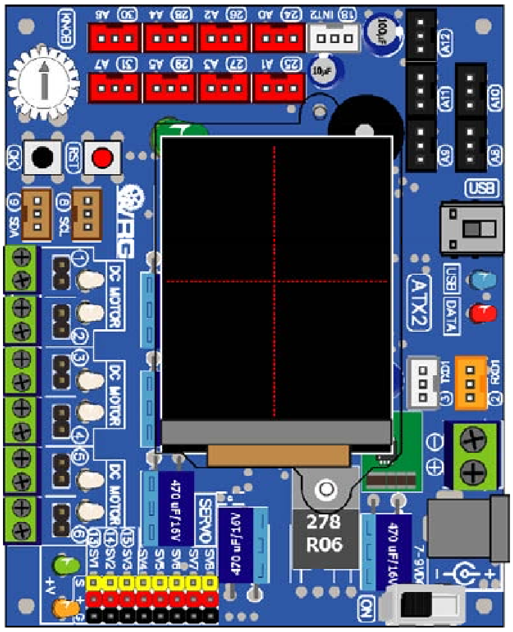
`x` - Horizontal axis coordinator or x-axis. Value is 0 to 127.

`y` - Vertical axis coordinator or y-axis. Value is 0 to 159

`color` - The target color. It is 16-bit data or variable data which is defined from the variable **`color[]`**

Example 5-18

```
#include <ipst.h>
int i;
void setup()
{
   for (i=0;i<128;i+=4)
   {
      glcdPixel(i,80,GLCD_RED);
      // Plot a dot every 4 dots on x-axis at the center of screen
   }
   for (i=0;i<160;i+=4)
   {
      glcdPixel(64,i,GLCD_RED);
      // Plot a dot every 4 dots
      // on y-axis at the center of
      // screen
   }
}
void loop()
{}
```



6.3.1.17 glcdRect

Draw the rectangular shape that refer dot coordinator at 128 x 160 dots resolution of color GLCD screen following the figure 6-2.

Syntax

```
void glcdRect(unsigned int x1,unsigned int y1,
   unsigned int width,unsigned int height,
   unsigned int color)
```

Parameter

x1 - Start point of the rectangular shape on x-axis. Value is 0 to 127

y1 - Start point of the rectangular shape on y-axis. Value is 0 to 159

width - The width of rectangular shape. Value is 1 to 128

height - The height of rectangular shape. Value is 1 to 158

color - Line color. It is 16-bit data or variable data which is defined from the variable **color[]**

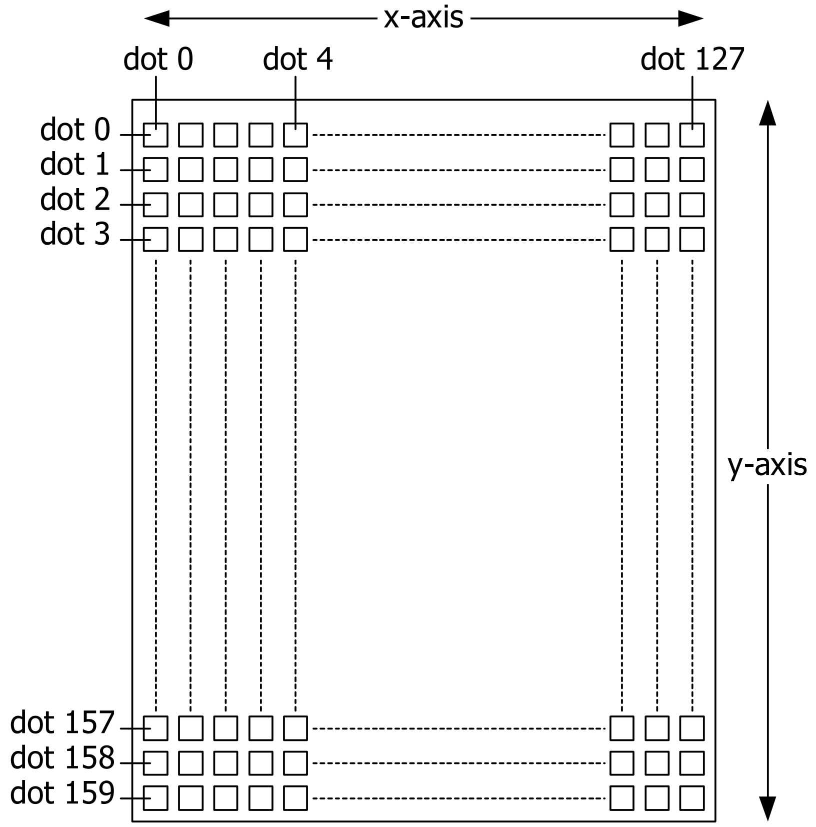
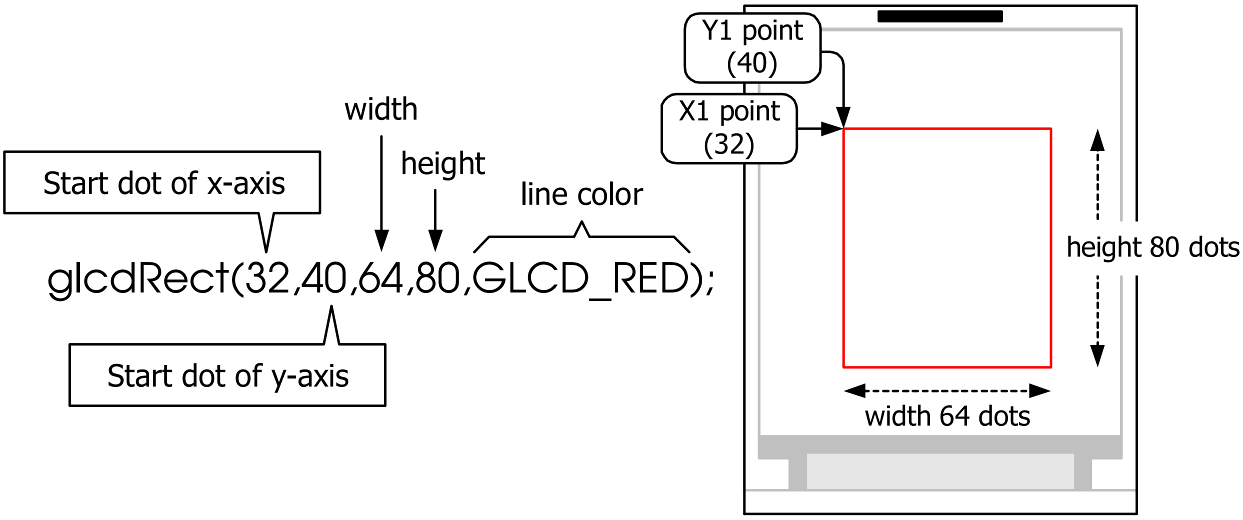


Figure 5-2 : Dot coordinator of ATX2 board's color display

Example 5-19

```
#include <ATX2.h>
void setup()
{
   glcdRect(32,40,64,80,GLCD_RED);
   // Draw the red rectangle with 64 x 80 dots size
}
void loop()
{}
```



5.3.18 glcdFillRect

This creates a filled rectangle. It is only fill color without an outline.

Syntax

```
void glcdFillRect(unsigned int x1, unsigned int y1,
   unsigned int width, unsigned int height,
   unsigned int color)
```

Parameter

x1 - Start point of the rectangular shape on x-axis. Value is 0 to 127

y1 - Start point of the rectangular shape on y-axis. Value is 0 to 159

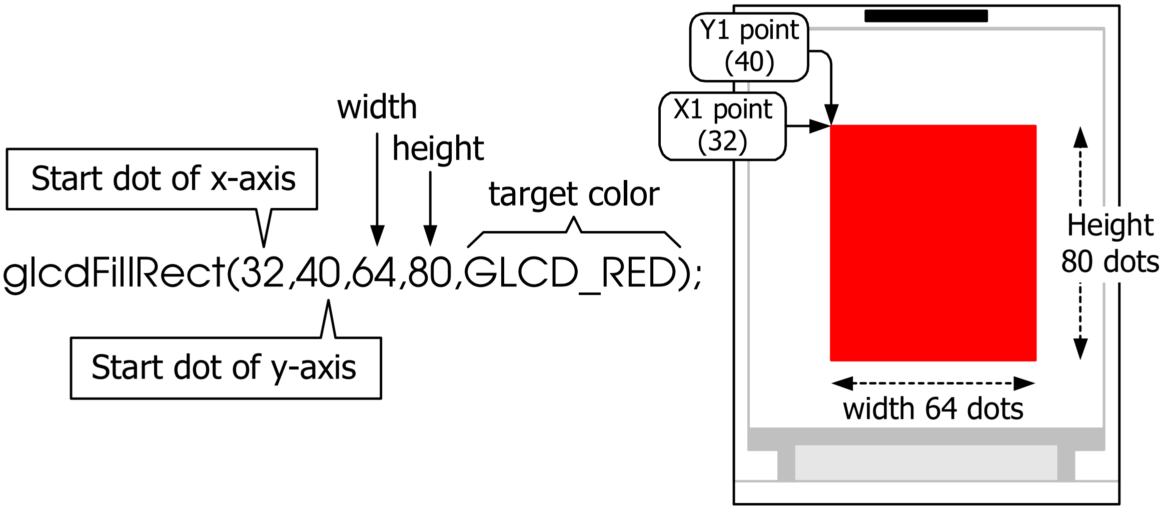
width - The width of rectangular shape. Value is 1 to 128

height - The height of rectangular shape. Value is 1 to 158

color - Fill color. It is 16-bit data or variable data which is defined from the variable **color[]**

Example 5-20

```
#include <ATX2.h>
void setup()
{
   glcdFillRect(32,40,64,80,GLCD_RED);
   // Create the solid red rectangle 64 x 80 pixels
}
void loop()
{}
```



5.3.19 glcdLine

Draw the straight line from point to point.

Syntax

```
void glcdLine(unsigned int x1,unsigned int y1,
  unsigned int x2,unsigned int y2,unsigned int color)
```

Parameter

x1 - Start point on the x-axis. Value is 0 to 127.

y1 - Start point on the y-axis. Value is 0 ro 159

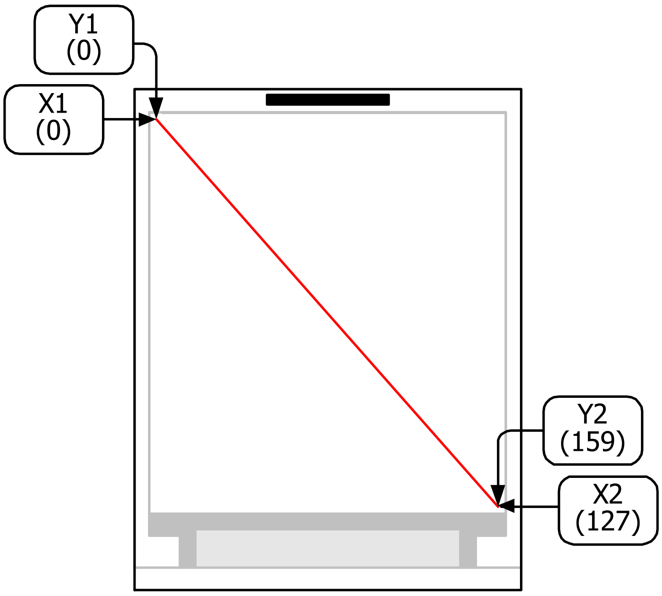
x2 - Destination point on the x-axis. Value is 0 to 127.

y2 - Destination point on the y-axis. Value is 0 ro 159

color - Line color. It is 16-bit data or variable data which is defined from the variable `color[]`

Example 5-21

```
#include <ipst.h>
void setup()
{
   glcdLine(0,0,127,159,GLCD_RED);
   // Draw a red diagonal line from top left to bottom right
}
void loop()
{}
```



5.3.20 glcdCircle

Draw a circle function.

Syntax

```
void glcdCircle(unsgined int x, unsgined int y,
  unsgined int radius,unsgined int color)
```

Parameter

x - Center of thge circle coordinator on x-axis. Value is 0 to 127

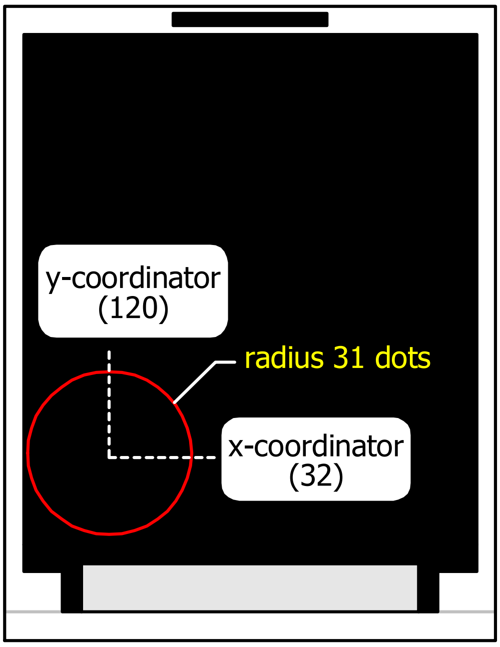
y - Center of thge circle coordinator on y-axis. Value is 0 to 159

radius - Radius value

color - Circumference color. It is 16-bit data or variable data which is defined from the variable `color[]`

Example 5-22

```
#include <ATX2.h>
void setup()
{
   glcdCircle(32,120,31,GLCD_RED);
   //Draw a red circle with 31 dots radius
}
void loop()
{}
```



5.3.21 glcdFillCircle

Creates a filled circle without the circumference.

Syntax

```
void glcdFillCircle(unsigned int x,unsigned int y,
   unsigned int radius,unsigned int color)
```

Parameter

x - Center of thge circle coordinator on x-axis. Value is 0 to 127

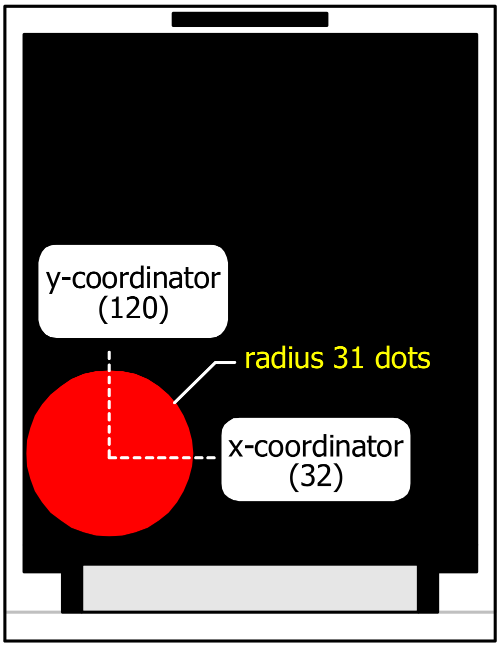
y - Center of thge circle coordinator on y-axis. Value is 0 to 159

radius - Radius value

color - Circle color. It is 16-bit data or variable data which is defined from the variable **color[]**

Example 5-23

```
#include <ATX2.h>
void setup()
{
   glcdFillCircle(32,120,31,GLCD_RED);
   // Create the solid red circle with radius 31 dots
}
void loop()
{}
```



5.3.22 glcdArc

Draw the arc line function.

Syntax

```
void glcdArc(unsigned int x,unsigned int y,
  unsigned int r,int start_angle,int end_angle,uint color)
```

Parameter

x - Center of thge circle coordinator on x-axis. Value is 0 to 127

y - Center of thge circle coordinator on y-axis. Value is 0 to 159

radius - Radius value of the arc

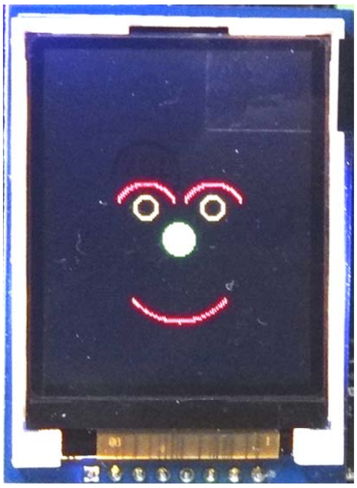
start_angle - Start angle of the arc

end_angle - Ending angle of the arc

color - Arc line color. It is 16-bit data or variable data which is defined from the variable **color[]**

Example 5-24

```
#include <ATX2.h>
void setup()
{
   glcdArc(48,80,16,30,150,GLCD_RED);
   glcdCircle(48,75,5,GLCD_YELLOW);
   glcdCircle(80,75,5,GLCD_YELLOW);
   glcdArc(80,80,16,30,150,GLCD_RED);
   glcdFillCircle(64,90,7,GLCD_GREEN);
   glcdArc(64,100,30,220,320,GLCD_RED);
}
void loop()
{}
```



Chapter 6

Hardware experiment of ATX2 board

This chapter presents examples of the hardware experiment with the ATX2 controller board of the Robo-Creator2 robot kit. There are 4 main activities as follows.

Activity 1 : Shows message on the display of the ATX2 board (5 sub-activities)

Activity 2 : Reading KNOB and OK button of the ATX2 board

Activity 3 : Control the simple output devices

Activity 4 : Simple I/O controller

Activity 5 : Sound activity

The procedure of development of each activity is the same. That is open Arduin 1.0.7 software, create the sketch, compiled and uploaded onto the ATX2 board. Finally, test the running program.

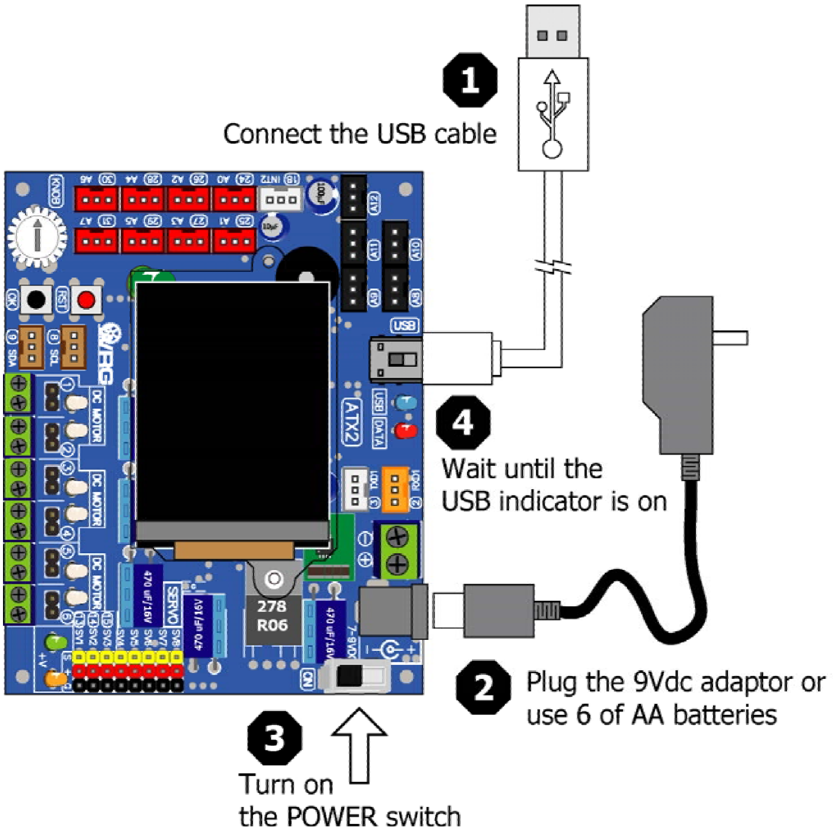
The important thing to emphasize is every time you turn on the power to the ATX2 board. Wait for the controller to ready first. It takes about 3 to 5 seconds after turning on the power or after pressing the RESET switch. If it is uploaded before ATX2 board ready, The work may cause an error in the connection or the uploaded code does not work as it should be. It will not damage the controller board. It just the board does not work or work not properly.

Activity 1 : Shows message on the display of the ATX2 board

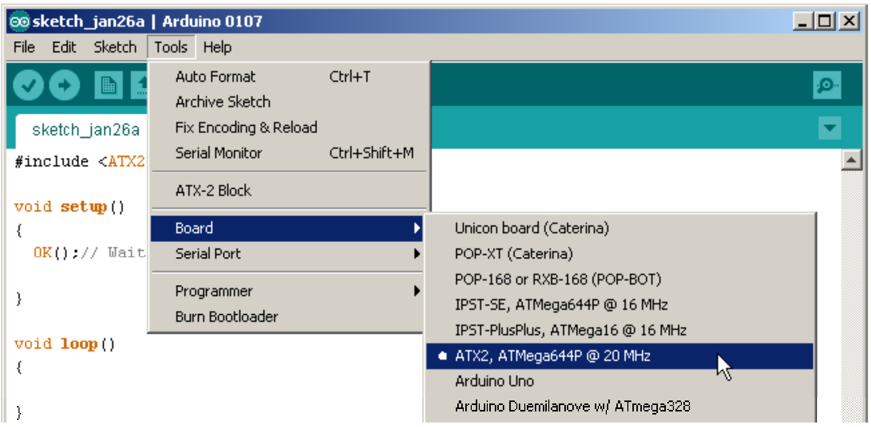
Activity 1-1 Hello World

(A1.1.1) Open the Arduino 1.0.7 software. Create the new sketch file. Type the code following the Listing A1-1. then save as the sketch.

(A1.1.2) Apply the supply voltage to the ATX2 board and turn on power. Connect the USB cable between ATX2 board and computer.



(A1.1.3) Choose the hardware at menu **Tools > Board > ATX2; ATmega644P @20MHz**


sketch_jan26a | Arduino 0107
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
ATX-2 Block
Board
Serial Port
Programmer
Burn Bootloader
Unicon board (Caterina)
POP-XT (Caterina)
POP-168 or RXB-168 (POP-BOT)
IPST-SE, ATMega644P @ 16 MHz
IPST-PlusPlus, ATMega16 @ 16 MHz
● ATX2, ATMega644P @ 20 MHz
Arduino Uno
Arduino Duemilanove w/ ATmega328
sketch_jan26a

```
#include <ATX2
void setup()
{
  OK();// Wait
}
void loop()
{
}
```

```
#include <ATX2.h>              // Include main library
void setup()
{
  glcdMode(1);                 // Set orientation mode 1 (90 deg)
  setTextSize(2);              // Set text size 2x
  glcd(1,0,"Hello World");     // Send message to display
}
void loop()
{}
```

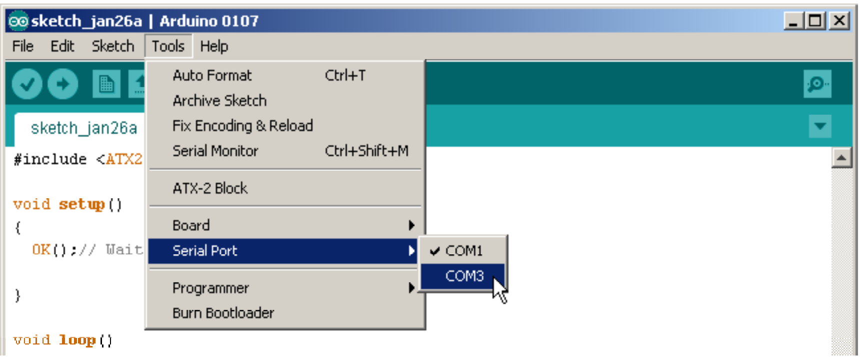
Code description

This code will set the display orientation to 90 degree first (mode 1) because the ATX2 board orientation is wide. Next, set text size as 2x and send message "Hello World" to display at the line 1 column 0 of GLCD screen.

This sketch run at once. Because all code are contained in **`setup();`** bracket

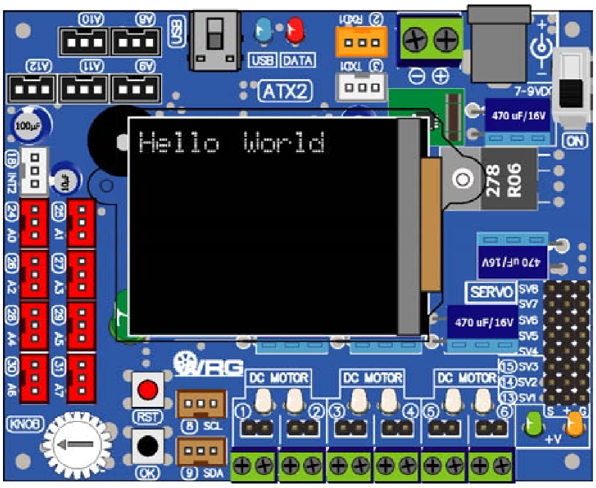
Listing A1-1 : ATX2_HelloGLCD.ino; the sketch file for sending message to display on the GLCD scrren of the ATX2 board

(A1.1.4) Choose the interfaced port at menu **Tools > Serial Port** (Number of interfaced port may be different depend on each computer)



(A1.1.5) Compile and upload the sketch to ATX2 board by clicking on the ➔ button or choose at menu **File > Upload**

At the ATX2 screen, message Hello World *will be displayed.*



Activity 1-2 Multipleline display

The GLCD screen of ATX2 board size is 128 x 160 dots. Show the character with 5 x 7 dots resolution by 21 charactes 16 lines. Developer can define the position of each line and column on the screen by using glcd function of the ipst.h library file.

Additionally, glcd function provides the special characters for setting the display position insteadto use the position number. It will be show in the Listing A1-2.

(A1.2.1) Open the Arduino 1.0.7 software. Create the new sketch file. Type the code following the Listing A1-2. then save as the sketch.

```
#include <ATX2.h>                         // Include main library
int i;
void setup()
{
  glcdMode(1);                            // Set orientation mode 1 (90 deg)
  glcdFillScreen(GLCD_WHITE);             // Set background color as white
  setTextColor(GLCD_BLACK);               // Set text color as black
  setTextBackgroundColor(GLCD_WHITE);     // Set text background as white
  for (i=0;i<13;i++)                      // Loop 13 times for displaying messages

  {
    glcd(i,i,"Row %d ",i);                // Display messages on the screen
  }
}
void loop()
{}
```

Code description

This sketch add 3 functions of GLCD displaying as follows :

1. `glcdFillScreen` - Set the screen background color function
2. `setTextColor` - Set the text color function
3. `setTextBackground` - Set the text background color function

After screen setting already, the sketch will send the Row message following line number that get from increase the value of i variable and also shift the position by using i variable value.

Thus, firt line will be displayed Row0 message at column 0. Next on the line 2 will show Row1 message respcetively untill line 12. It will be show Row 12 message at the column 12.

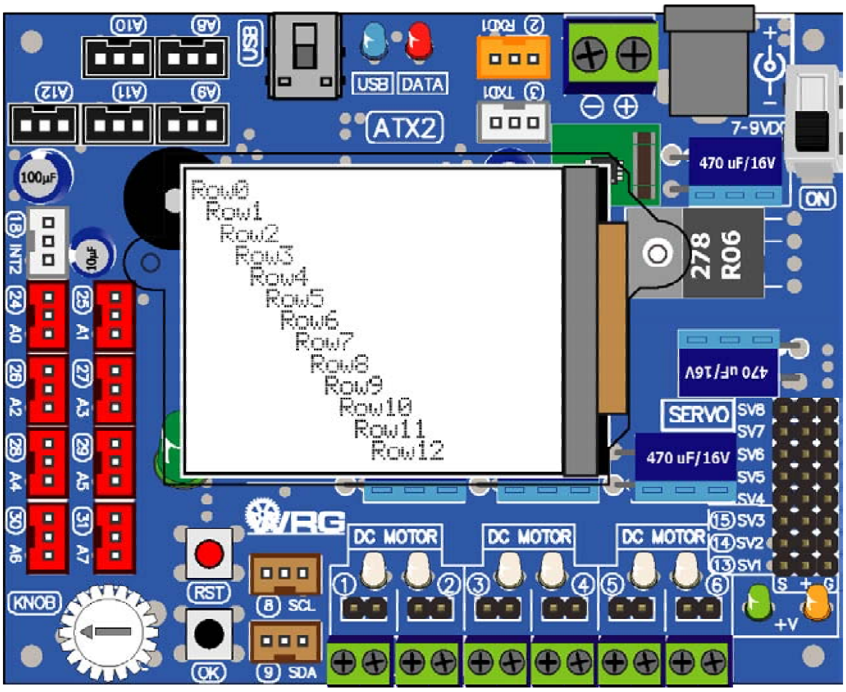
Listing A1-2 : ATX2_GLCDmultipleline.ino; the sketch file of Arduino for sending message to display on the GLCD screen of the ATX2 board in multiline.

(A1.2.2) Apply the supply voltage to the ATX2 board and turn on power. Connect the USB cable between ATX2 board and computer.

(A1.2.3) Compile and upload the sketch to ATX2 board by clicking on the ➡ button or choose at menu **File > Upload**

At the ATX2 screen, it shows Row0 to Row15 message on each line respectively.

Summary, wide orientation of GLCD screen could be display 13 lines maximum with standard text size.



```
Row0
 Row1
  Row2
   Row3
    Row4
     Row5
      Row6
       Row7
        Row8
         Row9
          Row10
           Row11
            Row12
```

Activity 1-3 Fun with size and rotation text on GLCD

Font size displayed on the GLCD screen of the ATX2 board when it starts is the smallest. The number of dots per character is 6 x 10 dots (the actual font size is 5 x 7 dots). To adjust the font size, use **`setTextSize()`** function. The value size in () is time of the default size. For example :

`setTextSize(2)` is set text size to 2 times of the default size. The number of dots per character is 12 x 20 dots.

`setTextSize(3)` is set text size to 3 times of the default size. The number of dots per character is 18 x 30 dots.

When adjusting the font size is larger. Number of characters per line was reduced from 21 characters 16 lines. Afdter set the text size 2 times, it will display 10 characters 8 lines. This is important factor that all programmers have to concentrate.

In addition to setting the font size, programmer can rotate the displaying by using the function **`glcdMode()`**. The default mode is 0 (**`glcdMode(0)`**) is displayed vertically. For another 3 modes 1, 2 and 3, rotate the displaying 90 degrees each. That is from 0 to 90 degrees (mode 1), 90 to 180 degress (mode 2) and 180 to 270 degrees (mode 3).

(A1.3.1) Open the Arduino 1.0.7 software. Create the new sketch file. Type the code following the Listing A1-3. then save as the sketch.

```
#include <ATX2.h>
int x,m;
void setup()
{
   //glcdSetColorWordRGB();  // Enable this function if displaying color
                             // of GLCD incorrect
   setTextColor(GLCD_RED);   // Set text color as red
}
void loop()
{
   for (x=1;x<6;x++)
   {
     setTextSize(x);         // Set text size
     for(m=0;m<4;m++)
     {
        glcdClear();         // Clear screen
        glcdMode(m);         // Set rotation of display
        glcd(0,0,"%dX",x);   // Show text size
        glcd(1,0,"M=%d",m);  // Show rotation mode number
        sleep(500);
     }
   }
}
```

Listing A1-3 : ATX2_FlipText.ino; the sketch file for testing about setting text size and rotation display direction of GLCD screen of the ATX2 board

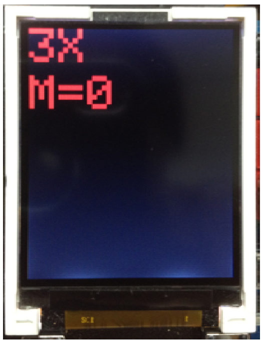
(A1.3.2) Apply the supply voltage to the ATX2 board and turn on power. Connect the USB cable between ATX2 board and computer.

(A1.3.3) Compile and upload the sketch to the ATX2 board by clicking on the ➡ button or choose at menu **File > Upload.**

ATX2 display shows the number of text size and number of display mode. Starts from the upper left, upper right, lower right and lower left corner. Surrounding the display starts from 1X, 2X, 3X, 4X and 5X each with 4 display direction by the M value.

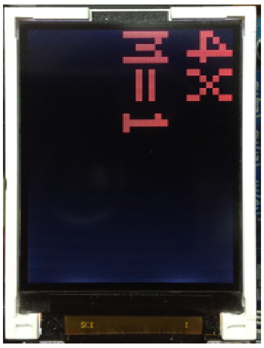
M = 0; display in vertical

Text size is 3 times (3X)



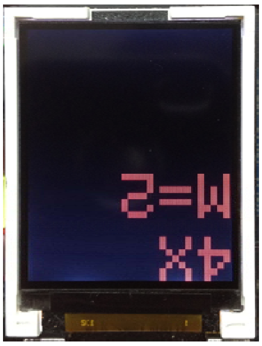
M = 1; rotate displaying to 90°

Text size is 4 times (4X)



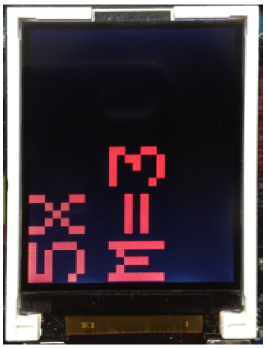
M = 2; rotate displaying to 180°

Text size is 4 times (4X)



M = 3; rotate displaying to 270°

Text size is 5 times (5X)



Activity 1-4 : Simple graphic displaying

The glcd() function is main function for GLCD screen of ATX2 board operation. Except showing text messages, this function also support more commands for drawing lines and simple geometry shape. Includes :

`glcdRect(int x1,int y1,int width,int height,uint color)` is function to draw a rectangle shape.

`glcdFillRect(int x1,int y1,int width,int height,uint color)` is function to create a solid rectangle shape.

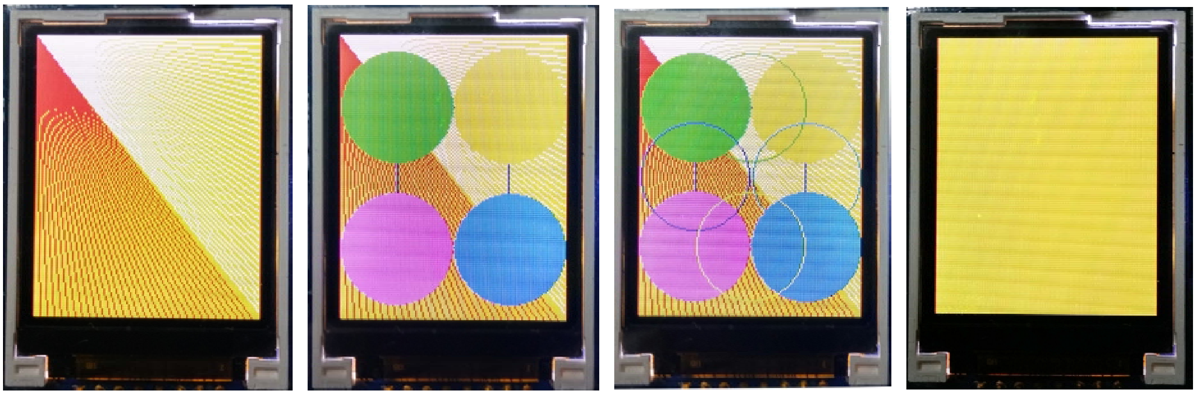
`glcdLine(int x1, int y1, int x2, int y2,uint color)` is function to draw the line.

`glcdCircle(int x, int y, int radius,uint color)` is function to draw a circle.

`glcdFillCircle(int x, int y, int radius,uint color)` is function to create a solid circle shape.

`glcdClear(uint color)` is clear display function.

The testing program is shown in the Listing A1-4. Create the sketch, compile and upload to the ATX2 board. The result is as follows :



If the color of the display is not correct, may be due to the version of the GLCD monitor. This GLCD monitor has 2 versions that different data setting. The solution is remove // symbol from the ***glcdSetColorWordRGB ();*** function in **setup();** . It will be change the color data setting.

```
#include <ATX2.h>                   // Include IPST-SE library
int i,j;
void setup()
{
   //glcdSetColorWordRGB();         // Enable this function if displaying color
                                    // incorrect
}
void loop()
{
   glcdClear;                       // Clear screen and set background to black
   sleep(300);
   for (i=0;i<160;i+=4)
   {
     glcdLine(0,0,128,i,GLCD_WHITE);      // Draw the white line from 0,0
                                          // coordinate to target
   }
   for (i=0;i<128;i+=4)
   {
     glcdLine(0,0,i,160,GLCD_RED);        // Draw the red line from 0,0
                                          // coordinate to target
   }
   sleep(2000);
   glcdRect(32,40,64,80,GLCD_BLUE);       // Draw the blue rectangle layout
   sleep(300);
   glcdFillCircle(32,40,31,GLCD_GREEN);   // Create the solid green circle
   glcdFillCircle(96,40,31,GLCD_YELLOW);  // Create the solid yellow circle
   glcdFillCircle(32,120,31,GLCD_MAGENTA);   // Create the solid margenta circle
   glcdFillCircle(96,120,31,GLCD_SKY);    // Create the solid light blue circle
   sleep(1000);
   glcdCircle(64,40,31,GLCD_GREEN);       // Draw the green circumference
   glcdCircle(32,80,31,GLCD_BLUE);        // Draw the blue circumference
   glcdCircle(64,120,31,GLCD_YELLOW);     // Draw the yellow circumference
   glcdCircle(96,80,31,GLCD_SKY);         // Draw the light blue circumference
   sleep(1000);
   glcdFillRect(0,0,128,160,GLCD_YELLOW); // Create the  solid yellow rectangle
   sleep(1000);
}
```

Listing A1-4 : ATX2_SimpleGraphic.ino; the sketch file for demonstration of simple graphic displaying of the ATX2 board

Activity 1-5 : Draw the curve

Except the circles and rectangle, the curve is a key component in creating graphics. In glcd function also provides a function to create the curve. It is **`glcdArc()`**. The parameter that must be reasonable. See more detail in chapter of the GLCD library.

(A1.5.1) Open the Arduino 1.0.7 software. Create the new sketch file. Type the code following the Listing A1-5. then save as the sketch.

(A1.5.2) Apply the supply voltage to the ATX2 baord and turn on power. Connect the USB cable between ATX2 board and computer.

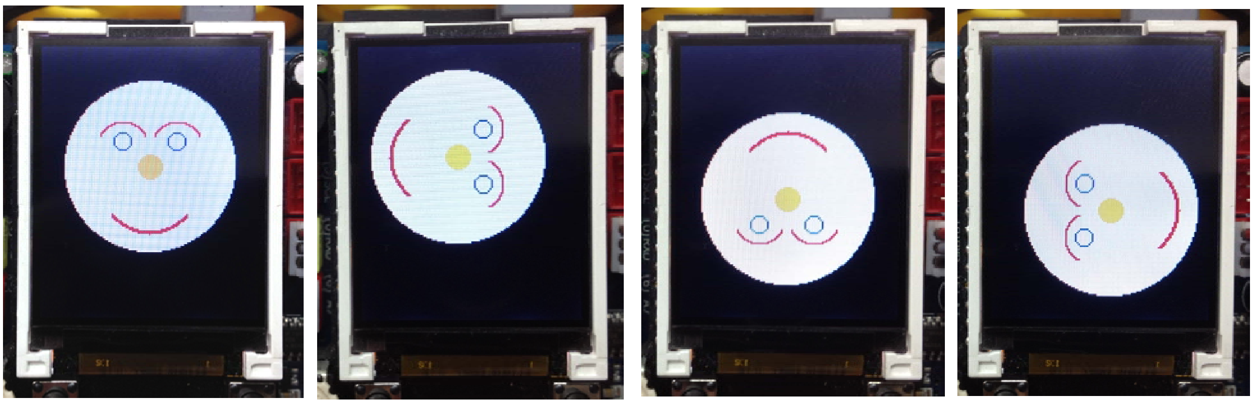
(A1.5.3) Compile and upload the sketch to ATX2 board by clicking on the ➡ button or choose at menu **File > Upload**

```
#include <ATX2.h>
int i;
// Smiley face graphic function
void face()
{
   glcdFillCircle(64,70,50,GLCD_WHITE);
   glcdArc(48,60,16,30,150,GLCD_RED);
   glcdCircle(48,55,5,GLCD_BLUE);
   glcdCircle(80,55,5,GLCD_BLUE);
   glcdArc(80,60,16,30,150,GLCD_RED);
   glcdFillCircle(64,70,7,GLCD_YELLOW);
   glcdArc(64,80,30,220,320,GLCD_RED);
   glcdArc(64,80,29,220,320,GLCD_RED);
}
void setup()
{
   //glcdSetColorWordRGB();         // Enable this function if displaying color
                                    // incorrect
}
void loop()
{
   for(i=0;i<4;i++)
   {
     glcdClear();
     glcdMode(i);                   // Rotate displaying
     face();
     sleep(1000);
   }
}
```

Listing A1-5 : GLCDarcTest.pde; the sketch file for demonstration of creating the curve on GLCD screen of the IPST-SE board

(A1.5.4) Run the sketch. See the ATX2 board operation.

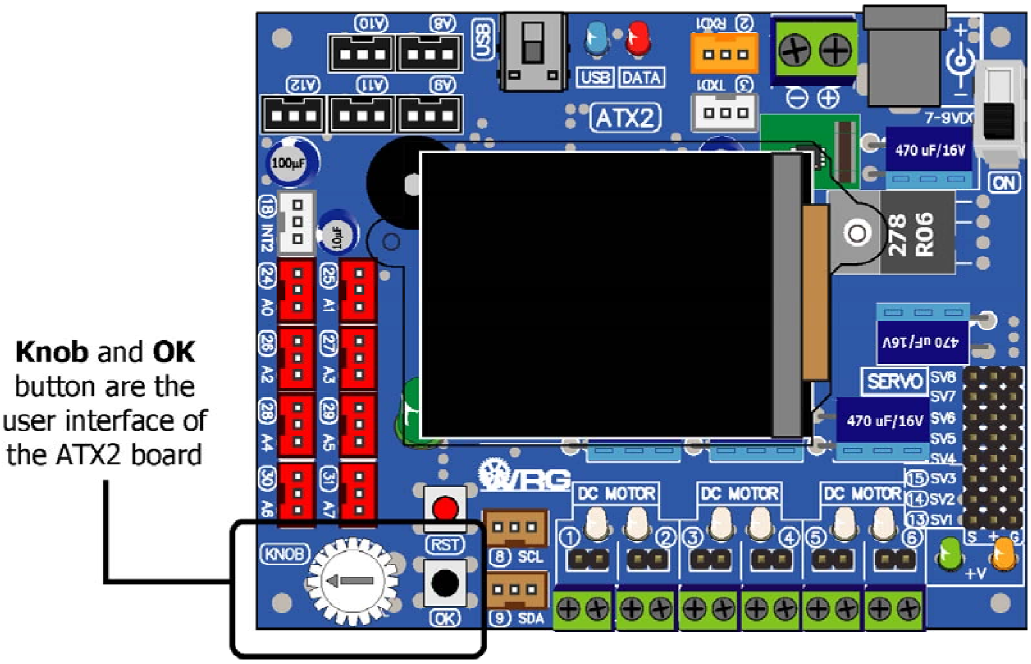
The display shows a smiley face for 1 second and turn it each 90 degrees respectively and then loop back to the start page. The display will go all the time.



Robo-Creator2
creative robotics kit

Activity 2 : Reading KNOB and OK button of the ATX2 board

Normal automatic control system must be configured with a switch in user interface hardware. The ATX2 board also provides the user interface hardware as well. It contains KNOB and OK button.



Knob and **OK** button are the user interface of the ATX2 board

(A2.1) Create the new sketch with Listing A2-1 and save as to **ATX2_OKbuttonKnobTest.ino**.

(A2.2) Compile and upload to the ATX2 board then run the sketch.

ATX2 display shows below message :

Press OK (*text size 2x)*

(A2.3) Press the **OK** switch to continue.

GLCD monitor shows a yellow circle 1 second then show message :

Knob value *(text size 2x)*

XXXX *(text size larger to 3x)*

therefore; XXXX *value is 94 to 1023*

(A2.4) Adjust the **KNOB** button on the ATX2 board.

Knob's value at the screen is changed following adjustment at KNOB button.

(A2.5) Press the **OK** switch and release.

GLCD shows the solid green circle 1 second then shows message and KNOB value.

```
#include <ATX2.h>            // Include the main library
void setup()
{
  //glcdSetColorWordRGB();   // Enable this function if displaying color
                             // incorrect
  glcdClear();               // Clear screen and set black backgound
  glcdMode(1);               // Set display mode 1
  setTextSize(2);            // Set text size 2x
  glcd(1,2,"Press OK");      // Show entry essage
  sw_OK_press();             // Loop for pressing the OK switch
  glcdClear();               // Clear screen and set black backgound
  glcdFillCircle(64,70,50,GLCD_YELLOW);  // Draw the solid yellow circle
  delay(1000);                           // Show graphic 1 second
  glcdClear();               // Clear screen and set black backgound
}
void loop()
{
  if (sw_OK())               // Check the OK switch pressed
  {
    glcdClear();             // Clear screen and set black backgound
    glcdFillCircle(64,70,50,GLCD_GREEN); // Draw the solid green circle
    delay(1000);                         // Show graphic 1 second
    glcdClear();             // Clear screen and set black backgound
  }
  glcd(1,2,"Knob value");    // Show message
  setTextSize(3);            // Set text size as 3x
  glcd(2,2,"%d   ",knob());  // Show the value of KNOB button
  setTextSize(2);            // Set text size 2x
}
```

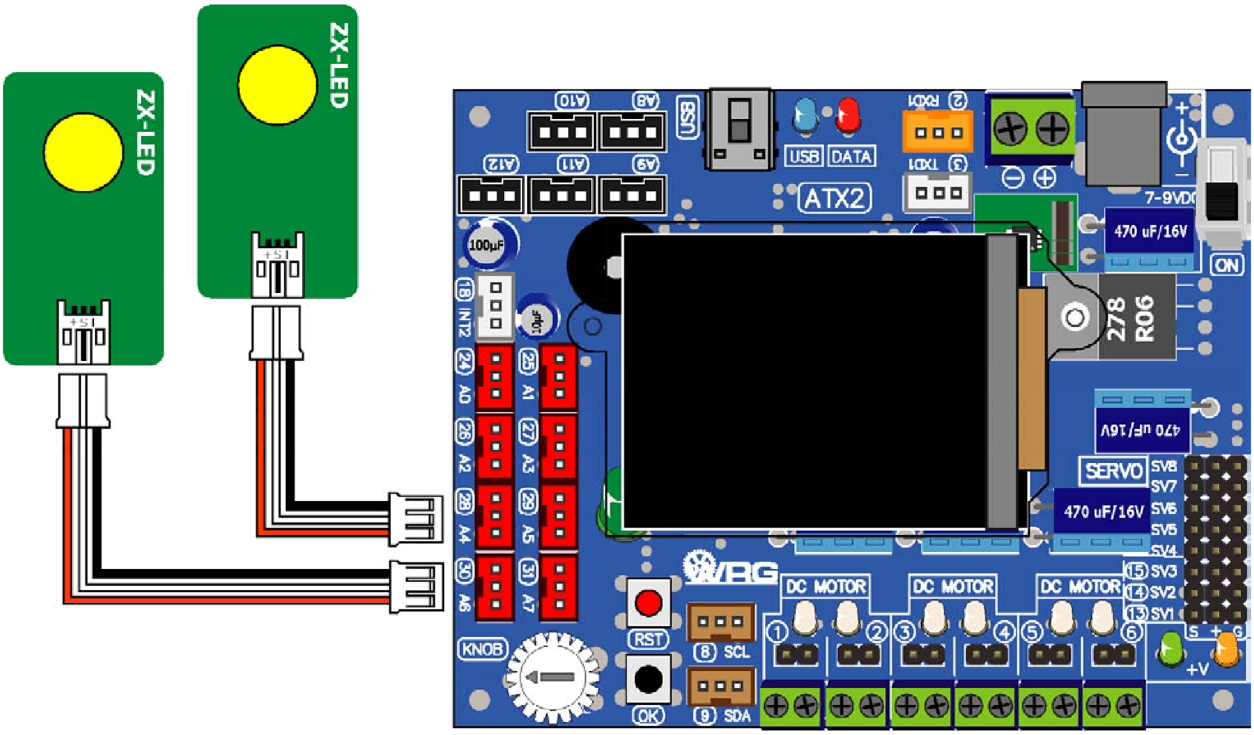
Listing A2-1 : ATX2_OKbuttonKnobTest.ino; the sketch for demonstration about reading the value of KNOB button, OK and SW1 switch on the ATX2 board

Activity 3 : Control the simple output devices

The library file ATX2.h provides out (char _bit, char _dat) function for sending a logic "0" or "1" to the microcontrolle's digital output pin. It helps the ATX2 to to drive the output device easier. The simplest example device is LED.

In this activity, connect the ZX-LED board to the ATX2 board at any pin. LED on ZX-LED board will be on when receive the logic "1" and off when receive the logic "0"

(A3.1) Connect the first ZX-LED board to pin 28 and second one to pin 30.



(A3.2) Create the new sketch with Listing A3-1 and save as **ATX2_LEDTest.ino** file.

(A3.3) Apply the supply voltage to the ATX2 board and turn on power. Connect the USB cable between ATX2 board and computer. Compile and upload to the ATX2 board then run the sketch.

(A3.4) Observe the operation of both ZX-LED board.

*The ATX2 monitor shows the title message. Then press the **OK** button start. The screen shows message* > Running.. *and LED of both ZX-LED boards on and off alternately continuous.*

```
#include <ATX2.h>         // Include the main library
void setup()
{
   OK();
}
void loop()
{
   out(28,1);             // LED at pin 28 is on
   out(30,0);             // LED at pin 30 is off
   sleep(400);
   out(28,0);             // LED at pin 28 is off
   out(30,1);             // LED at pin 30 is on
   sleep(400);
}
```

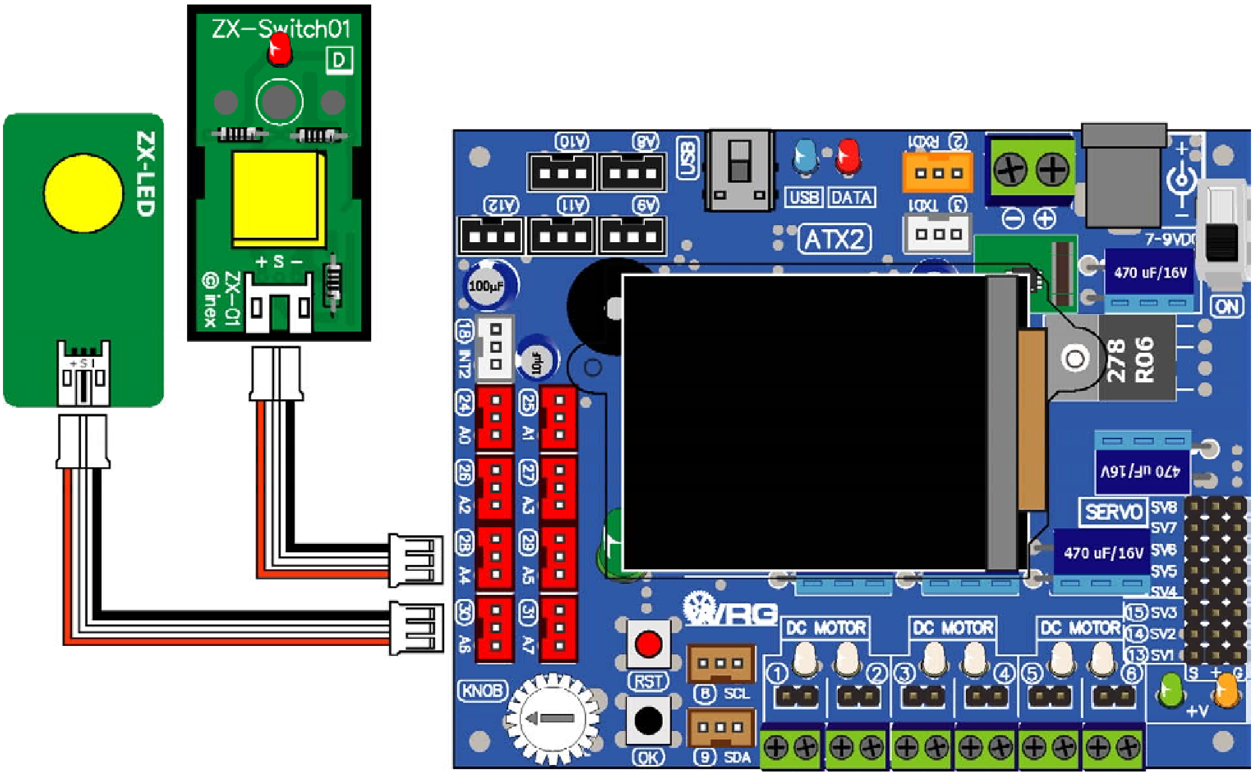
Listing A3-1 : ATX2_LEDTest.ino; the sketch for demonstration about simple driving the output devices of the ATX2 board

Activity 4 : Simple I/O Controller

The library file ipst.h provides in (x) function for getting the digital logic "0" or "1" from input port of microcontroller. It helps the ATX2 to get the logic from external input device such as a push-button swith.

In this activity, connect the ZX-LED board to the ATX2 board at any pin. Connect the ZX-SWITCH01 board that represent the digital input device to any pin of ATX2 board. Pressing and Releasign are used to control the LED on the ZX-LED board.

(A4.1) Connect the ZX-SWITCH01 board to pin 28 of ATX2 board and connect ZX-LED board to pin 30.



(A4.2) Create the new sketch with Listing A4-1 and save as **ATX2_SimpleIO.ino** file.

(A4.3) Compile and upload to the ATX2 board then run the sketch.

(A4.4) Observe the operation of switch and LED.

*The ATX2 monitor shows the title message. Then press the **OK** button start. The screen shows message* > Running..

Try to press the ZX-SWITCH01. The LED on ZX-LED boards will operate same pressing the switch. For example, no pressing LED is on because the logic is high. If pressed, LED is off because the logic is low.

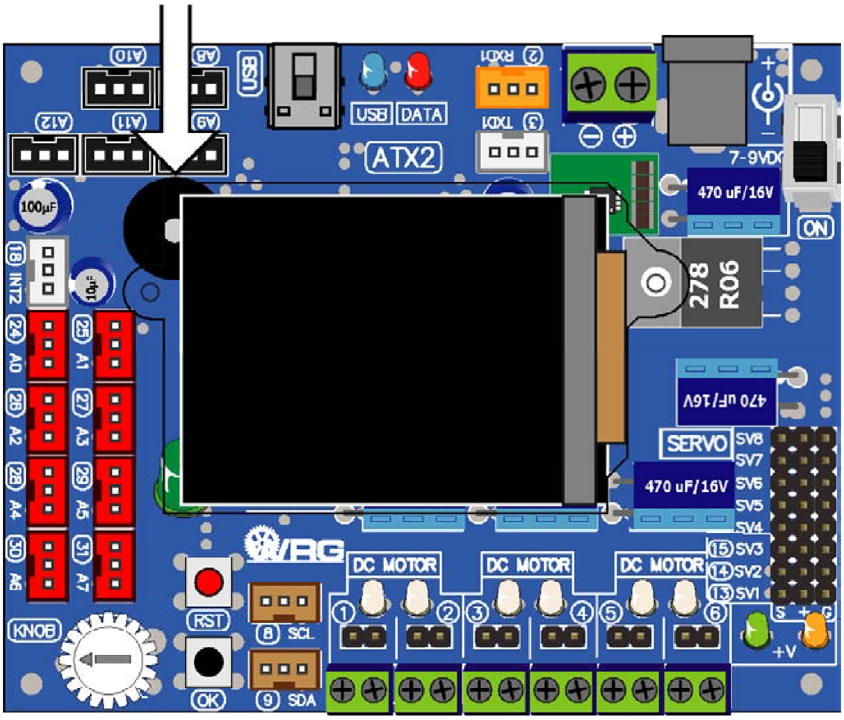
```
#include <ATX2.h>
void setup()
{
  OK();          // Wait for OK to Start
}
void loop()
{
  if(in(28)==1)   // Check status at pin 28
  {
    out(30,1);    // If pin 28 is high, LED at pin 30 is on
  }
  else            // If pin 28 is low, LED at pin 30 is off
  {
    out(30,0);
  }
}
```

Listing A4-1 : ATX2_SimpleIO.ino; the sketch for demonstration about simple input/output controller by using the ATX2 board

Activity 5 : Sound activity

The ATX2 controller board provides the sound output device on-board. It is piezo speaker. The speaker's resonance frequency is range of about 300 to 3,000Hz. For programming to drive this device, use `beep()` and `sound()` function.

On-board piezo speaker of ATX2 board; it is under the GLCD screen.



Listing A5-1 is example about how to use `beep()` function to drive the beep signal with 4 fixed frequencies every 1 second.

Listing A5-2 is example about how to use `sound()` function to drive the sound at any frequency in defined period.

(A5.1) Create the new sketch with Listing A5-1 and save as **ATX2_BeepTest.ino** file.

(A5.2) Compile and upload to the ATX2 board.

(A5.3) Run the sketch. Observe the operation of ATX2 piezo speaker.

To hear "beep" sound with 4 different frequencies in every 1 second.

(A5.4) Create the new sketch with Listing A5-2 and save as **ATX2_SoundTest.ino** file. Compile and upload to the ATX2 board again.

(A5.5) Run the sketch. Observe the operation of ATX2 piezo speaker.

To hear sound 2 frequencies alternately continuous.

```
#include <ATX2.h>    // Include the main library
void setup()
{
  OK();
}
void loop()
{
  beep();           // Generate the default “beep” signal
  sleep(1000);
  beep(1);          // Generate type 1 of beep signal
  sleep(1000);
  beep(2);          // Generate type 2 of beep signal
  sleep(1000);
  beep(3);          // Generate type 3 of beep signal
  sleep(1000);
}
```

Listing A5-1 : ATX2_BeepTest.ino; the sketch for demonstration about generate the beep signal of the ATX2 board

```
#include <ATX2.h>    // Include the main library
void setup()
{
  OK();
}
void loop()
{
   sound(500,500);     // Generate the 500Hz signal in 0.5 second
   sound(2500,500);    // Generate the 2500Hz signal in 0.5 second
}
```

Listing A5-2 : ATX2_SoundTest.ino; the sketch for demonstration about generate the sound with any frequency of the ATX2 board