EduCake Getting Started – Digital I/O

1. Introduction to EduCake and Integrated I/O

As part of today's modern living, there are broad ranges of microcontroller being used as part of the product all around our home and work place, such as electronic toy, appliance, remote control and entertainment system at home, and automated assembly system and other automation control at the work place. To the general public not working in the computer electronic and related field, learning to use these micro-controller and SoC is a difficult challenge, which require knowledge about how different type of electronic components work, knowing how to work with electronic schematic layout and perform complex low-level programming which take enormous times and efforts to learn. Many beginners attempting to learn are discouraged by the difficult challenge, lack of resources and give up along the way.

To address the difficulty in learning and using microcontroller, a team in Italy created the Arduino platform to help beginner learn and use microcontroller technology with a simple and easy to learn development environment. Released as an open-source platform, the Arduino platform gains popularity quickly and is being used by technical and non-technical hobbyist, professional and academic developers around the world. As an open-source platform, many Arduino communities and user groups are formed in different region and created large pool of technical resources, covering both hardware and software, in different languages. As the Arduino platform evolves, it becomes one of the popular platform for the academic and hobbyist communities.

Created by DMP Electronic, a company based in Taiwan, the 86Duino series of hardware is designed with Arduino compatible electronic interface, built with the following features:

- 300MHz 32-bit system-on-chip with an x86 CPU core
- 128MB DDR3 system memory
- 10/100 high speed Ethernet

- Two USB 2.0 interfaces
- Micro-SD
- Open-Source Hardware
- Support DOS, Windows, Linux
- Provide an integrated development environment (IDE) similar to Arduino with the same application programming interface (API), which enable existing application codes and class libraries written for the Arduino platform to function on the 86Duino platform without modification.

Created to support the academic community, the 86Duino EduCake, designed with a metallic enclosure to protect the internal circuitry and an integrated breadboard that provides easy access to the EduCake's I/O, is an ideal platform for teaching computer engineering, embedded system and related courses.

Fig-1-EduCake

Fig-2-EduCake rear view

Fig-3-EduCake front view

Fig-4-EduCake dimension

On the EduCake's integrated breadboard, digital I/O 0~13, analog input 0~5, ground, 5V, 3.3V and RX/TX signals similar to the Arduino Leonardo are clearly marked. These I/O interfaces are designed to be compatible to the Arduino Leonardo, with the same hardware and software function. Existing application codes and software libraries support these I/O interfaces without modification.

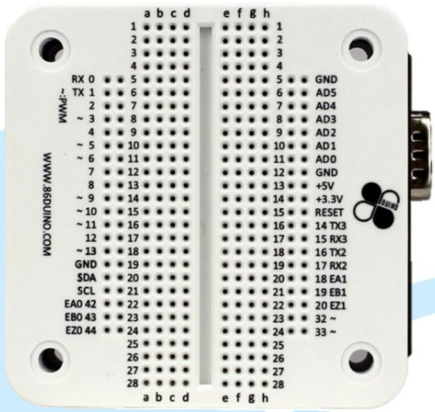

Fig-5- EduCake breadboard

Description for the EduCake breadboard's I/O:

1. There are 26 general purpose I/O (GPIO) accessible from the breadboard, 0~20, 31, 32, and 42~44. Each of these GPIO can support up to 16 mA of current, with over-current protection to a total of 26 digital I/O can be used as a general use of digital IO, current up to 16mA, limited protection to prevent inappropriate use of faults, so beginners can feel free to try a variety of applications.

2. Analog 0~5 is used to capture type of sensor inputs.

3. There are 3 groups of RX/TX with TTL signal, RX/TX, RX2/TX2, RX3/TX3, to support variety of communication.

4. Pin labeled with the "~" mark can provide PWM output similar to the Arduino platform. The 86Duino EduCake provides 3 addition PWM output via GPIO pin 13, 31 and 32.

5. There are additional SCL and SDA pins to support I²C communication.

6. Note: On the Arduino 328 platform, when I²C is used, analog pin #4 and #5 are needed, which take away 2 analog pins from performing other function.

7. The EA0~1, EB0~1, EZ0~1 are dedicated ENCODER for Motion Control. On the Arduino platform, addition add-on board is needed to provide these function.

8. The 5V pin is designed to bypass 3.3V, with 800mA output.

With existing application codes and software libraries written for the Arduino platform able to function on the 86Duino Educake, Arduino enthusiasts will find the EduCake development environment similar to the Arduino IDE and able to adopt and use the 86Duino IDE with their existing Arduino development skills. With the EduCake's powerful 300 MHz x86 processor, 128 MB of DDR3 system memory and additional integrated functions, Arduino enthusiasts can use EduCake to develop new innovations that are not possible on the Arduino platform.

2. Development Environment

One of the design objective for DMP development team is to create an 86Duino integrated development environment (IDE) that is compatible to the Arduino IDE with similar function and enable existing Arduino enthusiasts to adopt and use the 86Duino IDE without learning curve. Other than the color scheme, the 86Duino IDE is almost identical to the Arduino IDE.

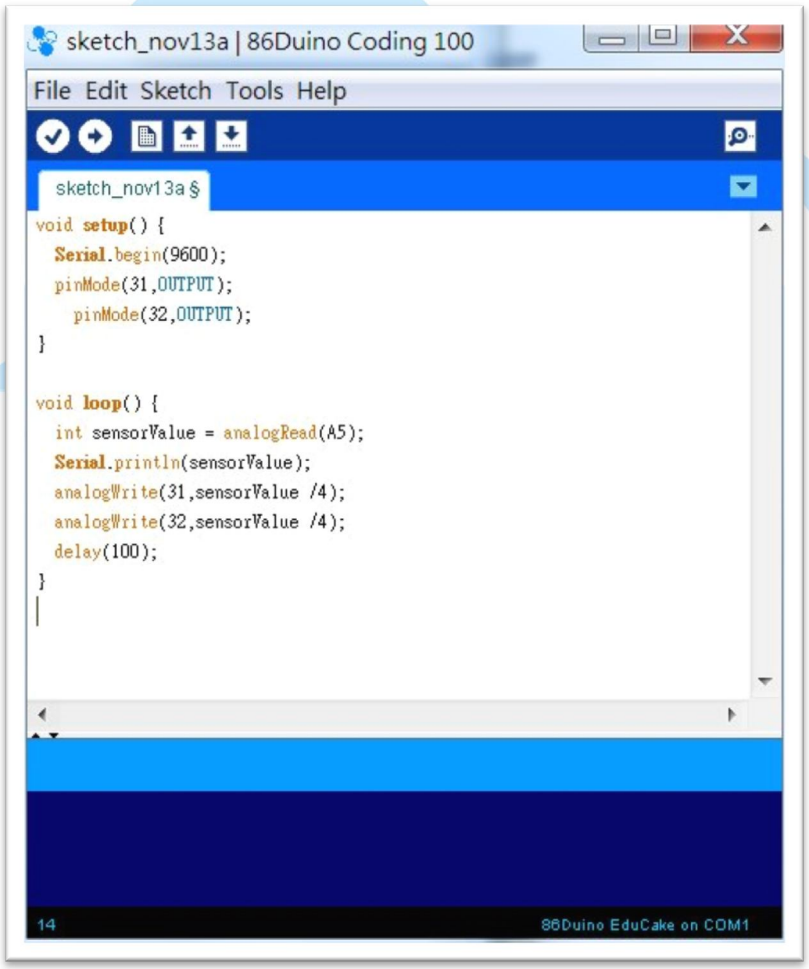

Fig-6: 86Duino integrated development environment

From the 86Duino IDE, click on **File→Examples**, you can see examples provided as part of the 86Duino IDE are similar to the Arduino examples, as shown in Fig-7.

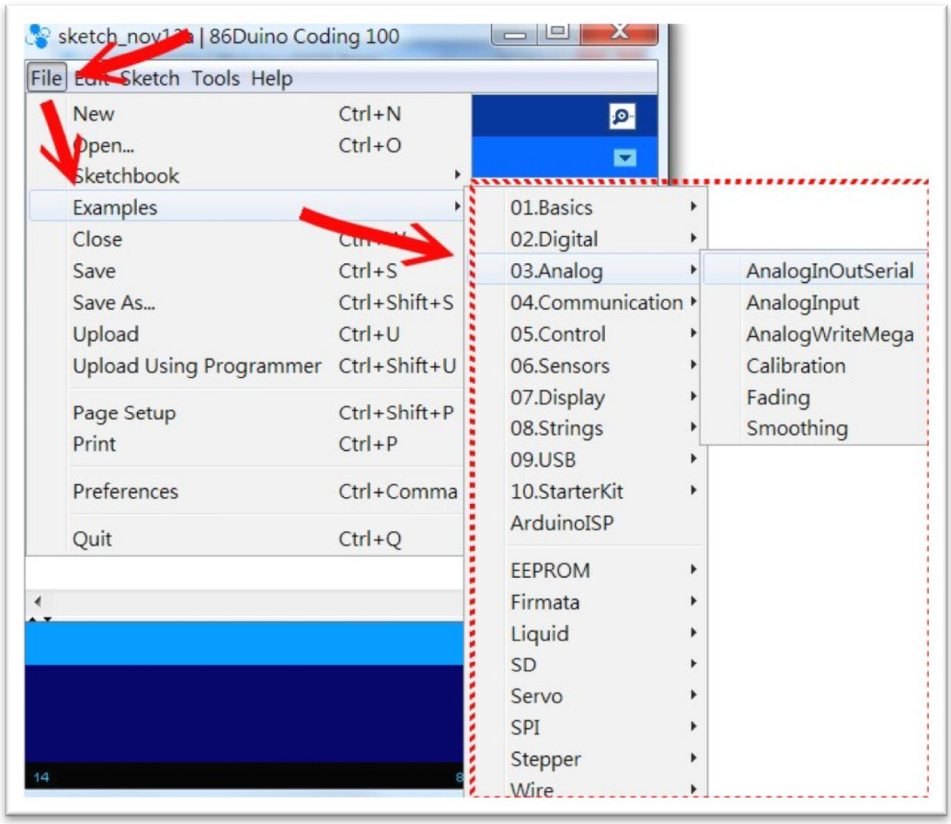

Fig-7: 86Duino IDE - Examples

The 86Duino IDE uses the same programming command and API as the Arduino platform. As shown on the 86Duino website, under the Reference-Language section (http://www.86duino.com), the 86Duino development environment adopt the same function call and programming structure as the official Arduino platform, as indicated on Arduino website (http://arduino.cc) in the Language Reference section. Experienced Arduino developers can jump into the 86Duino IDE and start writing codes for 86Duino, in the same manner as they've been writing codes for the Arduino platform.

The file directory structure for the 86Duino development environment is different from the Arduino. However, it does not impact the usage and function compatible to the Arduino IDE. Individual user can configure their development environment. Customizing the development environment is an advanced subject and will be covered in a later chapter.

After an application program is written, the process to deploy the application for 86Duino platform is the same as Arduino. To deploy application to the EduCake, you need to select the 86Duino EduCake as the target device from the 86Duino IDE, as shown in Fig-8.

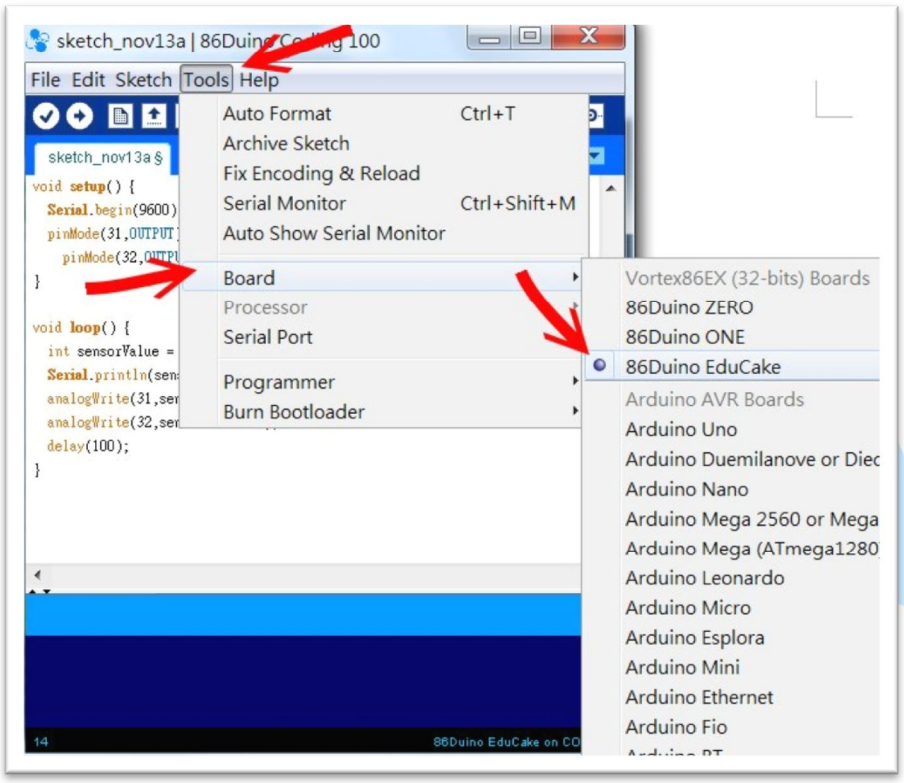

Fig-8: 86Duino IDE – Selecting target device

In addition to selecting the correct target device, the 86Duino IDE needs to be configured to use the correct serial port as the transport to deploy the application. For the example in this application note, COM1 is selected. The selected serial port is displayed on the 86Duino IDE screen's lower right. Make sure the correct serial port is selected. Incorrect serial port setting will prevent application from deploying to the EduCake. To configure the serial port setting, from the 86Duino IDE menu, click on **Tools→Serial Port** and select the serial port you want to use, as shown in Fig-9.

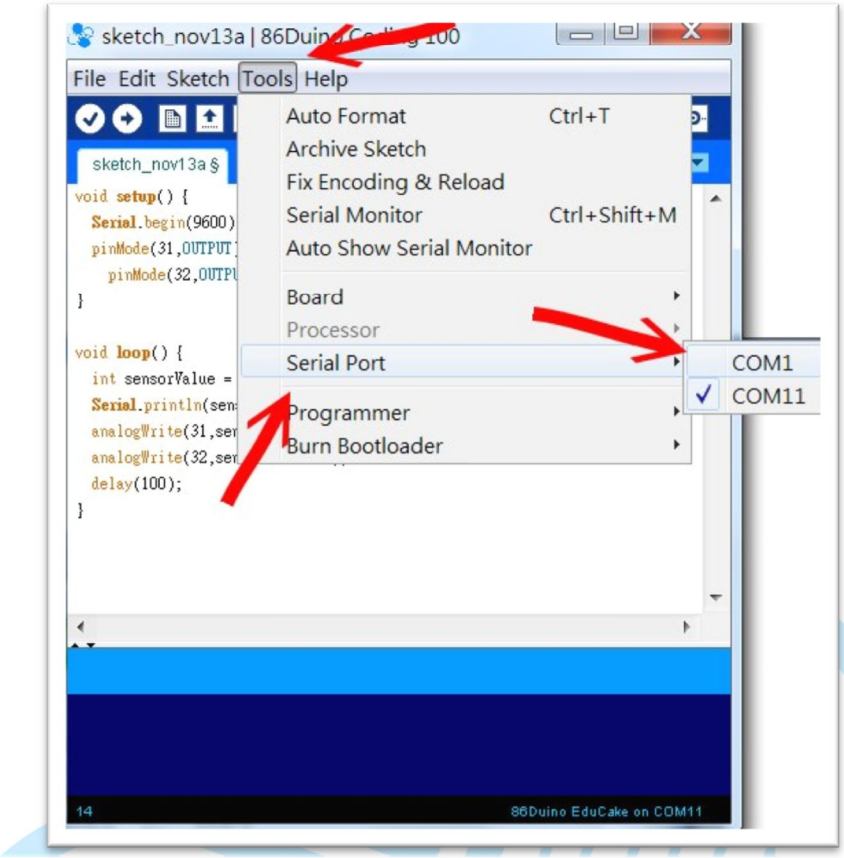

Fig-9: 86Duino IDE – Selecting serial port

With the application code written, proper target device and serial port settings configured, the 86Duino IDE is ready to deploy the application to the EduCake. To deploy application to the target device, from the 86Duino IDE screen's top left, click on the round icon with right-pointing arrow ("→"), as shown in Fig-10. The round icon with a check mark ("√") is used to verify the codes, which the author of this application note rarely used. When deploying application code to target device, the code is verified prior to deploy and avoid a separate redundant step to verify the code.

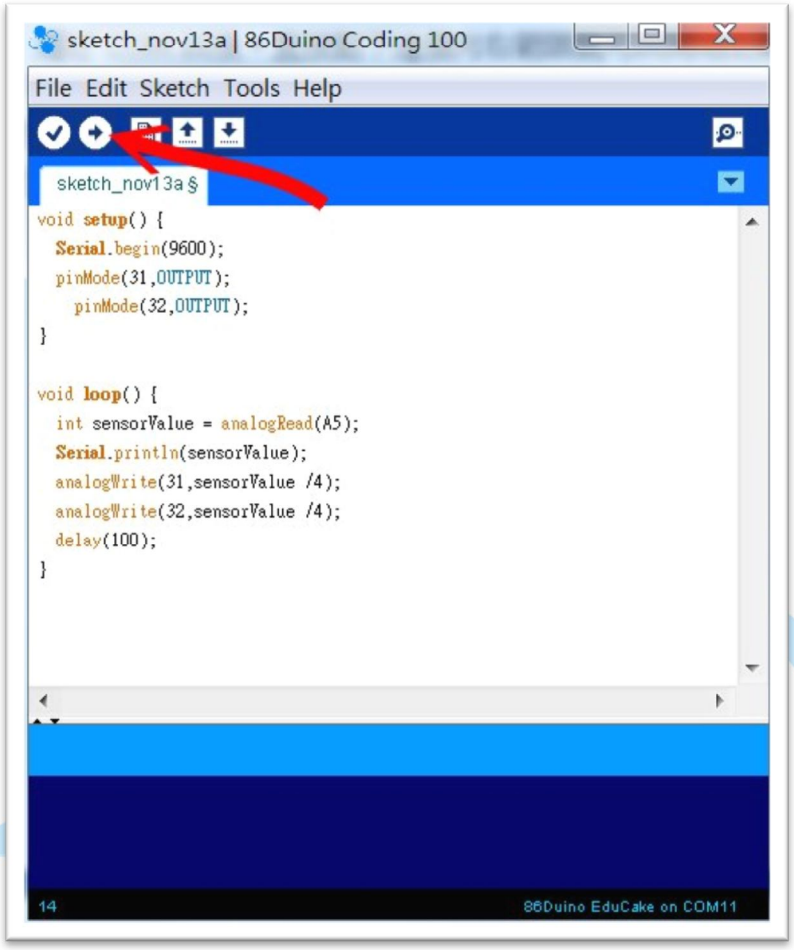

Fig-10: 86Duino IDE – Deploy application to target device

3. First 86Duino Sample Application

In this section, we will work through the process to develop a simple 86Duino application, to blink an LED. An LED and a resistor are needed for this example, attached to the EduCake's breadboard as shown in Fig-11. The current supply by the GPIO pin on the EduCake's breadboard is minimal. It's possible to work through this example without the resistor. However, it's recommended to include the resistor to work through this example. The resistor limit the current and protects the LED.

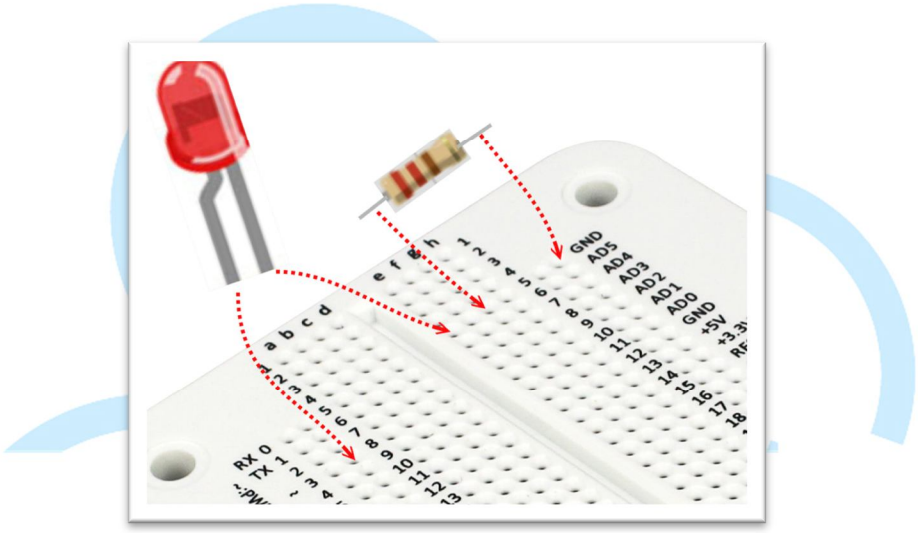

Fig-11: LED and resistor connected to the breadboard on EduCake

The LED is polarity sensitive. Make sure to attach the cathode of the LED to GPIO pin #3, as shown in Fig-11 above.

From the 86Duino IDE, enter the following codes:

```
void setup()
{
  pinMode(3, OUTPUT); // configure GPIO pin #3 as output
}
void loop()
{
  digitalWrite(3, HIGH); // Set GPIO pin #3 to HIGH and turn on the LED
  delay(1000);           // Delay 1000 ms (or 1 second)
  digitalWrite(3,LOW);   // Set GPIO pin #3 to LOW and turn off the LED
  delay(1000);                 // Delay 1000 ms (or 1 second)
}
```

When the above program is deployed to the EduCake and running, it continues to blink the LED on and off, in 1000 millisecond (1 second) interval. The example works the same on an Arduino board. As you can see, writing codes for the 86Duino platform is simple as the Arduino platform. By modifying the above codes to use PWM function, using the analogWrite() function, instead of the digitalWrite() function you can gradually turn on the LED and turn off the LED, with varying lighting intensity. By adding some random value to the analogWrite() function, you can create a candle-light flickering effect. Using just a simple LED, you can create different interesting variations.

From working through the above simple example, you can see the 86Duino IDE and API used in the application are identical to the Arduino environment. To the experienced Arduino enthusiast, the 86Duino IDE is just another Arduino development environment.

4. Second 86Duino Sample Application

With the first application that help validate the 86Duino development is properly setup and working, we can move forward and work on more complex application. In this second example, we will attach eight LEDs to the EduCake breadboard and work through the process to develop a little more complex application.

First, attach the eight LEDs and resistors as shown in Fig-12.

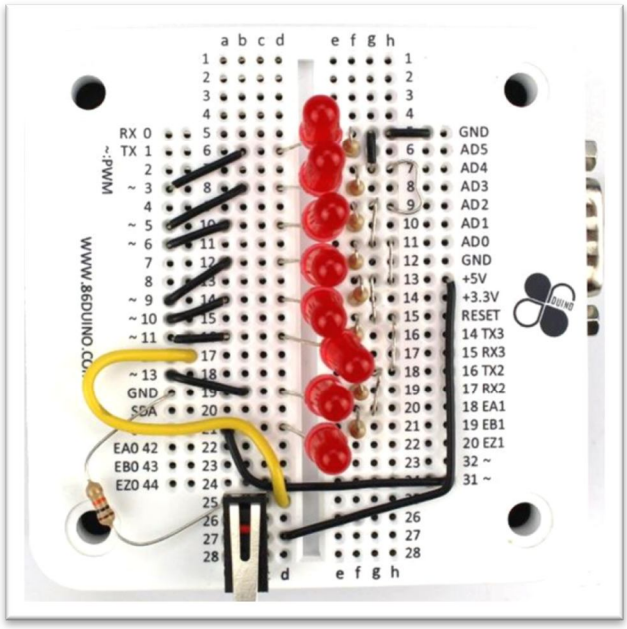

Fig-12: EduCake with LEDs and resistors attached – back view

Without making any change, here is the view, looking at the EduCake from the front with the eight LEDs and resistors attached, as shown in Fig-13.

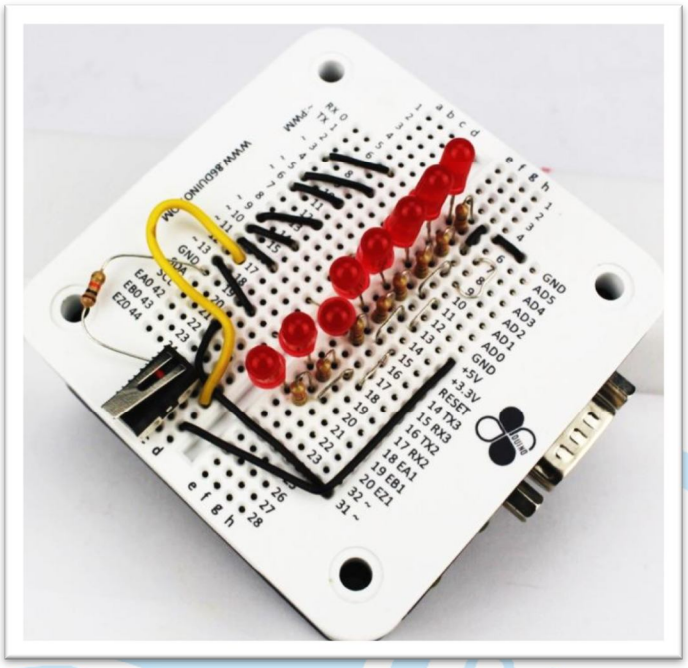

Fig-13: EduCake with eight LEDs and resistors attached – Front view

Taking into account that we may need a push button later on, a push button is attached to the EduCake breadboard, as shown in Fig-14.

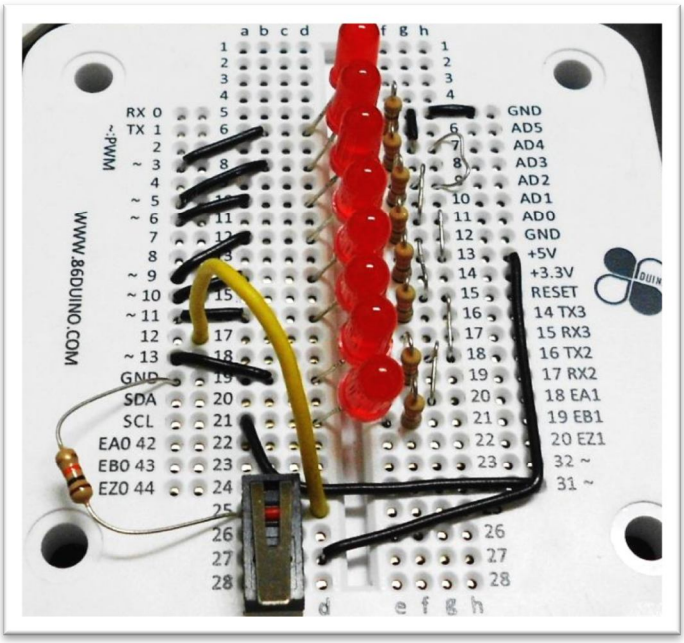

Fig-14: EduCake with LEDs, resistors and push button attached

With the eight LEDs, resistors and push button circuitry as shown in Fig-14 above, we can use the same circuitry for different exercises and experiments. In this next exercise, we will develop a marquee program that will light up each of the eight LEDs sequentially.

Following is the codes for the marquee program:

```
// GPIO pins with PWM are deliberately chosen to support multiple exercises
// without having to change the circuit
int led[]={3,5,6,9,10,11,13,31}; // store GPIO pins ID to an array
int pos =0;
void setup() {
   for(int a=0;a<8;a++) // Configure the GPIO pins to function as output
      pinMode(led[a],OUTPUT);
}

void loop() {
   digitalWrite(led[pos],LOW); // turn off the previous LED
   pos=(pos+1)%8; // determine the ID for the next LED to be turn on
   digitalWrite(led[pos],HIGH); // turn on the next LED
   delay(200); // delay briefly，increase delay time to slow down the light
movement
    }
```

The above codes will sequentially turn each of the 8 LEDs on and then off, making it looks like the light is moving, like a marquee.

The following line of code is simplified and shorten from multiple lines of code:

```
        pos=(pos+1)%8;
```

The following codes are equivalent to the above line of code.

```
pos ++;
        if (pos>=8) pos =0;
```

In the last line of code (delay (200;) for the marquee application, you can change the delay value to change the marquee's apparent, increase or decrease the marquee movement speed.

5. Third 86Duino Sample Application

In this next exercise, we will use the EduCake's serial port to communicate with the PC, and use the PC to control the EduCake remotely. In addition to the application running on the EduCake, we will need a serial port application running on the PC to communicate with the EduCake.

Following are the codes, for the EduCake, for the exercise in this section:

```
int led[]={
  3,5,6,9,10,11,13,31};
int pos =0;
void setup() {
  Serial.begin(9600); // Initialize and set communication baud rate for the
serial port
                      // Serial port can support 9600/19200/38400/115200
and etc.
                      // Both the PC and EduCake must be configured to
support the same speed
  for(int a=0;a<8;a++)
    pinMode(led[a],OUTPUT);
}

void loop() {
  char ch;
  if (Serial.available())          // Check for incoming messages from the
PC
  {
    ch=Serial.read();              // When there is message, read one
byte at a time.
    if (ch>='1' && ch<='8')        // Check for 1~8 within the incoming
message.
      digitalWrite(led[ch-49],HIGH);  // If 1~8 is detected, turn on the LEDs.
  }
  delay(200);
    }
```

In the following line of code:

```
      digitalWrite(led[ch-49],HIGH);
```

The variable ch is used to store ASCII data read from the serial port's incoming message, which is read one byte at a time. After the data is read, the ch variable contains one ASCII code. The ASCII representation for decimal 1 ~ 8 are 49 ~ 57. By subtracting 49 from the current ch value, we can convert the incoming message from the serial port betweem 0 ~ 7, which we can map the value to the led[] array, and use the value to turn the corresponding LED on.

To send serial port message to the EduCake, you can use the Serial Monitor from the 86Duino IDE, and send 1 ~ 8 to remotely turn on the corresponding LED on. From the 86Duino IDE, click on the icon on the top right of the screen to launch Serial Monitor, as shown in Fig-15.

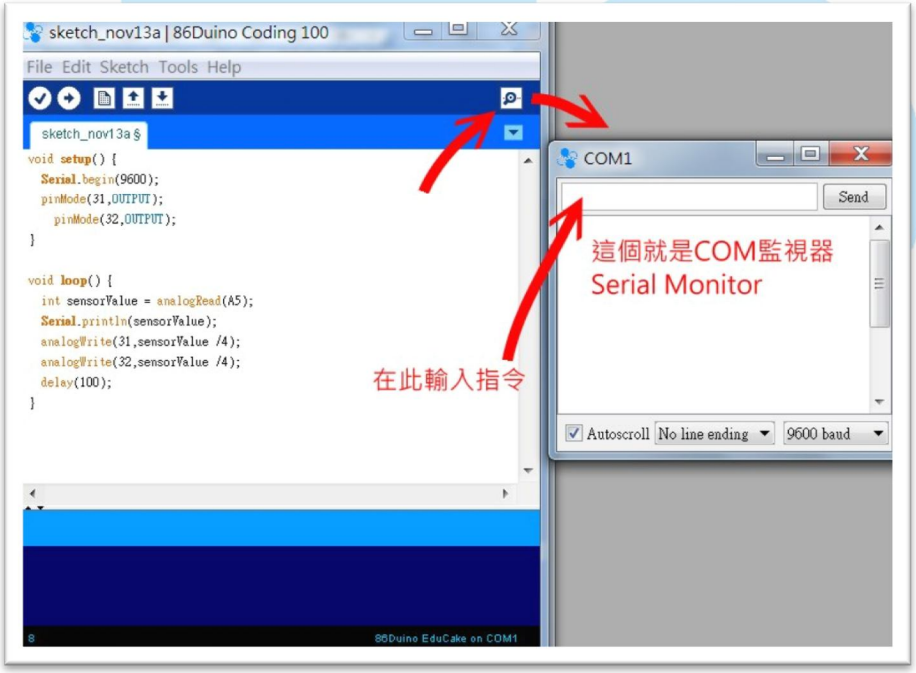

Fig-15: Serial Monitor - sending command to turn on LED

From the Serial Monitor screen, enter a value between 1 ~ 8 and click Send to turn on the corresponding LED attached to the EduCake's breadboard. The Serial Monitor is a common tool used during development to communicate with the 86Duino target device for debugging and testing purposes.

When the Serial Monitor is in use, it captures the serial port connecting to the 86Duino target device. When using an alternative serial port communication program, such as an application that you create using VB or C# with different User Interface to

control the EduCake, make sure the Serial Monitor is shut down to release the serial port, making it available to the alternative serial port communication program. Otherwise, when another application attempt to capture the same serial port already in-use by the Serial Monitor, it will cause severe conflict and require the development PC to reboot to resolve the conflict.

Any program that can communicate via the serial port can be used to communicate with the EduCake. Following is a program we use, written in Visual Studio 2008, as shown in Fig-16. You can use any version of Visual Studio, from 2005 through 2013 to develop similar application in Visual Basic or C#.

Fig-16: LED control application written in Visual Basic, using Visual Studio 2008

To use the LED control application, select the COM port used to connect to the EduCake and set the transfer rate (BPS) to 9600 and click on the connect button (連線) to connect to the EduCake.

The LED control application is a Windows application written in Visual Basic and is not within the scope to be covered by this application note. You can develop similar application in Visual Basic with additional function, such as using the mouse wheel as the control, to turn on each LED sequentially in different direction as the mouse wheel turns. As you learn more about the Analog pin's function, you can create application to detect battery voltage and use the detected voltage value to control the LED lighting, and more. With some creative thinking, you can create many different applications.

If you prefer to use C#, the programming logic for C# is quite similar to Visual Basic. Searching the Web with the "VB2C#" keyword, you can find a number of websites with standalone utility or online resources to convert Visual Basic source codes in to C#. In the later section, we will include sample application for mobile phone.

The application code written in the previous section has a problem. After each LED is turned on, it remains on. If you send subsequence command to turn on the same LED, it does not show any visible affect and may give the wrong impression that the application is not working properly. To fix this problem, you can simply replace the following line of code:

```
digitalWrite(led[ch-49],HIGH);
Replace the above line of code with the following:
}
digitalWrite(led[ch-49],HIGH);
  delay(1000);
  digitalWrite(led[ch-49],LOW);
        }
```

The above code turn on the LED for 1 second and then turn the LED off. You may be asking: Why we did not use this code to begin with?

It's a process to learn programming. It's a process to develop an application. Often, we simply cannot have a perfect answer with a single attempt. As part of the application development process, we start by develop the application with some of the function and feature, test to make sure each of these function is doing the job as intended. Then, we add additional function, continue to identify problem area and correct them, continue to test and validate the function until all of the required function and feature are included in the application and passed the required test. As we repeating the process, reviewing and testing the codes we wrote, we find better approach, write better code and learn new things through this repeated process.

6. Fourth 86Duino Sample Application

Once you know the base functionality, you can write a more complete application with practical functions. Since this chapter talks about the EduCake's digital I/O, we will create a simple application using the LEDs to simulate a dice and rolling the dice. The application will turn each LED on and off sequentially, moving from one direction and reverse to the opposite direction as it reaches the last LED. As the application is running, the rate (speed) at which the LED is moving decrease (slow down) gradually and stop randomly at one of the LED to simulate the same effect as rolling a dice. For the circuit we are using, there are 8 LEDs. If preferred, you can modify the circuit to use 6 LEDs instead.

Here are the codes for this rolling dice application:

```
int led[]={
   3,5,6,9,10,11,13,31};

// A group of 3 variables are used to keep track of the current LED that is on,
// the next LED to turn on and the last LED within the current group to turn on,
// to control the direction of the moving LED.
int nowPos =2;          // Current LED that is turned on
int midPos=1;           // Next LED to turn on
int lastPos=0;          // Last LED to turn on
// The dir variable is used to control the moving direction.
int dir=1;              // Moving direction, 1: from 0 to 7, -1: from 7 to 0
// The spd variable is used to control the moving speed.
int spd = 20;           // Moving speed, smaller value = faster, larger value =
slower

void setup() {
Serial.begin(9600);
   for(int a=0;a<8;a++)
      pinMode(led[a],OUTPUT);
   randomSeed(analogRead(0));      // Retrieve a random number
Start to initialize the random number seed

}
```

```
    void loop() {
      if (Serial.available())
      {
        char ch=Serial.read();
        if (ch=='1')       // Received command from the serial port to start the
process
        {
          spd =20;
          // nowPos =2;
          midPos=nowPos-1;
          lastPos=nowPos-2;
          dir =1;
        }
      }
      if (spd<220) // as the value controlling the moving speed reach 220, the process
stop
      {
        digitalWrite(led[nowPos],HIGH);   // turn on the current LED
        if (midPos<8 && midPos>=0)
          analogWrite(led[midPos],40);     // turn on the next LED with medium
intensity
        if (lastPos<8 && lastPos>=0)
          analogWrite(led[lastPos],15);     // turn on the last LED with low
intensity

        delay(spd);

        spd +=5;          // Increase the spd value to gradually slow-down the
movement
        if (spd>=220 )
        {
          if (midPos<8 && midPos>=0)          // turn off all LED
            digitalWrite(led[midPos],LOW);
          if (lastPos<8 && lastPos>=0)
            digitalWrite(led[lastPos],LOW);
          digitalWrite(led[nowPos],LOW);
```

```
      digitalWrite(led[random(0, 8)],HIGH);  //get a random number, turn on
corresponding LED
            spd =1000;
                }
    if (lastPos<8 && lastPos>=0)
            digitalWrite(led[lastPos],LOW); //turn off last LED，preparing to move
        lastPos=midPos;
        midPos=nowPos;
        nowPos+=dir;
        if (nowPos>7)
        {
          nowPos=7;
          midPos=8;
          lastPos=9;
          dir=-1;
        }
        else    if (nowPos <0)
        {
          nowPos=0;
          midPos=-1;
          lastPos=-2;
          dir=1;
        }
      }

     delay(spd/3);
          }
```

In the last line of code.

```
          delay(spd/3);
```

The spd variable is used as a self-adjusted value to control the moving speed. You can use a serial port communication program to change the speed or other methods to make the application more interesting.

Instead of using the Serial Monitor or a separate serial port communication program to send reset signal to the application, you can use the push button, to function as the reset button. In the earlier section, as part of setting up the LED

circuit, a push button is attached to digital pin #12. You can use the following modified codes to check the push button's status, to detect whether it's been pressed (The line with "。。。" represent unmodified code and is omitted to safe space):

```
。。。
void setup() {
  。。。
  pinMode(12,INPUT);    // New code added，to set digital pin #12 as input
}

void loop() {
  int bb;
  bb=digitalRead(12); // reading digital Pin #12
  Serial.println(bb); // Output the reading to serial port for debug
  if (bb==1) // If 1 is detected, the button is pressed
    run_again(); // Call the run_again function

  。。。
}

void run_again()      // Add this function outside of the program loop
                      // This function reset the LED, preparing the program to run
again
{
  spd =20;
  midPos=nowPos-1;
  lastPos=nowPos-2;
  dir =1;
      }
```