EduCake Serial Port Communication

1. Serial Communication Introduction

In previous chapters, we talked about the input and output (I/O) interfaces on the EduCake and work through sample exercises to blink LED, control motor and capture sensor data. All of these I/O interfaces are part of 86Duino EduCake hardware. To control another microcontroller, interface to a PC or communicate with a smartphone, we will need to use different interface. As part of the sample exercises in the earlier chapters, we used the "Serial Monitor" to capture debug messages as the 86Duino sketch (application code) executes on the EduCake. Serial port is a common interface for microcontroller to interface and communicate with external device. This chapter provides an introduction to serial communication and work through sample exercise showing how to work with EduCake's serial port.

To better understand serial communication, let's look at the different between serial port and parallel port data transfer. As shown in figure-1, serial port transfers data through a single communication line, where data is sent one bit at a time. Parallel port transfers data through multiple communication lines, where data is send in group of 8-bits at a time. While parallel port has greater communication bandwidth, serial port requires less communication link. Under the same clock rate, serial communication requires more time to send the same amount of data as a parallel communication. When transferring data over long distance using parallel communication, there are challenges dealing with signal synchronization.

Compare to the legacy computer, the current generation of computer is built with much higher clock rate which enable high speed data transfer via serial communication. Parallel port is no longer part of design specification for newer generation PC.

Common interfaces such as USB, SATA, RS232, RS422, RS485, I²C, IEEE 1394, PCI-E and Thunderbolt are based on serial data transfer. Comparing to the parallel port's DB25 connector, connectors for these new generation of serial data transfer connectivity is much smaller.

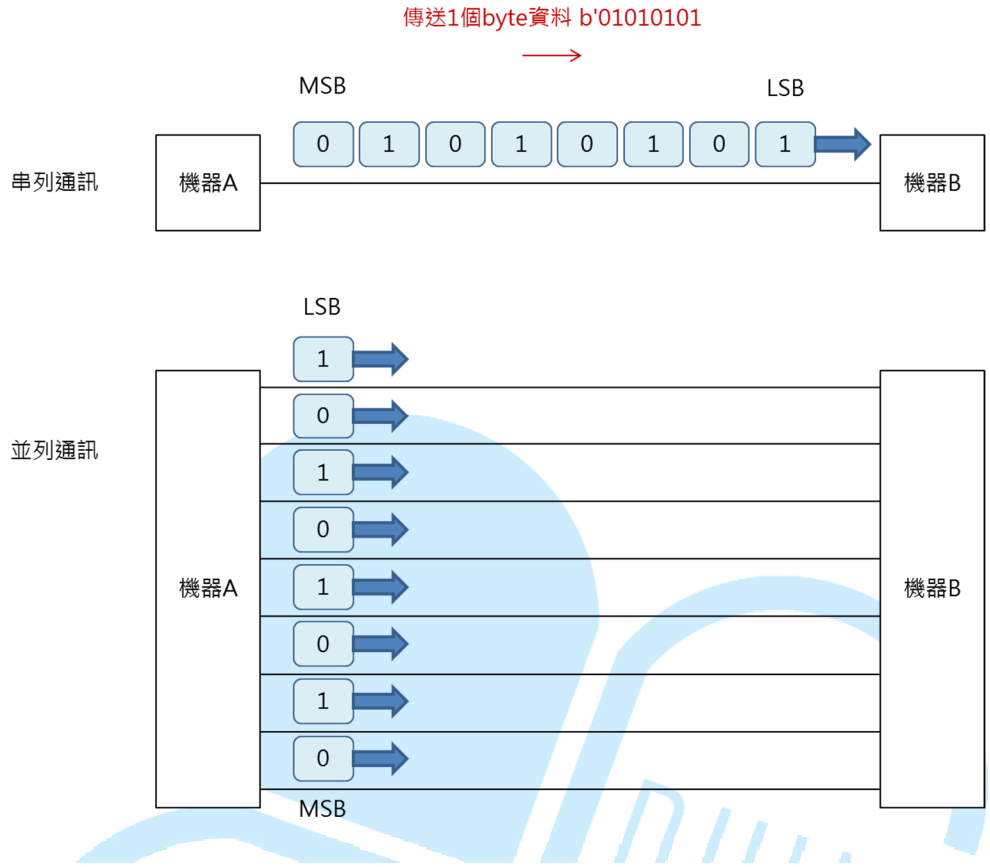


Figure-1. Serial and Parallel data transfer comparison

As shown in figure-1, while it's possible to use a single wire to support serial communication, most of the serial communication interfaces require multiple wires, where the additional wires are used to send control signal. For instance, the I²C interface has a SCL and SDA wires, where SDA is used for data transfer and SCL is the clock needed to synchronize data transmission. RS485 interface uses twisted pair wires, with D+ and D- signal lines, as a "balanced differential" transmission that support up to 4,000 feet cable length. While each of these serial communication standards, RS232, RS422 and RS485, has different electrical and mechanical properties, all of them are UART (Universal Asynchronous Receiver/Transmitter) interfaces.

The 86Duino EduCake hardware includes multiple UARTs with Tx and Rx signal lines (some documentation may refer to as TxD/RxD, which is the same as Tx/Rx), where Tx represent the Transmitting data line and Rx represent the receiving data line. With separate Tx and Rx signal lines, the hardware platform is able to transmit and receive data simultaneously to support "full duplex" communication mode. Some serial communication link is designed to support "half duplex" communication mode, where the device can perform one of the data transfer tasks at a given time, transmitting data or receiving data. I²C and RS485 are half-duplex serial interfaces. The common RS232 interface and 86Duino's UARTs are full-duplex capable. Figure-2 provides

graphical representation showing the different between full and half duplex communication.

When interfacing two UART devices, it's important to understand the devices' electronic signal. TTL level Tx/Rx signals operate in 5V range where the LOW/HIGH signal is between 0 ~ 5V. On the other hand, for RS232 interface, -3 to -15V represent signal HIGH(1) and +3 to +15V represent signal LOW(0). It's important to connect UART devices with compatible electronic signals. To connect a TTL UART to RS232, it's common to use a TTL to RS232 converter chip, such as the MAX232.

It's common for many hardware design to include FIFO (first in first out) buffer for each UART, a temporary buffer to hold transmitting and receiving data while the processor is busy with other tasks, as shown in figure-2. Depending on the hardware design, buffer size is different between devices, where some hardware is designed without UART buffer. Later in this chapter, we will work through sample exercise involving UART buffer.

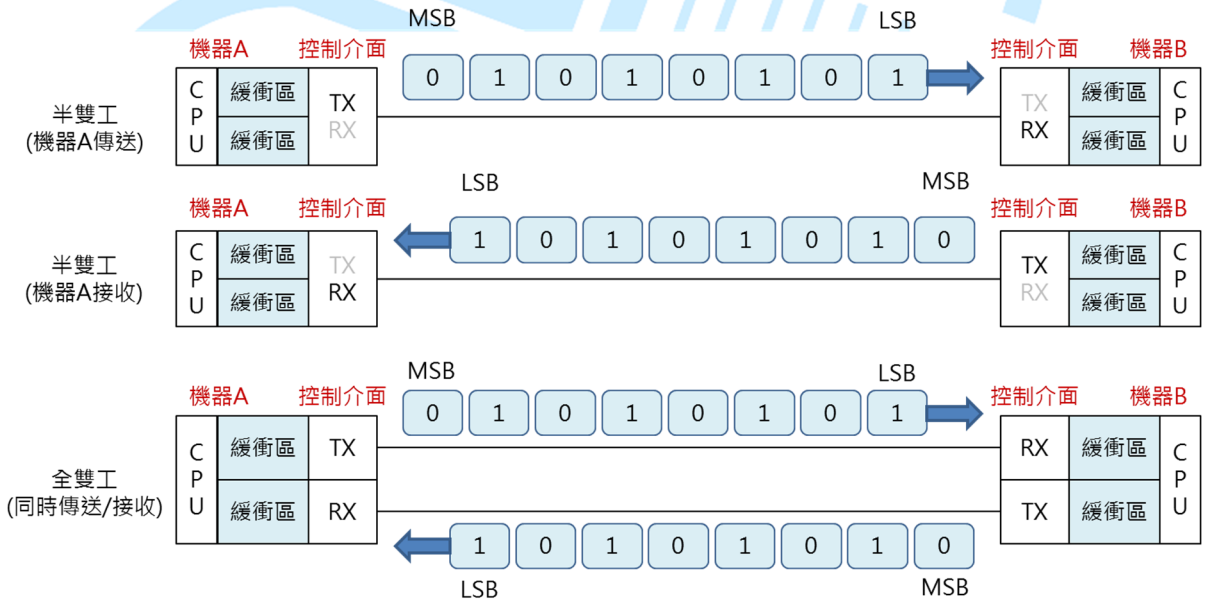


Figure-2. Full and Half duplex UART interfaces

One of the common feature for UART communication link is the communication protocol, where the transmitting and receiving devices follow the same parameters to send and receive data, to avoid error. As shown in figure-3, a serial communication packet (data frame) is sent with a start bit at the beginning, follow by the data, parity and end with a stop bit, where every single data packet is sent following this transmission rule. The start bit is used to signal the beginning of a new data packet to

the receiving node, an asynchronous communication. Other communication mode will be covered later in this chapter.

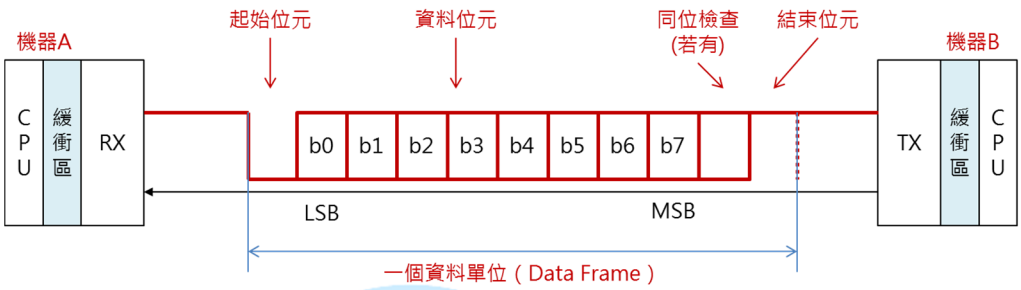


Figure-3. Communication protocol

It's common for many hardware design to include FIFO (first in first out) buffer for each UART, a temporary buffer to hold transmitting and receiving data while the processor is busy with other tasks, as shown in figure-2. Depending on the hardware design, buffer size is different between devices, where some hardware is designed without UART buffer. Later in this chapter, we will work through sample exercise involving UART buffer.

On your PC workstation, when an USB to serial adapter is attached and recognized, it shows up as a serial port. From the Device Manager screen, the USB to serial adapter shows up as "USB Serial Port", a virtual serial port as shown in figure-4, where you can right mouse click and select properties to view serial port settings.

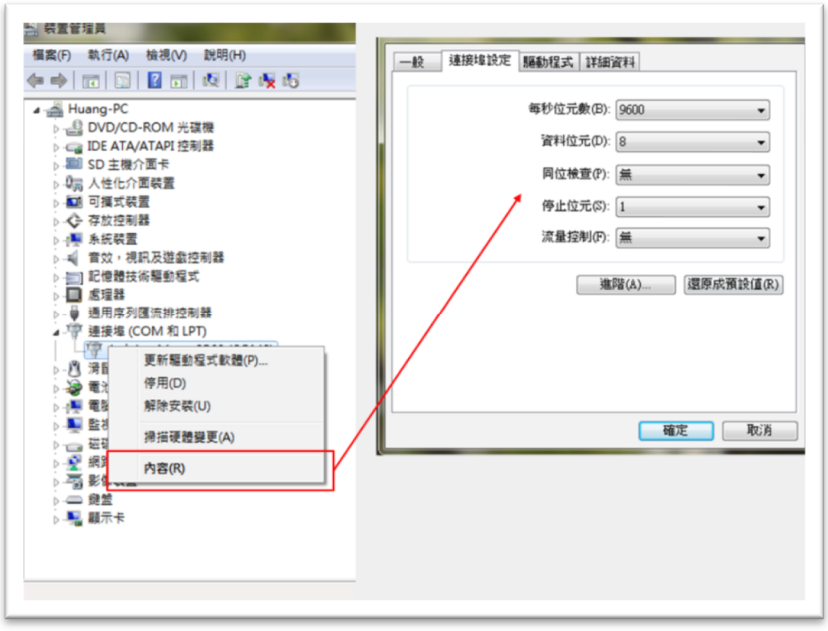


Figure-4. Device manager COM Port setting screen

Here are the available settings for serial port:

- Bits per second (Baud Rate)

  The setting controls the serial port transfer speed, in number of bit per second (bps). Since UART does not have data synchronization control signal and use the start bit to identify the beginning of a new data packet, both the transmitting and receiving serial points must operate at the same transfer speed. Otherwise, there will be error in receiving data. Using the Serial Monitor as an example, when the serial port communication is not configured properly, the receiving information on the Serial Monitor windows looks like a series of cryptic character that does not have any meaning. Some of the microcontroller boards with limited processing capability cannot support high transfer rate and need to be configured to support lower speed such as 4800 or 9600 Baud Rate. Some of the newer controller boards are designed with the ability to automatically detect Baud Rate (transfer speed).

- Data bits

  This setting designates the number of data bit in a transmission packet, where 8-bits which is the equivalent of 1 byte is one of the common settings. Data transmission starts with the least significant bit (LSB) and end with the most significant bit (MSB). For the exercise in this chapter, data bits is set to 8.

- Parity

  Interference or bad cabling can cause error in data transmission, where data bits "01010101" from the transmitting node arrived the receiving node as "01000101". Parity is a form of error checking for data transmission. The parity setting can be set to one of the following:

  1. No parity: Parity error checking is disabled. Parity bit is excluded from the transmission.
  2. Odd parity: When the data packet contains even number of bits that are high ("1"), a "1" is assigned to the parity bit, which change the total number of bits that are high (including the parity bit) to an odd number. When the data packet contains odd number of bits that are high, a "0" is assigned to the parity bit.
  3. Even parity: When the data packet contains odd number of bits that are high ("1"), a "1" is assigned to the parity bit, which change the total

number of bits that are high (including the parity bit) to an even number. When the data packet contains even number of bits that are high, a "0" is assigned to the parity bit.

4. Mark: This setting is rarely in use. Regardless of the information in the data packet, the parity bit is set to "1".
5. Space: This setting is rarely in use. Regardless of the information in the data packet, the parity bit is set to "0".

- Stop bits

  Stop bits is sent at the end of every data packet to signal the end of the current packet to the receiving node, which help synchronize the communication stream.

- Flow control

  When working with microcontroller with limited processing capability, it may not be able to process all incoming data in a timely manner without flow control, which result in loss of data. With flow control, when the receiving node completed receiving the current data packet, it sends a signal to the transmitting node to send the next packet of data. Pic

Following is a list of default settings that is commonly in use:

- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None
- Baud Rate: The transfer rate vary depending on the hardware and user preference.

Improper wiring and communication settings are common causes for serial communication. When there is a communication problem between two serial ports, check to make sure wiring are connected correctly on both ends and communication settings for both ports are the same.

If you are new to serial port communication, you may think develop a serial communication application for the 86Duino platform is a complicated process, having to deal with all of these communication parameters. In contrary, with help from the

86Duino IDE and support library, it's a simple and straight forward process to develop serial port application for 86Duino platform.

In this chapter, we will work through a series of tutorials showing how to access the EduCake's serial port using the "Serial Library". For these tutorials, the serial port is configured with the following:

- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None
- Baud Rate: 115200 bps

2. First tutorial: Serial.write()

As data is transmitting between two serial ports, the data stream is represented by a series of High/Low or "1" and "0" electronically. In software application, the data stream between two serial ports can be a binary data stream or character data stream, where binary data stream can be encoded in binary, decimal, hexadecimal and other formats. On the other hand, character data stream is encoded as a series of ASCII character. For example, a data packet with the value b'01000001 encoded in decimal is equal to 65. Whereas, the same data value is equivalent to 'A', when encoded as ASCII character.

In the exercise for this tutorial, we will transmit one byte of data between the EduCake and the development PC, in binary. Launch the 86Duino Coding 100 IDE and enter the following codes:

```
void setup()
{
  Serial.begin(115200);// Set Serial Port Baudrate
}

void loop()
{
  if(Serial.available())
  {
    int value = Serial.read();// Receive binary data
    delay(100);
    Serial.write(value + 1);// Transmit binary data
  }// end if(Serial.available()

}
```

The above sample program function as follow:

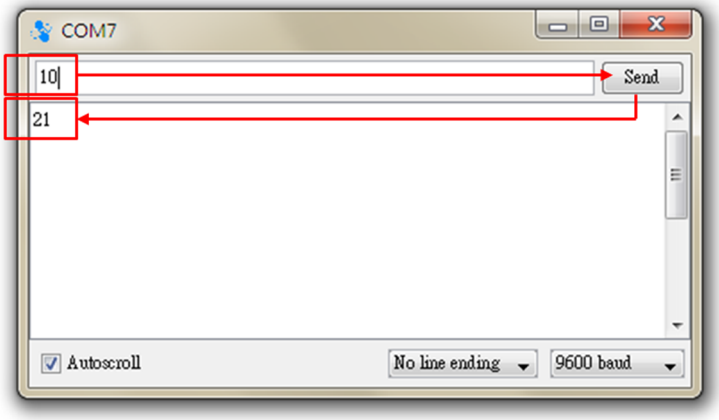
- The development PC transmit one byte of data to the EduCake through the serial port. After receiving the data, EduCake increases the value for the received data by 1 and transmit the data back to the PC.

When the above application executes, inside the setup () function, the Serial.begin (115200) function is called to set the serial port Baudrate. Inside the loop () function, the Serial.available () function is called to check whether there are data in the receiving buffer, and return the total byte of data in the temporary buffer.

The Serial.Read () function is called to read receiving data. Since the data is decoded as value, we can treat the value as int or unsigned int. As the receiving data is read, the value is increased by 1 and transmit back to the PC after 100 ms delay, using the Serial.write () function. The 100 ms delay is added to demonstrate data stream transmitted with delay (You can remove the delay to see the difference).

The Serial.write () and Serial.read () function transmit and receive one byte of data at a time. Make sure the transmitting value can be represented by one byte of data (0~255).

Since serial monitor send and receive data as ASCII characters by default, it's not used in this exercise. With serial monitor, a numeric value 10 is sent as two ASCII character, '1' and '0'. The ASCII character representation for '1' and '0' is 49 and 48. As the result, the above sample app will return 49+1=50 and 48+1=49, which interpret by the serial monitor as '2' and '1", instead of the numeric representation for (10+1), which is 11, as shown below.



Following is a terminal program UI showing results from the sample program:

- Data is transmitted as value encoded in Hexadecimal. When a value of 1 is sent, it receives a value of 2. When a value of 2 is sent, it receives a value of 3. When a value of A1 is sent, it receives a value of A2, and so on.

3. Second tutorial: Serial.print()

For the exercise in this section, data stream is sent as character. Enter the following code to the 86Duino Coding 100 IDE:

For the exercise in this section, data stream is sent as character. Enter the following

```
void setup()
{
  Serial.begin(115200);// Set Serial Port Baudrate
}

void loop()
{
if(Serial.available())
  {
    byte value = Serial.read();// Read receiving data as value
    delay(100);

    if(value == 'A')
    {
      Serial.write(32);// space
      Serial.write(65);// A
      Serial.write(66);// B
      Serial.write(67);// C
      Serial.write(68);// D
      Serial.write(69);// E
      Serial.write(10);// \n
    }
    else
{
      Serial.println(value);// 以數字方式回傳收到的資料
    }
  }// end if(Serial.available()
}
```

After the above code is downloaded to the EduCake, launch the Serial Monitor to communicate with the EduCake. When the above code is running on the EduCake, it function as follow:

- When the character 'A' is sent from the serial monitor, EduCake returns decimal values ('32', '65', '66', '67', '68', '69' and '10'). Serial monitor

interpret data from the EduCake as ASCII character and receives the ‘space’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ and ‘Line feed’ characters.

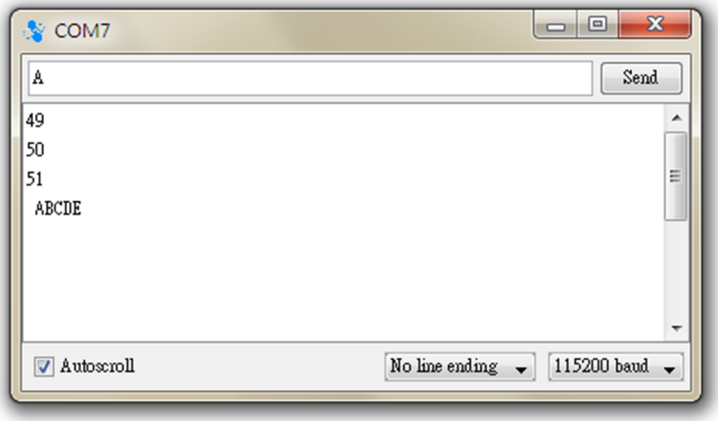
ASCII code 0 through 31 are control characters which do not have an associated display or printable character. Such as the Serial.write(10) function call in the above code, that send a ‘Line feed’ control code to the serial monitor, causing it to advance to a new line (In other programming languages, it’s common to see the ‘\n’ character combination, which also represent line feed.)

ASCII code 32 through 126 are character codes, where each of the has a display and printable character, such as ‘0’ through ‘9’, ‘a’ through ‘z’, ‘A’ through ‘Z’ and etc.

When the serial monitor send data other than the ‘A’ character, EduCake uses the Serial.printIn(value) function to return the received ASCII code as decimal value to the serial monitor. Similar to Serial.println(), the Serial.print() function transmit data as decimal value without adding a ‘line feed’ character at the end of each message. The Serial.println() function is used in previous chapter to send debugging messages.

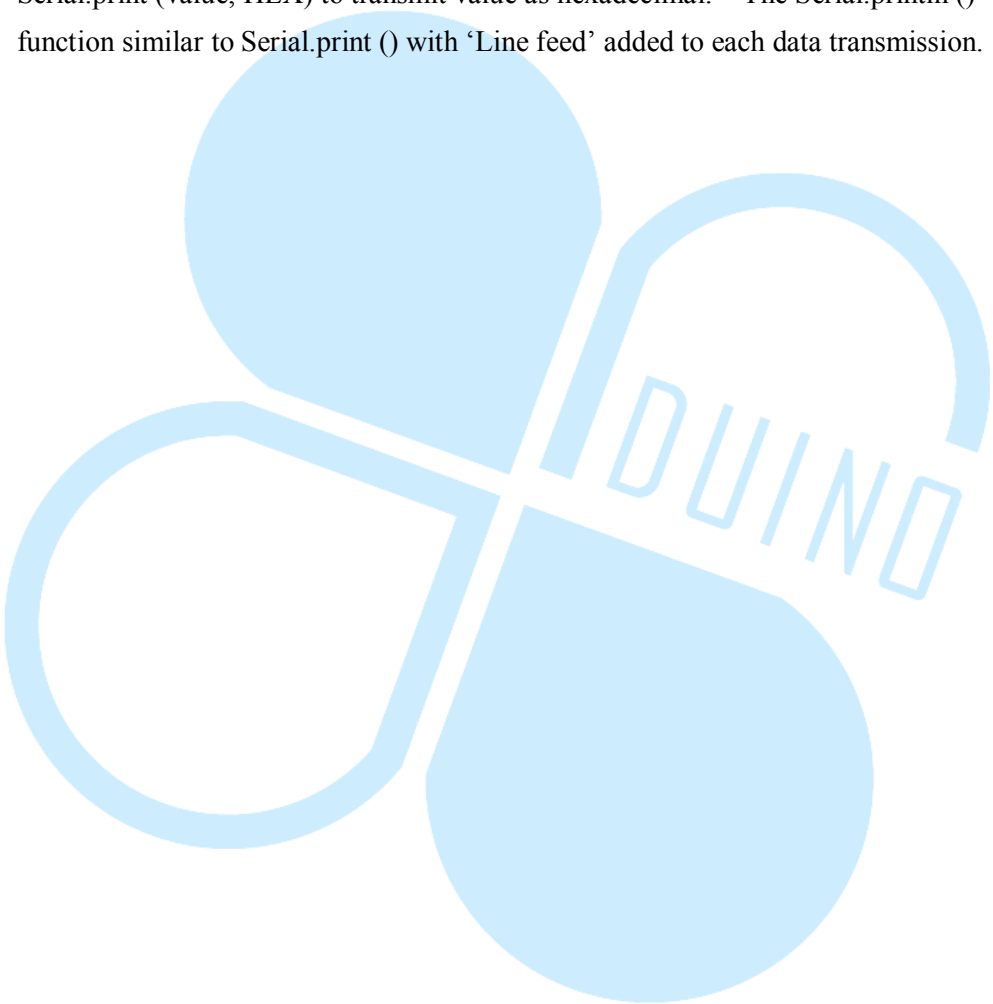
When the character ‘1’ is sent from the serial monitor, EduCake receives the message as ‘49’, using the function “byte value = Serial.read ()”, where 49 is the ASCII code value for the ‘1’ character. When EduCake transmit the value 49 using the Serial.println(49) function, serial monitor receive the data as two separate ASCII character, ‘52’ and ‘57’, which represent 4 and 9, and interpret the received data as ‘49’. This application can be used to observe displayable ASCII code character and view their corresponding decimal value.

As shown on the serial monitor screen below, when ‘1’, ‘2’ and ‘3’ are sent as separate messagse, EduCake returns ‘49’, 50’ and 51’, ASCII code for ‘1’, ‘2’ and ‘3’. When the ‘A’ character is sent, EduCake returns ‘ ABCDE’ follow by a ‘line feed’ character.



From the above example, we can see the difference between Serial.write () and Serial.print () functions. Serial.write () transmits data as byte, where the receiving node receives the data as value. Serial.print () transmits data as ASCII character, where the receiving node receives the data as ASCII character.

The Serial.print () function can be called with an additional parameter, such as Serial.print (value, DEC) to transmit Value as decimal, Serial.print (value, BIN) to transmit Value as binary, Serial.print (value, OCT) to transmit Value as Octal and Serial.print (value, HEX) to transmit value as hexadecimal. The Serial.println () function similar to Serial.print () with 'Line feed' added to each data transmission.



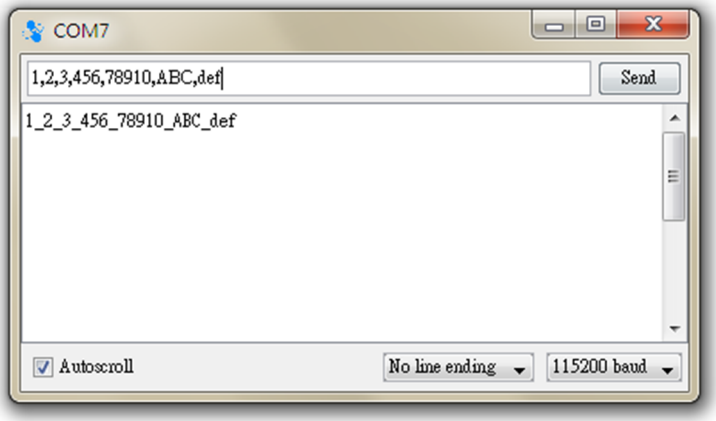
4. Third tutorial: Transmit/Receive Multiple Bytes of Data (as value)

In the previous exercise, the serial port transmit one byte of data at a time. In this exercise, let's look into transmitting/receiving multiple bytes of data. From the 86Duino Coding 100 IDE, enter the following codes:

```
void setup()
{
  Serial.begin(115200);// Set Serial Port Baudrate
}

void loop()
{
  int byteCount = Serial.available();// Number of data byte
                                     // currently in buffer
  if(byteCount > 0)
  {
    byte RX_buff[byteCount];// byte array declaration
                            // use to copy data from the buffer
    for(int i = 0; i < byteCount; i++)
    {
      RX_buff[i] = Serial.read();// Read data from buffer
    }

    for(int i = 0; i < byteCount; i++)
    {
      if(RX_buff[i] != 44)//
      {
        Serial.write(RX_buff[i]);
      }
      else
      {
        Serial.write(95);// '_'
      }
    }
    Serial.println();// Line feed
  }// end if(byteCount > 0)
}
```

After the above code is downloaded to EduCake, launch serial monitor to communicate with EduCake. When the above code is running on EduCake, it function as follow:

- When a string of characters (with multiple ',' characters in the string), is are sent from the serial monitor, EduCake return the string, where the ',' character is replaced by '_', as shown below.



In the loop () function, the line of code, "int byteCount = Serial.available ()" detect the length of data (number of bytes) currently in the receiving buffer. Next, the RX_buff byte array is declared, where the length of the array is set by the byteCount value. The RX_buff aray is used to store value from the receiving buffer. Inside the For loop, value from the receiving buffer is copied to the RX_buff array, using the Serial.read () function. Codes inside For loop can be accomplished by the Serial.readBytes (RX_buff, byteCount) function, which is much simpler.

The codes inside the For loop is written to demonstrate the Serial Port's FIFO (first in first out) data buffer. Each time the Serial.read () function executes, it reads one byte from the buffer, causing the data in the buffer to shift by one byte, to remove one byte of data from the buffer, which is read by the Serial.read () function.

In the second For loop, each time it iterates through the RX_buff byte array, when the current RX_buff array item data is not equal to 44, it simply transmit the data back to the serial monitor using the Serial.write (RX_buff[i]) function. If the RX_buff array item data is equal to 44, which is the ASCII code for the ',' character, it transmits value 95, which is the ASCII code for the '_' character, using Serial.write (95) function, to replace the ',' character.

Note: Codes execution inside the loop () function happen fairly quick. When data from the serial monitor transmits 10 bytes of data, at a slow pace, without an appropriate time delay, the byteCount = Serial.available () function may not capture all 10 bytes of data right away and may need to go through multiple loop to receive all 10 bytes of data. However, this situation does not affect the current exercise.

5. Third tutorial: Transmit/Receive Multiple Bytes of Data (as character)

EduCake's serial port provides other function to support application that communicates using character string. Enter the following codes to the 86Duino IDE.

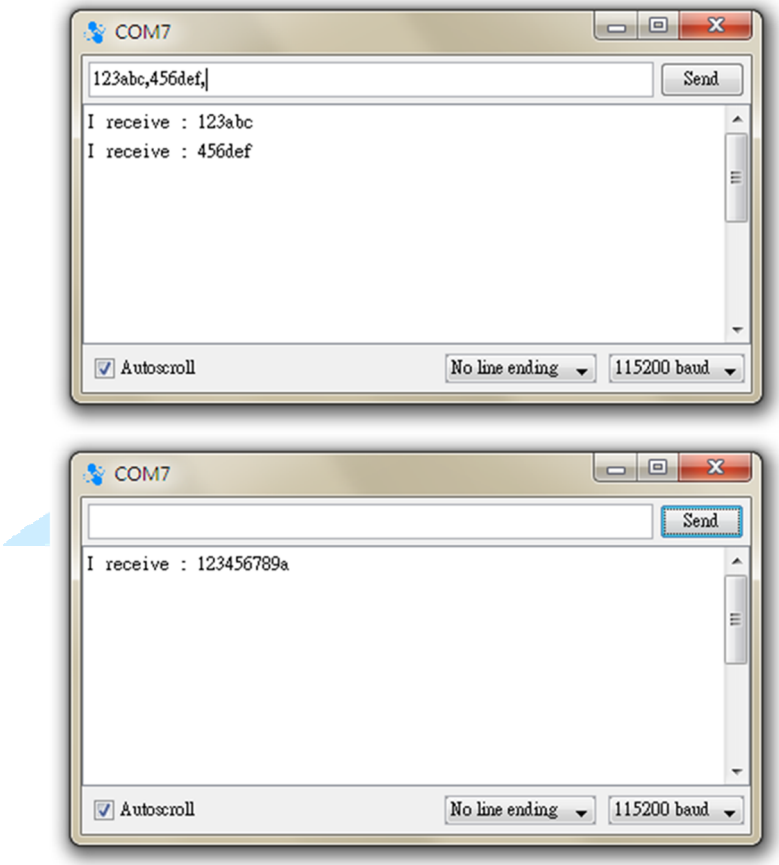
```
void setup()
{
  Serial.begin(115200);// Set Serial Port Baudrate
  Serial.setTimeout(2000);// Set time out duration for
                          // the Serial.readBytesUntil()
                          // function
}

void loop()
{
  char RX_buff[10];// Character array declaration

  // Retrieve data from buffer until the ',' or line feed is
  // detected
  int byteCount = Serial.readBytesUntil(',', RX_buff, 10);

  //Serial.print("byteCount : ");
  //Serial.println(byteCount);

  if(byteCount > 0)
  {
    Serial.print("I receive : ");
    for(int i = 0; i < byteCount; i++)
    {
      Serial.write(RX_buff[i]);
    }
    Serial.println();// Line feed
  }// end if(byteCount > 0)

  delay(100);//
  Serial.flush();// Clear buffer
}
```

After the above code is downloaded to the EduCake, launch the Serial Monitor to communicate with the EduCake. When the above code is running on the EduCake, it function as follow:

- When a string of characters, which include the ',' character, is sent from the serial monitor, EduCake receive the string of data and stop after a ',' character is detected.

Then, EduCake return the received data string with “I receive: “ added to the received string data, back to the serial monitor, such as “I receive: xxxx”.

When the string of data is sent without the ‘,’ character, EduCake return the character data string back to the serial monitor, with “I receive: “ added at the beginning, as shown below.



In the above code, the “Serial.readByteUntil (‘,’, RX_buff, 10)” function is used to read and copy data to the RX_buff array. Data is copied from the receiving buffer to RX_buff array until a ‘,’ character is detected or 10 characters are copied. Without setting an appropriate time out, when the received message does not contain the ‘,’ character and the length of the message is less than 10 characters, the application will halt at this line of code. The “Serial.setTimeout (2000)” function, inside the Setup () function, is use to set the maximum length of time (in ms), which is 2 seconds, for the Serial.readBytesUntil () function to wait for additional data.

The RX_buff character array is declared at the beginning of the loop () function, where the length of this array must equal or greater than the parameter for character length in the readBytesUntil function, “int byteCount = Serial.readBytesUntil(‘,’, RX_buff, 10)”. The readBytesUntil function reads multiple bytes of data from the receiving buffer.

When the value for byteCount, in “if (byteCount >0)”, is larger than zero data is retransmit to the serial monitor.

The Serial.print (“I receive : “) function can transmit a series of ASCII characters. The character string to be sent must be enclosed by the double quote (“) characters. The double quote (“) characters are not part of the data stream being send to the receiving node. Instead of using the Serial.print () function to send the “I receive : “ character string, you can send each individual character in separate function call, as shown below:

```
Serial.print('I');
  Serial.print(' ');
  Serial.print('r');
  Serial.print('e');
  Serial.print('c');
  Serial.print('e');
  Serial.print('i');
  Serial.print('v');
  Serial.print('e');
  Serial.print(' ');
  Serial.print(':');
  Serial.print(' ');
```

While the above codes deliver the same result, it’s not an efficient way to code.

To avoid left over data from the current operation, the last line of code call the Serial.flush () function to clear the buffer, both the transmitting and receiving buffer are cleared by this function. When calling the Serial.flush () function, it’s a good practice to delay the function by calling the delay (100) function to ensure the earlier Serial.write () and Serial.print () functions competed their tasks. You can try calling the Serial.flush () function without delay to see the impact, which will cause incomplete data transmission.

6. Fifth tutorial

In this tutorial, we will combine functions covered in the previous section with some new functions to create a more interest example, sending command to the EduCake through the serial interface. Enter the following codes to the 86Duino development IDE:

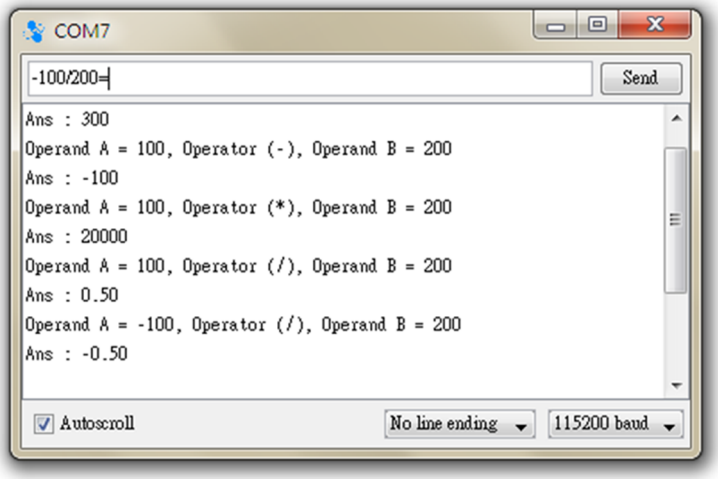
```
int Op_A = 0;// Variable declaration
int Op_B = 0;//

void setup()
{
  Serial.begin(115200);// Set Serial Port Baudrate

  // Command string format
  Serial.println("Please enter format : Operand A +(or -) Operand
B =");

  //  Set time out period for the Serial.parseInt () function
  Serial.setTimeout(1000);
}

void loop()
{
  // Parse integer value for Op_A from the buffer
  Op_A = Serial.parseInt();
  Serial.print("Operand A = "); Serial.print(Op_A);

  // Retrieve data from the buffer
  char opr = Serial.read();
  Serial.print(", Operator ("); Serial.write(opr);
Serial.print(")");

  // Parse integer value for Op_B from the buffer
  Op_B = Serial.parseInt();
  Serial.print(", Operand B = "); Serial.println(Op_B);
```

```
    if(Serial.read() == '=')
    {
      Serial.print("Ans : ");
      switch(opr)
      {
        case '+':
          Serial.println(Op_A + Op_B);
          break;

        case '-':
          Serial.println(Op_A - Op_B);
          break;

        case '*':
          Serial.println(Op_A * Op_B);
          break;

        case '/':
          Serial.println((float)(Op_A) / (float)(Op_B));
          break;

        default:
          Serial.println("illiegal operator !!!");
          break;
      }// end switch
    }// end if(Serial.read() == '=')
}
```

After the above code is downloaded to the EduCake, launch the Serial Monitor to communicate with the EduCake. From the serial monitor, you can send a data stream, in the following format (without blank space in between):

[1st integer] [math Operator] [2nd integer]

EduCake will perform the necessary calculation and return the result, as shown below.



In the above application, the code starts with two declarations, to declare Op_A and Op_B, as integer, which are used as operands for the math function. The setup () function is similar to the exercise in tutorial #4.

In the loop (), the code starts with the Serial.parseInt () function to read data from the buffer, parses integer value from the data and assign the value to the first operand, Op_A. As the Serial.parseInt () function reads data from the buffer, it parse the first numeric or group of numeric characters, prior to reaching a non-numeric character, into an integer value. For example, when the '1', '2', '3', '+' and '1' ("123+1") data stream is read, the first 3 numeric characters, prior to the '+" character, are parsed into a single integer value of 123 (One hundred twenty three). If the data stream begins with a '-' character, such as '-', '1', '2', '3', '+' and '1' ("-123+1"), a negative integer value of -123 is parsed.

Next, the Serial.read () function read the next character, which represent the math operator and assigns the value to the opr variable, which will be used to select an associated math function. After the Serial.read () function, the Serial.parseInt () function parses the remaining data as integer value as the 2nd operand, Op_B.

Finally, the " if(Serial.read () == '=' " statement compare the next character read from the buffer against the '=' (equal) sign. If the next character is '=', the opr variable (which represent the math operator) is used in the switch-case statement, to launch one of the 4 math functions (addition, subtraction, multiplication and division). Each of the 4 math function will be executed with Op_A and Op_B as the required operands. Since result from the addition, subtraction and multiplication math function based on two integer value is also an integer, special handling is not needed. Since the division math function between two integer operands may create a floating point number (such as dividing 3 by 2 will yield 1.5), both operands (Op_A and Op_B) are converted to floating point number for the division math function. Otherwise, the fractional value for the result from a division math function will be truncated and lost.

Instead of using integer based operands, you can declare Op_A and Op_B as floating point numbers and use the Serial.parseFloat () function in place of the Serial.parseInt () function.

For more information about the functions used in this exercise, visit the following URL:

http://arduino.cc/en/Reference/Serial

http://arduino.cc/en/Serial/ParseFloat

http://arduino.cc/en/Serial/ParseInt

http://arduino.cc/en/Serial/ReadBytes

7. Sixth tutorial: GPS

In this last exercise, we will work through a simple GPS application. GPS (Global Positioning System) was initially created by US Department of Defense. GPS is a satellite navigation system covering 98% of the Earth's surface, which is widely used for navigation, tracking a device's location and traveling speed. In addition, GPS system also provide precision time service. We can use GPS to provide timer function and set correct time for EduCake. It's not within this tutorial's intention to dig deep into GPS. If you are interested to learn more about GPS, search the Internet or query Wikipedia using the "GPS" and "Global Positioning System" key words.

http://www.alibaba.com/product-detail/Ublox-NEO-6M-GPS-Module-with_1663977022.html

To work through the exercise, you will need a GPS module. There are quite a few variety of GPS modules available in the market, built with RS-232 (UART), I²C or SPI interface. The NEO-6M GPS module, with TTL UART interface, is used for the exercise in this section, as shown in figure-1.

Figure-1. GPS module with TTL interface

The NEO-6M is a low cost module with a simple interface which includes:

- NC (No connection)
- VCC (3.6 to 5.2V power source, which can be attached to EduCake's 5V output)
- RX (Receive)
- TX (Transmit)
- Ground

- PPS (Timing pulse in 1 second interval)

There are a few things you need to know about GPS module:

1. **Power on (cold start)**: When power on, the GPS module needs to be initialized, find an available satellite and uses the satellite signal to calculate the current information, which may take 20 to 30 seconds and longer. Most GPS navigation devices need to re-generate map data and routing during power on, which will take much longer than 30 seconds to complete the initialization process during power on. Most GPS navigation devices need to capture signals from 3 or more satellite in order to function and may take a few minutes to complete the initialization process.
2. **Warm reset:** When warm reset (waking up from standby), a GPS navigation device may take just a few second to be ready.
3. **Re-capture lost satellite signals:** When traveling in metropolitan city where tall buildings are situated closely, in underground parking structure, driving under a bridge or through a tunnel, the GPS device is not able to receive signal from the satellite and has to re-capture satellite signal. When re-capturing satellite signals, the GPS device compares the new data with the data stored in the device's buffer and make the necessary adjustment or correction. Typically, it takes just a few seconds to re-capture lost satellite signals.

To use the Neo-6M GPS module on EduCake, connect the module's 5V, GND and TX signals to the EduCake, as shown in figure-2. Connections on the EduCake are clearly marked. To acquire accurate data that involves timing, such as calculating traveling speed, you can use the PPS pin from the Neo-6M module which provides a 1-second timing signal.

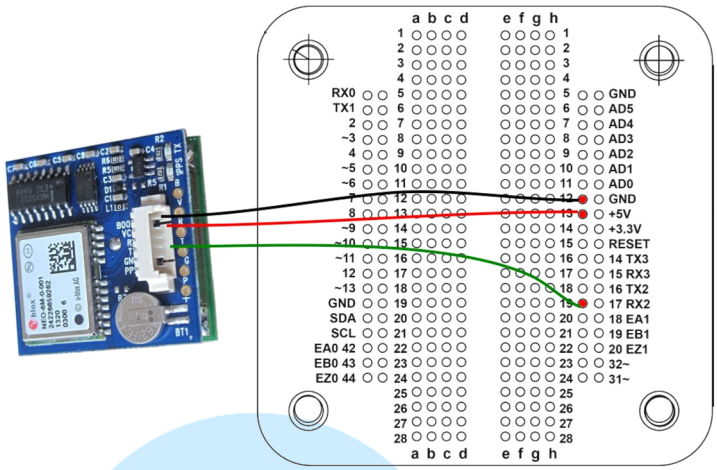


Figure-2. GPS 模組接線方式

After connecting the GPS module to the EduCake, enter the following codes to the 86Duino IDE:

```
void setup()
{
  Serial.begin(9600);
  Serial2.begin(9600);  //GPS attached to 2nd serial port
  Serial.println("Begin");
  delay(2000); // Delay needed for GPS to initialize
}

void loop(){
   char ch;
   if ( Serial2.available()) // check for incoming data
   {
    Serial.println("Get:");
    while (Serial2.available()) //Loop to receive data
    {
     ch=Serial2.read();
     Serial.print(ch); // Transmit received data
    }
   }
}
```

The above codes capture data from the GPS module from the 2nd serial port and re-transmit the received data to the serial monitor through the 1st serial port, which enables us to see the raw data from the GPS module, such as the following:

$GPGGA,073849.00,2301.37777,N,12012.94773,E,1,09,0.098,41.2,M,-2.5,M, ,*42

We need to write code to parse relevant data from the data stream captured from the GPS module. Data stream from the GPS module typically adhere to a certain NMEA format, which refer to as a sentence. GPS module from different company may adopt different data format. Each individual piece of data within the NMEA sentence above is separated by a comma (“,”), as described in the following table:

$GPGGA	Indicate the beginning of sentence based on the GGA standard.
073849.00	Current time based on UTC time zone in hhmmss.sss format, where hh represent hour, mm represent minute, ss represent second and sss represent fraction of a second, which is equal to: – 07:38:49:00 (07 AM, 38 Minutes, 49 seconds)
2301.37777	Latitude data in ddmm.mmmm format.
N	North (N) and South (S) hemisphere indicator.
12012.94773	Longitude data in ddmm.mmmm format.
E	East (E) and West (W) hemisphere indicator.
1	Position fix indicator, where: – 0 = Fix not available or invalid – 1 = GPS SPS mode, fix valid – 2 = Differential GPS, SPS mode, fix valid – 3 = GPS PPS mode, fix valid
09	Number of satellites used. Number less than 3 may yield inaccurate position data.
0.098	Horizontal Dilution of Precision (HDOP), which indicates the accuracy of the horizontal position data. Working with higher number of satellites increase accuracy. When traveling between tall buildings which reflect satellites signal deteriorate data accuracy.
41.2	Height from sea level
M	Measurement unit. (M = meter)
-2.5	Geoid separation
M	Measurement unit. (M = meter)
*42	checksum

In addition to the above data stream, which includes information needed by most GPS application, there are other GSP data format, such as the GLL and RMC message format:

- $GPGLL, 2301.37777,N, 12012.94773,E, 073849.00,A,A*62

  GLL message format is shorter than GGA and includes latitude, longitude and UTC time information.

- $GPRMC,073141.00,A,2301.37777,N,12012.94773,E,0.016,0,190214,,,A*77

  RMC message format is somewhat similar to GGA, where the 3rd variable, which is “A” in the above message, is used to indicate whether the data is useful, where “A” represent active and “V” represent void and indicates the data is not usable, which may be caused by the lack of satellites signals. The value after the longitude data, “0.016” represent traveling speed measure in knots (1 knots = 1.15078 miles/hour or 1.852 km/hour). The next variable, “0”, track the angle in degree. The next variable, “190214”, represent the

date, which is the 19th of February in 2014. The next variable, which is blank in the above data message, represent magnetic variation and direction. The last variable, "A*77", is checksum.

There are many different variation of GPS message format, which is not within the scope for this tutorial. If you are interested to learn more, search the Internet using the "NMEA" and "NMEA message types" key words. NMEA is short for National Marine Electronics Association.

To help visualize the data, we can use a 16x2 LCD module to display the data.

```
#include <LiquidCrystal.h>

// Connect rs to digital 3
// Connect rw to digital 4
// Connect enable to digital 5
// Connect d4, d5, d6, d7 to digital 6、7、8、9 respectively
LiquidCrystal lcd(3,4,5, 6,7,8,9);
int LightBri = 32;  //Contrast control: digital 32

String data="";
int mark = 0;
boolean Start_Flag=false;
boolean valid=false;
String GGAUTCtime,GGAlatitude,GGAlongitude;
String GPStatus,SatelliteNum,HDOPfactor,Height;
String PositionValid,RMCUTCtime,RMClatitude;
String RMClongitude,Speed,Direction,Date,Declination,Mode;

void setup()
{
  pinMode(LightBri, OUTPUT);
  analogWrite(LightBri, 40);  // Set display contrast
  lcd.begin(16,2);
  lcd.clear();
  lcd.setCursor(0,0);  // Set cursor position
  lcd.print("DMP Demo");

  Serial.begin(9600);
  Serial2.begin(9600); // Connect to GPS module
```

```
 Serial.println("Begin");

 // GPS module needs time to initialize and detect
 //  Satellite signals.
 // Without sufficient number of satellites, the module
 //  cannot provide accurate data.
 // Depend on the module, set delay between 1 to 10 seconds
  delay(2000);
 }

void loop()
{
  while (Serial2.available()> 0)
  {
    if(Start_Flag){  //
      data=readGPS(); // Detect GPS message format, such as
                      //  GPGGA, GPGSV & etc.
      Serial.println(data);
      if(data.equals("GPGGA")){
        // Check for GGA data format
        // Each variable is separated by a comma.
        // Call readGPS function to read each variable
        // in the order from the data message.
        GGAUTCtime=readGPS();
        GGAlatitude=readGPS();
        GGAlatitude+=readGPS();
        GGAlongitude=readGPS();
        GGAlongitude+=readGPS();
        GPStatus=readGPS();
        SatelliteNum=readGPS();
        HDOPfactor=readGPS();
        Height=readGPS();
```

```
                Start_Flag =false;

              if (GPStatus>0) // 必須是有效定位才可以設定 valid=true，以
利後面判斷
                 valid=true;
              else
                 valid=false;
              data="";
            }
            else if(data.equals("GPGSA") ){  // GSA 封包跳過不處理
              Start_Flag =false;
              data="";
            }
            else if(data.equals("GPGSV")){  // GSV 封包跳過不處理
              Start_Flag =false;
              data="";
            }
            else if(data.equals("GPRMC")){
              // 抓到 RMC 的封包，這裡就是參照前面談過的表格內容
              // 每隔一個逗號都是一項資訊，呼叫 readGPS 函數按順序把欄位內容
讀取出來
              RMCUTCtime=readGPS();
              PositionValid=readGPS(); // 讀取到 A 才是有效封包
              RMClatitude=readGPS();
              RMClatitude+=readGPS();
              RMClongitude=readGPS();
              RMClongitude+=readGPS();
              Speed=readGPS(); //速度
              Direction=readGPS(); //方向
              Date=readGPS(); // 日期
              Declination=readGPS();
              valid=true;
```

```
        Start_Flag =false;
              data="";
            }
            else if(data.equals("GPVTG")){
              Start_Flag =false;
              data="";
            }
            else{
              Start_Flag =false;
              data="";
            }
          }

          if(valid){
            if(PositionValid=="A")
            {
              Serial.println("Get a valid position");
              output(); // 印出前面抓到的正確訊息到 LCD 和 serial port
            }
            else
              Serial.println("Not a valid position");
            valid=false;
            PositionValid="";// 重新把檢查旗標清空，才能確實判斷下次是否有
    抓到正確資訊
          }
          if(Serial2.find("$")){ // 若從 GPS 來的資料裡面含有$，代表開始
    抓到資訊了
            Serial.println("Get GPS string...");
            Start_Flag =true; // 就可以把開始旗標設定為 true 準備開始處理
          }
        }
      }
```

```
    //   從整串的 GPS 傳回字串裡面讀取被逗號隔開的資料
    //   範例 $GPGGA,073849.00, 2301.37777,N,
12012.94773,E,1,09,0.098,41.2,M,-2.5,M, ,*42
    String readGPS(){
      String value="";
      int temp;
    startRead:
      if (Serial2.available() > 0)
      {
        temp=Serial2.read(); //  一個字一個字讀取出來
        if((temp==',')||(temp=='*')) //需要一直讀取，直到讀到逗號或是*
號才是一段完整資訊
        {
          if(value.length()>0)  // value 裡面的長度比 0 大代表有確實讀到
資訊，直接傳回
            return value;
          else
            return "";  // 否則就傳回空字串
        }
        else if(temp=='$')  // 讀到$代表這是一段訊息的最開頭，準備開始截
取資訊
          Start_Flag =false;
        else
          value+=char(temp); // 把每一個讀到的字都加入 value 字串
      }

      // 這個 delay()的數值需要稍大或稍小調整，1~3 都可以
      // 等待一下，避免讀取速度過快，GPS 根本沒資訊傳回
      delay(1);
      goto startRead; // 跳回 if 開頭，不斷的讀取，直到確認能讀到完整資訊
    }
```

```
void output() // 輸出相關訊息的程式都包裝在這裡
{
  // 底下這一大段 serial.print 純粹是用來把訊息印出來看
  Serial.print("Date:");
  Serial.println(Date);
  Serial.print("UTCtime:");
  Serial.println(GGAUTCtime);
  Serial.print("Latitude:");
  Serial.println(GGAlatitude);
  Serial.print("Longitude:");
  Serial.println(GGAlongitude);
  Serial.print("GPStatus:");
  Serial.println(GPStatus);
  Serial.print("SatelliteNum:");
  Serial.println(SatelliteNum);
  Serial.print("HDOPfactor:");
  Serial.println(HDOPfactor);
  Serial.print("Height:");
  Serial.println(Height);
  Serial.print("Speed:");
  Serial.println(Speed);
  Serial.print("Direction:");
  Serial.println(Direction);
  Serial.print("Mode:");
  Serial.println(Mode);

  // 顯示相同的訊息到 LCD 上面
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("time:");
  lcd.print(GGAUTCtime);
  lcd.setCursor(0,1);
  lcd.print("Satellite:");
  lcd.print(SatelliteNum);
```

```
        delay(600); // 因為 16*2 LCD 螢幕只有兩行，需要頓一個時間以後清除
畫面，重新顯示新資訊
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("lat:");
        lcd.print(GGAlatitude);
        lcd.setCursor(0,1);
        lcd.print("lon:");
        lcd.print(GGAlongitude);

        delay(600); // 再度停頓一段時間後清畫面
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Height:");
        lcd.print(Height);
        lcd.print("m");
        lcd.setCursor(0,1);
        lcd.print("Speed:");
        lcd.print(Speed);
        lcd.print("km");
        delay(550); // 最後還是要頓一段時間，避免立刻回主迴圈，被清掉畫面，
最後兩行資訊來不及看到
    }
```