EduCake and Infrared Transceiver

1. Infrared Introduction

In previous chapters, we covered interfacing 86Duino EduCake to sensors and other device using UART Serial Port, I²C and etc. which require physical wire connection. In this chapter, we will explore a wireless connectivity option using infrared with EduCake and work through exercises to perform transmitting and receiving function.

Infrared is a part of the electromagnetic spectrum and is the dominating technology used in remote control for TV, DVD players, audio system and remote control for variety of other consumer and commercial products.

Light spectrum in the range of 400nm to 700nm is visible to human. Just below the visible 400nm spectrum is Ultraviolet (UV), ranging from 10nm to 400nm. Infrared (IR) is the light spectrum just above 700nm, ranging from 700nm to 1mm, as shown in Fig-1, electromagnetic spectrum diagram:

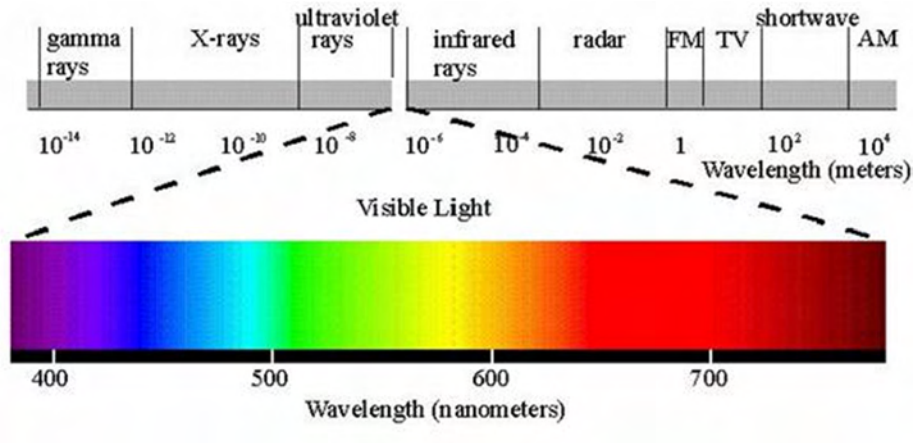


Fig-1: Electromagnetic spectrum

Within the IR spectrum, there are different categories of IR, such Near-infrared, Short-wavelength infrared, Mid-wavelength infrared, Long-wavelength infrared and Far-infrared. To learn more about infrared and electromagnetic spectrum, visit the following URLs:

http://en.wikipedia.org/wiki/Electromagnetic_radiation

http://en.wikipedia.org/wiki/Visible_spectrum

http://en.wikipedia.org/wiki/Infrared

http://en.wikipedia.org/wiki/Ultraviolet

Although IR is not visible to us, it's all around us. In addition to the common remote control for TV, audio entertainment system, home appliances and toys, IR is also used in imaging and other type of application.

The content in this chapter talks about using IR as wireless communication link to transmit and receive data and control signals, using IR transmitting and receiving devices as shown in Fig-2 and Fig-3.

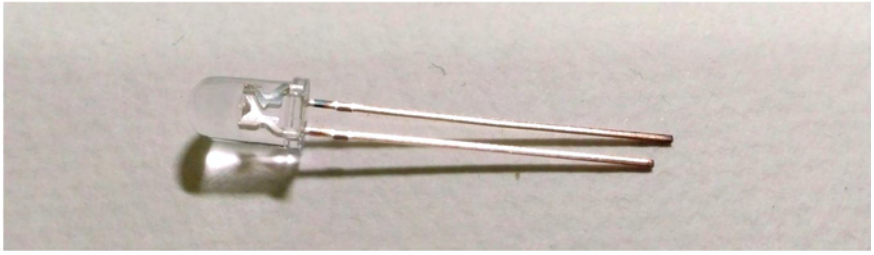


Fig-2: Infrared transmitter

Comparing to Infrared transmitter, Infrared receiver construction is more complex, which includes signal processing circuitry in addition to the Infrared

receiving mechanism. Infrared receiver typically has 3 signal pins, power-input, ground and signal-output, as shown in Fig-3.

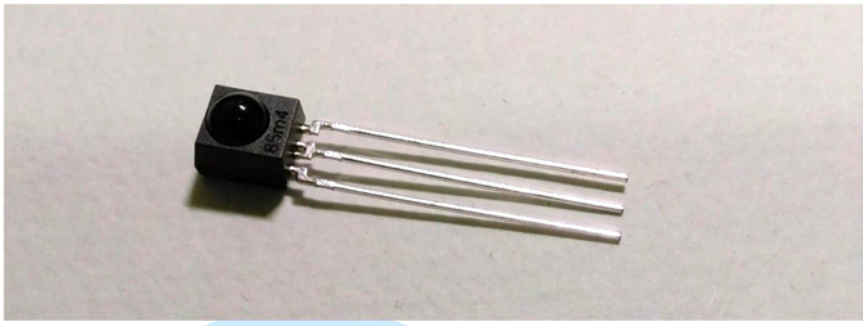


Fig-3: Infrared receiver

There are many different sources of electromagnetic waves within the Infrared range around us, to avoid interference and insure the Infrared receiver can receive the correct information, the transmitter modulate the transmission signal at a specific frequency. On the receiving end, a signal processing circuitry is used to filter unwanted signal and go through a demodulation step to interpret the received data. Fig-4 below demonstrate a typical Infrared transmit and receive scenario.

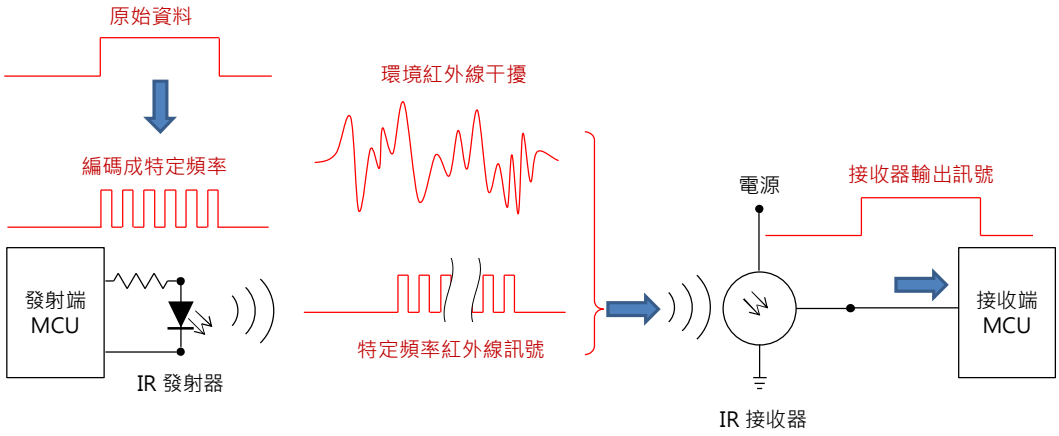


Fig-4: Typical Infrared transmit and receive process.

In order for the transmitting and receiving function to work, as shown in Fig-4, both the Infrared transmitter and receiver must function in the same modulation/demodulation frequency. Here are some of the common modulation/demodulation frequencies in use: 36 KHz, 38 KHz, 40 KHz & 56 KHz.

Please note the modulation/demodulation frequency is different from the electromagnetic spectrum frequency. Modulating the IR frequency help the

receiver distinguish the transmitter's signal from other infrared interference, where the receiver demodulate received signal using the same frequency as the transmitter. In addition to the modulation/demodulation process, some form of communication protocol can be implemented to further enhance the ability to communication more complex data and commands, by simulating different data bit's on/off condition.

Using a Sony Infrared remote control as example, which uses 40 KHz as the modulating carrier frequency. When one of the command button is pressed, it causes the remote control to transmit modulated Infrared signal with different data sequence that enable the receiver to identify which button is pressed, such as the modulated Infrared signal shown in Fig-5.

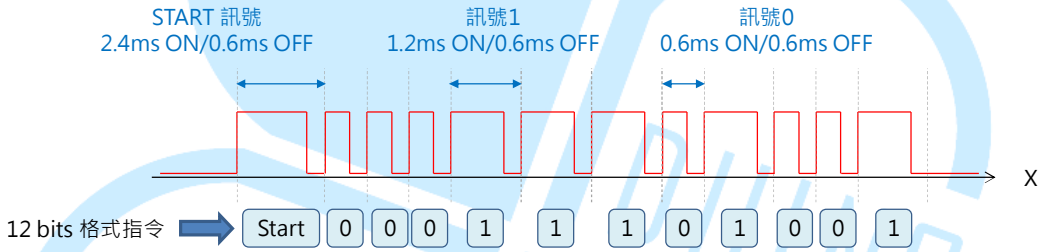


Fig-5: Infrared signal modulated around a predefined communication protocol

To learn more about Infrared communication protocol and related subjects, visit the following URLs:

http://www.righto.com/2010/03/understanding-sony-ir-remote-codes-lirc.html

http://users.telenet.be/davshomepage/sony.htm

http://www.cypress.com/？docID=46755

http://www.sbprojects.com/knowledge/ir/sirc.php

For the exercises in this chapter, we use an 86Duino EduCake with RPM6938, a low-cost 920nm Infrared receiver, which you can easily find in the DIY maker market, as shown in Fig-6.

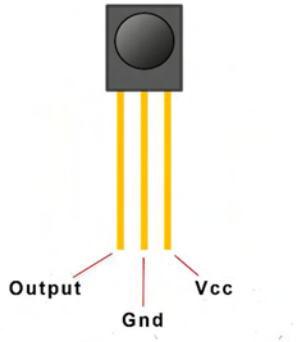
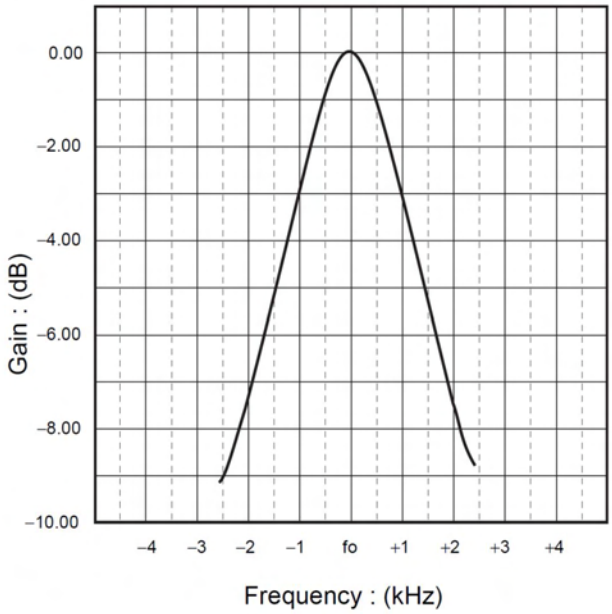


Fig-6: Infrared receiver

According to the datasheet for the RPM6938 chip, Vcc requires 5V power source, this chip has an effective horizontal receiving angle spanning 70 degree and effective vertical receiving angle spanning 60 degree, with the sensor's output signal set to High when no signal is detected and set to Low when signal is present. For this receiver, carrier frequency at 37.8 KHz yield the best receiving signal, and is able to receive signal at frequency range slightly higher or lower than 37.8 KHz.



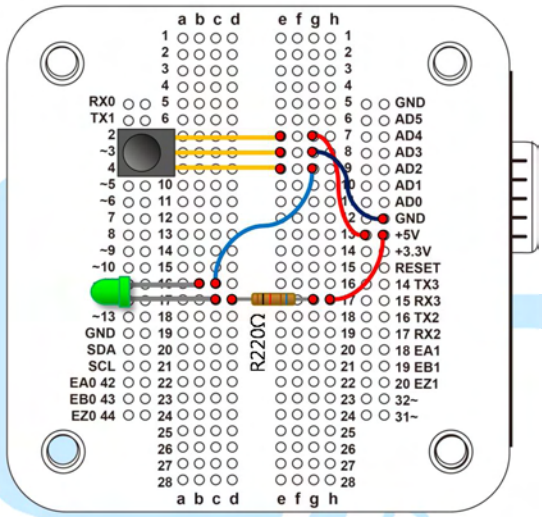
There are different variety of Infrared receiver in the market. If you are purchasing an Infrared receiver different from RPM6938, in addition to the carrier frequency, you need to pay attention to the receiver type. The output signal for some of the receiver is reversed. There are some receiver do not have

built in capability to demodulate the received signal. To avoid unnecessary headache, check the datasheet carefully when selecting an Infrared receiver.

Wiring standard for Infrared transmitter is similar to LED, the longer leg is the (+) positive signal and the shorter leg is the (-) negative signal.

2. First exercise: Testing Infrared function

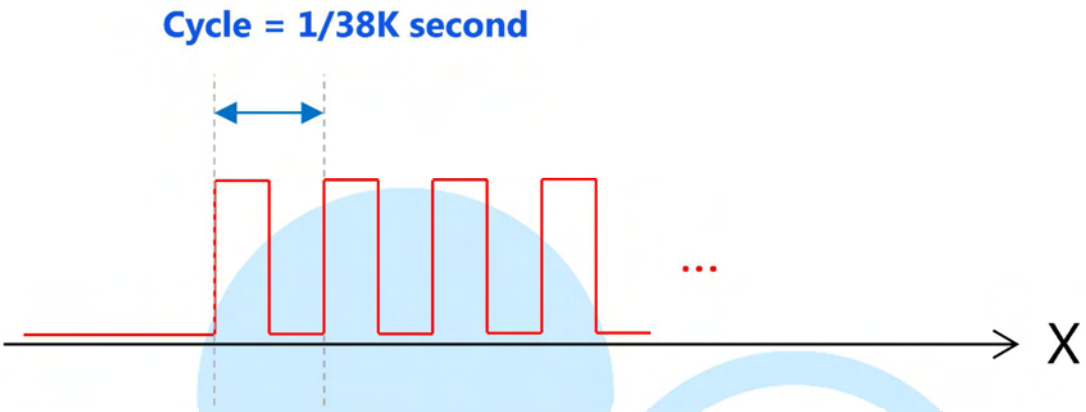
In this first exercise, we will work through simple steps to test Infrared transmit and receive function without coding, to check and make sure the components are working with circuitry as shown in the following figure:



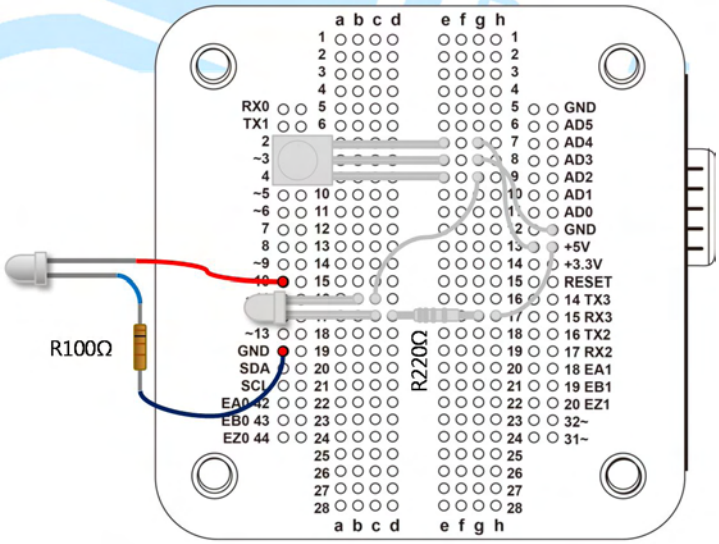
The infrared receiver's power source and ground are connected to +5V and GND on the EduCake. The infrared receiver's signal output pin is connected to the LED's negative pin. The LED's positive pin is connected to +5V on EduCake with a 220 Ohm resistor in series. When the infrared receiver does not detect any signal, the signal output pin is HIGH, which will keep the LED in off condition. When signals are detected, the receiver's signal output pin goes LOW and turn on the LED. Using one of the remote control you have for your TV or home entertainment system, which typically transmit Infrared signals that modulate around 38 KHz. As the infrared receiver, in the above circuit, receives signals transmitted by the remote control, the LED will blink as you press the buttons, which provides a way for us to check and confirm the infrared receiver is functioning as expected.

Instead of using a remote control to transmit infrared signal, we can use an infrared transmitter (which physically looks similar to an LED) to accomplish similar objective, to check and confirm the infrared receiver is functioning, using the following circuit along with a simple program.

In order for the receiver to detect its signals, the infrared transmitter must transmit signal at the same modulation frequency as the receiver, which is 38 KHz for this example, as shown in the following frequency waveform.



To generate the above waveform, we can use the digitalWrite() function with the following circuit:



In the above circuit, an infrared transmitter is added to the previous circuit. The transmitter's positive pin is connected to pin 10 on the EduCake and the negative pin is connected to GND with a resistor in series. Since infrared transmitter requires higher current than an LED to function, a 100 Ohm resistor is connected in series instead of 220 Ohm resistor typically use for an LED.

From the 86Duino Coding IDE, enter the following codes:

```
int IR_pin = 10;

void SendIR( )
{
  for( int i = 0; I < 800; i++ )
  {
    // ON + OFF 26us ~= 38.4kHz
    digitalWrite( IR_pin, HIGH );
    delayMicroseconds( 13 );

    digitalWrite( IR_pin, LOW );
    delayMicroseconds( 13 );
  }
  Serial.println( "IR send." );
}

void setup( ) {
  Serial.begin( 115200 );

  pinMode( IR_pin, OUTPUT );
}

void loop( ) {
  SendIR( );
  delay( 1000 );
}
```

In the above code, within the `void SendIR()` function, `digitalWrite()` function is called with `delayMicroseconds()` to general PWM waveform. Cycle for a 38 KHz waveform is about 26 µs, which takes 13 µs for the signal to go from HIGH to LOW or LOW to HIGH.

When you compile and run the above code, it modulate and transmit infrared signal at 38.4 KHz every other second, and causes the LED in the infrared receiver's circuit to blink.

The above code sample demonstrated a simple approach to test infrared transmitter and receiver. Using hardware generated PWM waveform, we can develop similar code to generate more granular and precise signals. As part of the 86Duino EduCake development environment, the `TimerOne` library is provided which includes a collection of routines for configuring the 86Duino's internal high-precision timer and provides a way to easily control the I/O pin's PWM output signals.

Using the following sample codes, you can produce hardware generated PWM signal, which is more precise than software generated signal:

```
#include "TimerOne.h"
int IR_pin = 10;
void SendIR( )
{
  Timer1.pwm( IR_pin, 512, 26 );// pin, duty (512=50%),
period(us)
  delay( 20 );
  Timer1.disablePwm( IR_pin );
  delay( 20 );
  Serial.println( "IR send." );
}
void setup( ) {
  Serial.begin( 115200 );
  pinMode( IR_pin, OUTPUT );
  Timer1.initialize( 26 );// TimerOne initialize,
period(us)
}
void loop( ) {
  SendIR( );
  delay( 1000 );
}
```

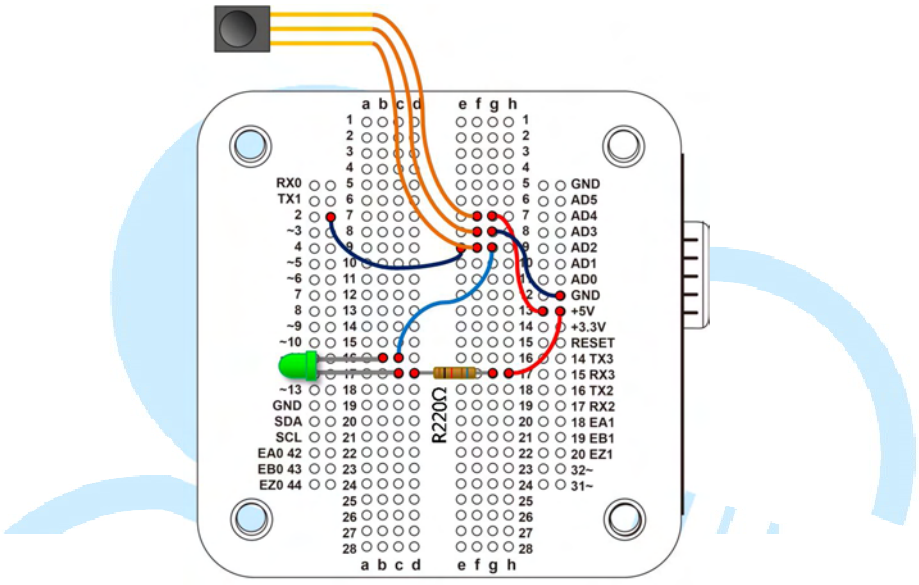
To use the `TimerOne` library, the 「`#include "TimerOne.h"`」 statement is needed. In the setup() function, the 「`Timer1.initialize(period);`」 function is called to initialize the Timer object, with parameter (period) in microsecond that represent timer cycle.

In the 「`void SendIR( )`」 function, the 「`Timer1.pwm( IR_pin, 512, 26 );`」 function is called to configure IR pin's PWM signal, where the 2 values 512 and 26 configure the IR pin's PWM signal 26 µs at 50% duty cycle, with 20ms PWM duration. The 「`Timer1.disablePwm( IR_pin );`」 function is called to disable IR pin's PWM output. As the `SendIR()` function is called in `loop()` function with 1000 ms (1 second) delay in between.

Similar to the code in the previous section, the above code switch on Infrared transmitter and then off for 1 second continuously causing the LED on the receiver circuit to blink.

3. Second exercise: Understanding Infrared Communication

In the previous section, we provided brief introduction about infrared and a simple exercise to test infrared receiver function. For this second exercise, we are going to talk about infrared communication format and protocols, using the following circuit:



The above circuit is based on the same circuit from the previous exercise, with a new connection added to connect the infrared receiver's signal output to EduCake's digital pin #2, to capture data from the infrared receiver.

From the 86Duino Coding IDE, enter the following codes for the exercise:

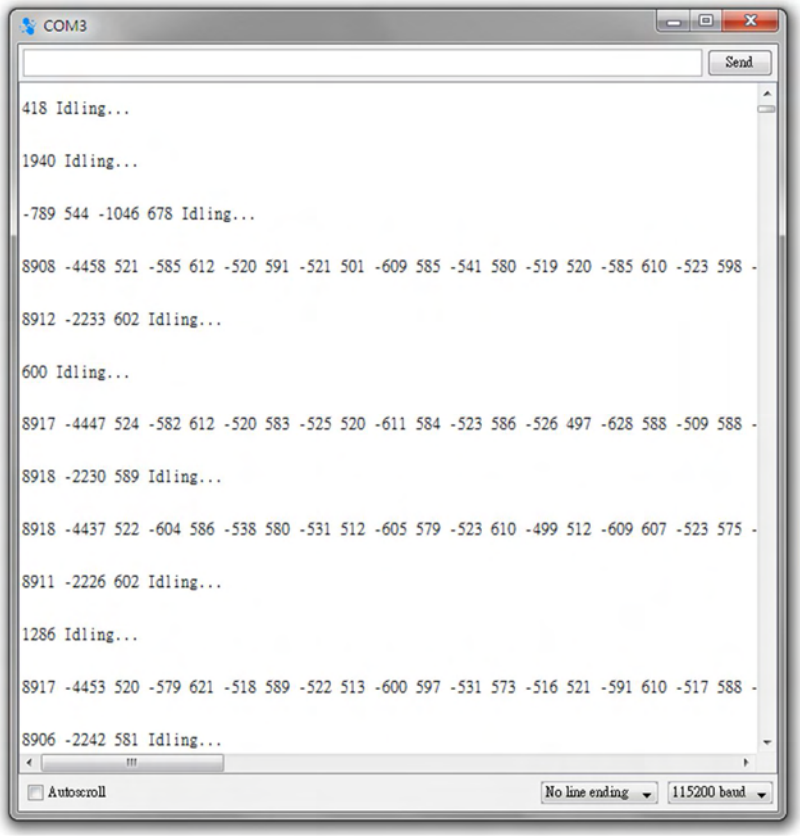
```
int IR_rec_pin = 2;// IR receiver signal output
int IRstate = LOW;// IR receiver output pin bit state
int IRstate_last = LOW;// Last IR receiver output pin bit state
long int time_last = 0;// Last IR state changed time

boolean isIdle = true;// Idling, waiting for IR signal
const long int durationMax = 10000;// inactive time to idle in µs
const long int durationMin = 400;// min. inactive duration in µs
```

```
void IR_rec_Check( )
{
    IRstate = digitalRead( IR_rec_pin );// retrieve pin status

    if( IRstate != IRstate_last ){// check for status changed

      long int timeNow = micros( );// retrieve current time
      long int dT = timeNow - time_last;// time from last event

      if( dT >= durationMax && !isIdle ){
        isIdle = true;
        Serial.println( "Idling...\n" );
      }
      else if( dT < durationMax && dT > 400 ){
        isIdle = false;
        Serial.print( IRstate == HIGH? dT : dT ); Serial.print( " " );
      }

      time_last = timeNow;
    }
    IRstate_last = IRstate;
}

void setup( ) {
  Serial.begin( 115200 );
  pinMode( IR_rec_pin, INPUT );// Configure pin operating mode
  IRstate = digitalRead( IR_rec_pin );// Read initial Pin status
  IRstate_last = IRstate;
}

void loop( ) {

  IR_rec_Check( );
  delayMicroseconds( 20 );
}
```

After compile and uploading the above code to EduCake, launch Serial Monitor from the IDE. Similar to the first exercise in this application note to test infrared receiver's function, using one of the remote control you have for your TV or home entertainment system, point the remote control toward the infrared receiver circuit and randomly press some buttons to transmit IR signal. When the remote control transmit IR signal, the signal is modulated around the 38 KHz carrier frequency.

As the IR signal from the remote control is detected by the infrared receiver, you can see series of output from the Serial Monitor, where positive value represent the signal is received when the output signal pin is HIGH and negative value

represent the signal is received when the output signal pin is LOW, as shown in the following figure:



Note: When the infrared receiver detected IR signal from the transmitter, the receiver's output pin goes LOW, when IR signal is not detected, the receiver's output pin goes HIGH. The negative values shown in the Serial Monitor represent data received from the transmitter. Since positive value represent detected signal when the receiver's output pin is HIGH where data are not detected, these value can be ignored.

The above code executes the `IR_rec_Check()` routine every 20 µs, which call the `digitalRead()` function to read infrared receiver's output pin status. When change in status is detected, the `micros()` function is called to record the time when the change took place. Then the elapsed time from the last event is calculated based on the current time retrieved by the micros() function, assigned to the `timeNow` variable, and `time_Last` variable which contains the time when the last event took place. If the elapsed time is longer than `durationMax` the code set the `isIdle` variable to true. If the elapsed time is

less than `durationMax` and more than `durationMin`, the code set the `isIdle` variable to false. If the elapsed time is less than `durationMin`, the received signal is treated as noise and ignored.

Since infrared data transmission can be a few hundred µs long, it's best to use data sampling interval in the multiple of 10 µs range. If the sampling rate is too slow, the received data may not be accurate.

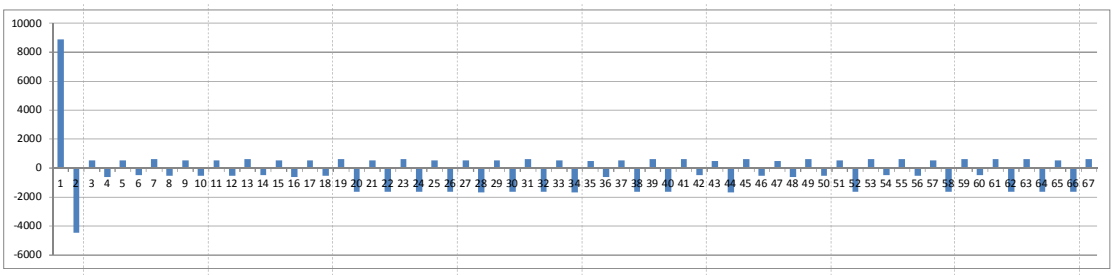
Note: Even when the IR remote is not transmitting, the receiver may be affected by interference signal and output signal with random duration. To minimize interference and insure the received data is accurate, it's necessary to implement some form of communication protocol which include special characters to indicate the beginning, ending and length of each data message, similar to the serial communication protocol discussed in the earlier chapter.

For the exercises in this application note, we use an IR remote control for an in-vehicle MP3 player, as shown in the following figure.

Using the above IR remote control as example, when pressing button "0" on the remote, the following stream of data is transmitted:

「8883 -4487 524 -599 591 -525 594 -516 522 -610 580 -527 596 -513 514 -600 595 -524 593 -1632 596 -1630 516 -1697 606 -1631 602 -1625 519 -1707 609 -1630 585 -1630 525 -595 630 -1594 523 -1704 610 -509 509 -1717 606 -510 517 -609 615 -503 582 -1630 594 -534 591 -506 520 -1707 607 -509 514 -1715 602 -814 1405 -1632 518」

Time duration for the above data is measured in µs. By placing the above data into an excel document, we can use excel's graph function to observe the data, as shown in the following figure.



After trimming away the starting and ending characters, there are 64 unit of data. When a different button is pressed, we can see different sequences of data.

From the above graph, we can see data duration from the remote is 600 µs when the signal is HIGH, and 600 µs when the signal is LOW. The combined duration is 1200 µs to from a data bit. Since the chance for transient signals to match the exact frequency pattern in use by both the transmitter and receiver is very small, interference problem is minimal. You can use IR remote control for different device to review different data pattern.

4. Third exercise: IRremote library (Receiving)

There are many different variety of IR communication protocols, which requires enormous effort trying to learn and figure out how to work with them. Luckily, there are open source libraries available to help. For the exercise in this section, we will use the IRremote for 86Duino library. This library is ported from the IRremote library created by Ken Shirriff, who generously shared this library to the open source community.

The IRremote library includes communication protocols that support IR transmitting and receiving functions for devices from different manufacturers, including NEC, Sony SIRC, Philips RC5, Philips RC6, Sharp, Panasonic, JVC, Sanyo, Mitsubishi and etc. The IRremote library includes function to output received data in raw data format, enabling you to observe all of the received data.

For this section, using the same circuitry from previous exercise, we will work through an example to receive IR signals and talk about some practical use case for IR, using the same circuitry from previous exercise.

From the 86Duino IDE, enter the following codes:

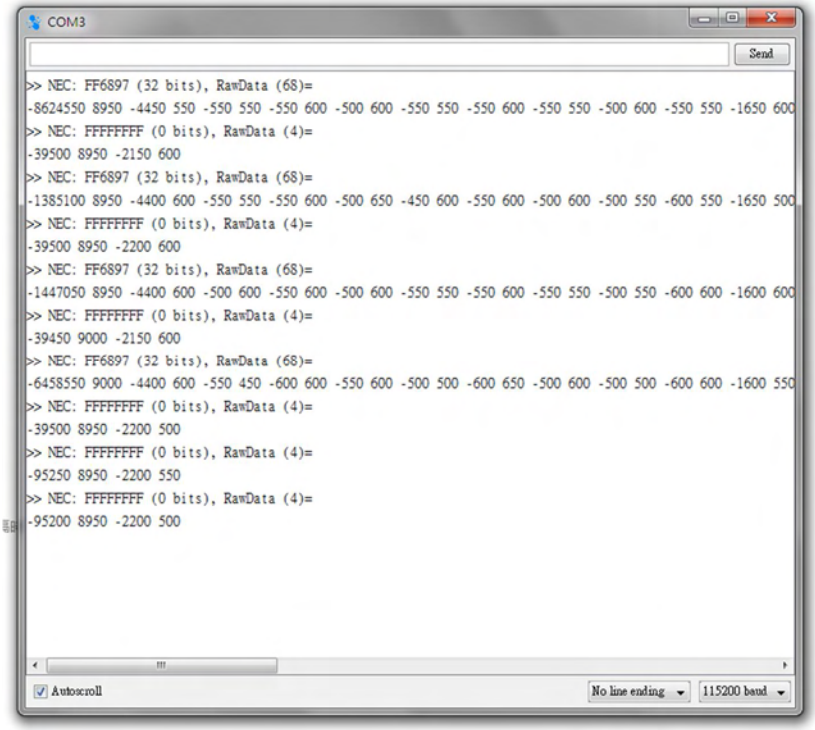
```
#include <IRremote.h>
int IR_rec_pin = 2;// I/O pin attach to IR receiver output

// IRremote library - IR receiver assignment
IRrecv IRrecver( IR_rec_pin );

// Variable to hold decoded result
decode_results results;

// Output decoded data for observation
void Print_IRdecodeResult( decode_results &decodeResults )
{
  int dataLength = decodeResults.rawlen;

  switch( decodeResults.decode_type )
  {
    case NEC:
      Serial.print( ">> NEC:\t" );
      break;

    case SONY:
      Serial.print( ">> SONY:\t");
      break;

    case RC5:
      Serial.print( ">> RC5:\t");
      break;
```

```
    case RC6:
      Serial.print( ">> RC6:\t");
      break;

    case DISH:
      Serial.print( ">> DISH:\t");
      break;

    case SHARP:
      Serial.print( ">> SHARP:\t");
      break;

    case SANYO:
      Serial.print( ">> SANYO:\t" );
      break;

    case MITSUBISHI:
      Serial.print( ">> MITSUBISHI:\t" );
      break;

    case PANASONIC:
      Serial.print( ">> PANASONIC(addr=\t" );

  Serial.print( results.panasonicAddress );
      Serial.print( "):\t" );
      break;

    case JVC:
      Serial.print( ">> JVC:\t" );
      break;

    case UNKNOWN:
      Serial.print( ">> Unknown:\t" );
      break;

    default:
      break;
  }

  // Decoded data from protocols (16-Bit)
  Serial.print( decodeResults.value, HEX );
  Serial.print( " (" );

  // Total number of received data bit.
  Serial.print( decodeResults.bits, DEC );
  Serial.print( " bits), " );

  Serial.print( "RawData (" );
  Serial.print( dataLength, DEC );
  Serial.println( ")= " );

  // Raw received data output
  for ( int i = 0; i < dataLength; i++) {
    int data = decodeResults.rawbuf[i] * USECPERTICK;
    if ( (i % 2) == 1 ) {
      Serial.print( data, DEC );// HIGH
    }
```

```
    else {
        Serial.print( -data, DEC );// LOW
      }
      Serial.print( " " );
    }
    Serial.println( );
  }


  void setup( )
  {
    Serial.begin( 115200 );
    IRrecver.enableIRIn( );// Initialize IR Receiver
  }

  void loop() {
    if ( IRrecver.decode( &results ) )
    {
      Print_IRdecodeResult( results );
      // Resume IR receiving after completing current data decoding
      IRrecver.resume( );
    }
  }
```

After compile and uploading the above code to EduCake, launch Serial Monitor from the 86Duino IDE. Then, point the IR remote control to the receiver circuit and press a few buttons to transmit IR data.

The Serial Monitor on the following figure is showing received data from the MP3 IR remote control used for the exercise.



When pressing 0 on the IR remote to transmit data, the received data from the IR receiver circuit is decoded by the IRremote library. With existing function that are part of the IRremote library, you can easily create 86Duino sketch capable of decoding IR data transmission using different protocols from different manufacture.

The MP3 IR remote control used, as part of the process to create the sample exercise, is using NEC's IR communication protocol. Data length and duration for the data stream in this exercise is similar to exercise 2 in the earlier section. Following is the graphical presentation for the data stream using Excel.

Following are some of the IR transmission behavior for MP3 IR remote control we use, which is based on NEC's transmission protocol:

- Leading signal: When a key is pressed on the remote control, a 9ms leading pulse burst with 4.5ms space.
- Data logic '0': A 560 µs pulse burst follow by a 560 µs space
- Data log '1': A 560 µs pulse burst follow by a 1690 µs space
- Repeat (If the key on the remote control is kept pressed, a repeat code will be issued): A 9 ms leading pulse burst with 2.25 ms space, follow by a 560 µs pulse burst to mark the end of the space.

When pressing the '0' button on the MP3 IR remote, it's corresponding to the 0x00FF6897 coding with 64 HIGH/LOW that make up 32 logical bit, total of 4 bytes, where the 1st 16-Bit is the address and the last 16-Bit is the encoded data.

When one of the button is kept pressed, after receiving the initial set of data, the repeat command ('FFFFFFFF') is sent every 110 ms. Let's say you press the volume up button on the remote, it transmit a data stream that represent the volume up button follow by the repeat code, 'FFFFFFFF', until the button is

released. Since the leading data stream prior to the repeat code is different, multiple IR remote kept pressed in the same room would not interfere the others' function, which is one of the key advantage of adopting and use communication protocol.

To use the IRremote library, you need to add "IRremote.h", follow by "IRrecv IRrecver(IR_rec_pin)", the syntax to declare an IR receiver object, where the I/O pin attached to the IR receiver is the function parameter. The "decode_results results" is a class with the following:

- decode_type: Designate encoding method
- panasonicAddress: Address field for Panasonic specific protocol
- Value: Data value
- bits: Total number of data bit.
- unsigned int*rawbuf: Raw data buffer
- rawlen: Raw data length

When pressing the '0' button on the MP3 IR remote, it's corresponding to the 0x00FF6897 coding with 64 HIGH/LOW that make up 32 logical bit, total of 4 bytes, where the 1st 16-Bit is the address and the last 16-Bit is the encoded data.

During the setup() phase, the IRrecver.enableIRIn() function is called to the initialize IR receiver. Then, in the main loop() function, the IRrecver.decode(&results) function is called to scan data from the IR receiver, identify encoding type and assign decoded data to the results variable upon successful decoding and return true. When failing to decode data, the function return false. When the function return true, the Print_IRdecodeResult() function is called to print out the decoded data.

Comparing with the 2nd exercise earlier, this exercise enables you to view and observe the protocol and transmitted data from different IR remote control you have.

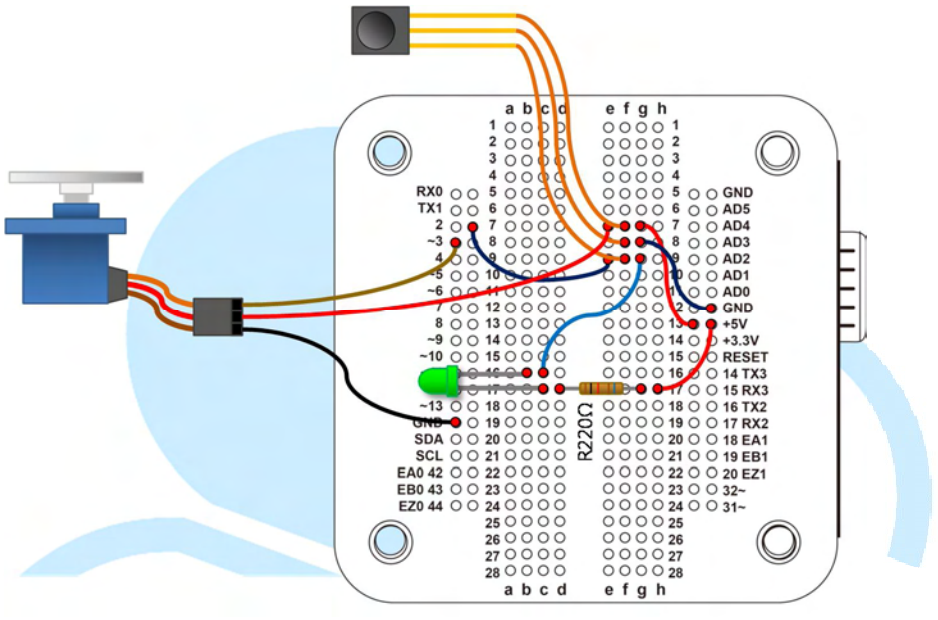
For more information about NEC's IR communication protocol, visit the following URLs:

http://mcudiy.blogspot.tw/2010/11/22-irinfrared-nec-protocol.html

http://www.sbprojects.com/knowledge/ir/nec.php

5. Third exercise (Part-2): IRremote library (Receiving)

Continue with the exercise from the previous section, Additional components are added to extend functionality to provide servo control, as shown in the following figure:



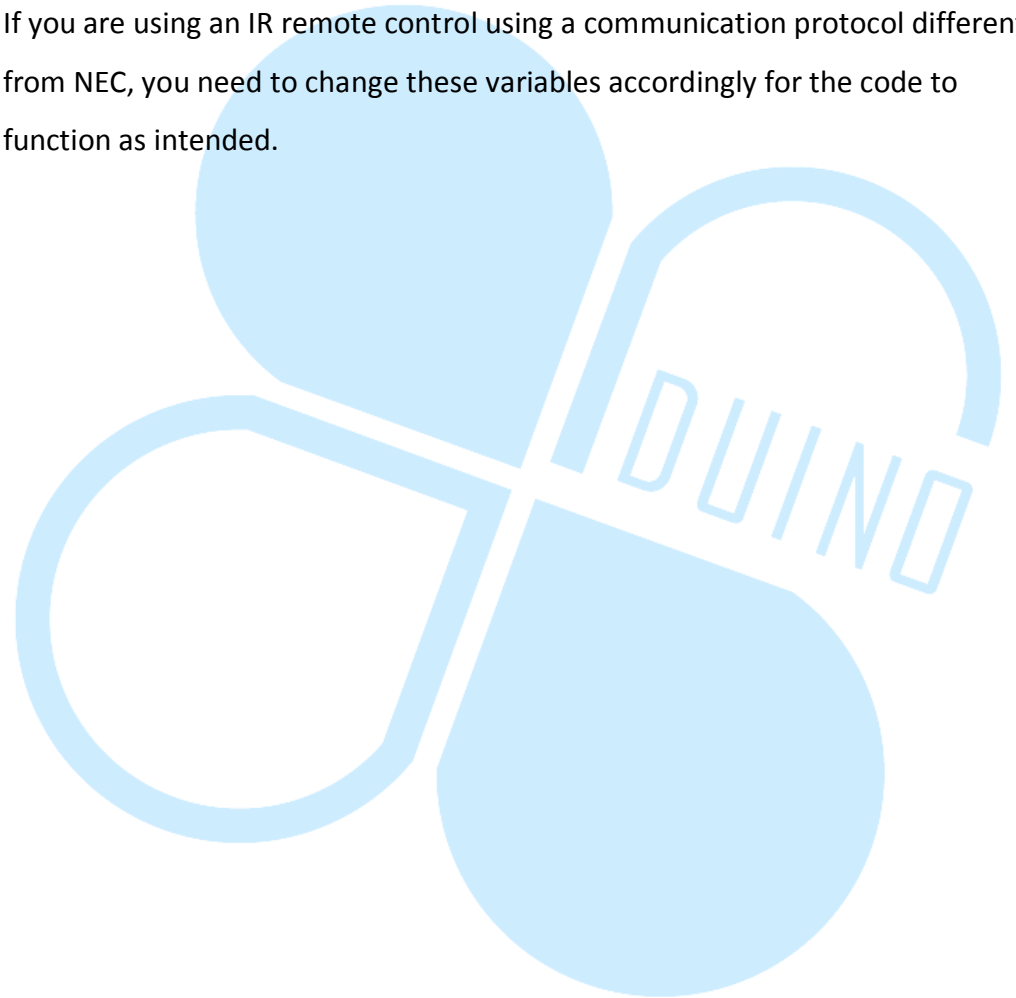
After completing the above circuitry, enter the following code to the 86Duino Coding IDE:

```
#include <IRremote.h>
#include <Servo.h>

int IR_rec_pin = 2;// IR receiver data output pin
int servo_pin = 3;// Servo output pin

IRrecv IRrecver(IR_rec_pin);// IR receiver object
decode_results results;// object to store decoded data

Servo servo_0;// Servo object
typedef enum
{
  DIR_NONE = 0,
  DIR_LEFT,
  DIR_RIGHT
} ServoDir;// Define rotation direction

int servoDir = DIR_NONE;// Servo rotation direction
unsigned int ServoPosition = 1500;// Serco rotation position

// Output decoded data
void Print_IRdecodeResult( decode_results &decodeResults )
{
```

```
    if( decodeResults.decode_type == NEC )
    {
      switch( decodeResults.value )
      {
        case 0x00FFA25D:// CH- buttom
          servoDir = DIR_LEFT;
          ServoPosition += 50;
          break;

        case 0x00FF629D:// CH buttom
          servoDir = DIR_NONE;
          ServoPosition = 1500;
          servo_0.writeMicroseconds(ServoPosition);
          break;

        case 0x00FFE21D:// CH+ buttom
          servoDir = DIR_RIGHT;
          ServoPosition -= 50;
          break;

        case 0xFFFFFFFF:// Repeat
          if( servoDir == DIR_RIGHT )
          { ServoPosition -= 50; }
          else if( servoDir == DIR_LEFT )
          { ServoPosition += 50; }

  // Limit servo position within a safe range
  ServoPosition = constrain( ServoPosition, 1100, 1900 );

          // Control rotation angle
          servo_0.writeMicroseconds( ServoPosition );

          Serial.print( "ServoPosition = " );
          Serial.println( ServoPosition );
          break;

        default:
          break;
      }
    }
  }
  void setup( )
  {
    Serial.begin( 115200 );
    IRrecver.enableIRIn( );// Initial IR receiver object
    servo_0.attach( servo_pin );// Set I/O pin attached to Servo
  }

  void loop( ) {
    if ( IRrecver.decode( &results ) )
    {
      Print_IRdecodeResult( results );

      // After retrieve and decode data from IR receiver,
      // this function is call to resume IR receiver function.
      IRrecver.resume( );
    }
  }
```

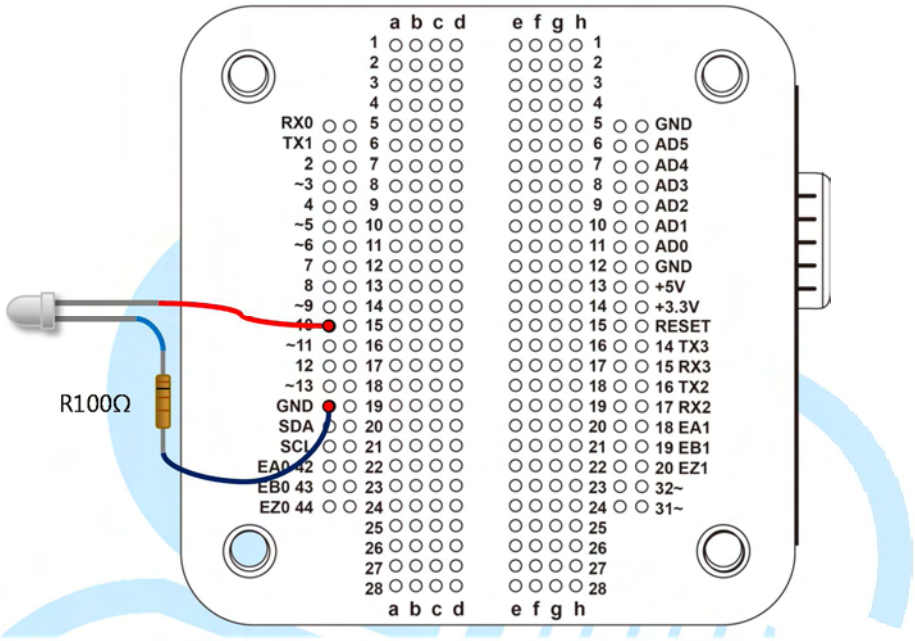
After compile and uploading the above code to EduCake, you can control the RC servo's movement, moving the servo's arm, using an IR remote control. The above code is similar to the 1st exercise, with some modification in the Print_IRdecodeResult() function. In the switch-case block of code, the case selection value "0x00FFA25D", "0x00FF629D" and "0x00FFE21D" are used to correspond to the CH-, CH and CH+ button on the IR remote control we use to create this exercise, which is based on NEC protocol.

If you are using an IR remote control using a communication protocol different from NEC, you need to change these variables accordingly for the code to function as intended.



6. Third exercise (Part-3): IRremote library (Transmit)

In the previous exercises, we talked about IR receiving function. In this exercise, we will talk about IR transmit function, using the following circuitry.

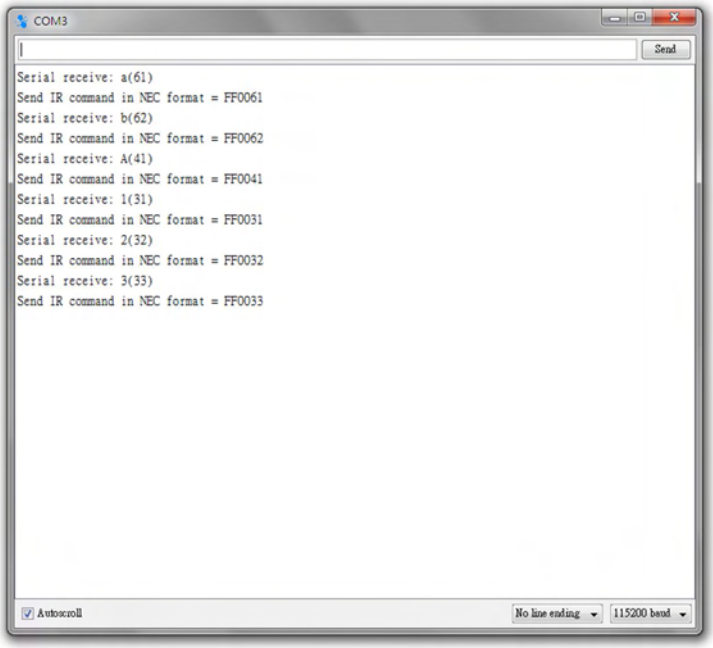


From the 86Duino Coding IDE, enter the following Code:

```
#include <IRremote.h>

int ID_send_pin = 10;// Define I/O pin attached to IR transmitter

IRsend IR_send;// IR transmitter object for IRremote library

void setup( )
{
  Serial.begin( 115200 );

  IR_send.outPin( ID_send_pin );
  // Note:  pin 10 on EduCake with PWM output capability is
  // used to send signal to the IR transmitter
}
```

```
void loop( ) {
if ( Serial.available( ) ) {
    char data = Serial.read( );
    //unsigned long cmd = 0x00FF1234;
    // Addr = 00FF, Data = 1234
    unsigned long cmd = 0x0;// Addr = 00FF, Data = ?
    unsigned long DeviceAddr = 0x00FF;

    // Use data received from Serial port as IR command
    cmd = ( DeviceAddr<<16 ) | (unsigned long)data;
    IR_send.sendNEC( cmd, 32 );// sned NEC code format (command, data
bits)

    Serial.print( "Serial receive: " );
    Serial.print( data );// char
    Serial.print( "(" );
    Serial.print( data, HEX );
    Serial.println( ")" );

    Serial.print( "Send IR command in NEC format = " );
    Serial.println( cmd, HEX );
  }
  delay( 100 );
}
```

After compiling and uploading the above code to EduCake, launch Serial Monitor from the 86Duino IDE and enter some character to the Serial Monitor. The above code encode ASCII characters received from the serial port and transmit via the IR transmitter and show the activities on the Serial Monitor, as shown in the following figure.



The above exercise begin with the "#include<IRremote.h>" statement to bring in the IRremote library, follow by the "IRsend IR_send;" statement to define an IR transmitter object. Then, in the "setup()" function, the "IR_send.outPin(ID_send_pin)" function is called to set the I/O pin attached to the IR transmitter (Note: You must use one of the I/O with the "~" mark, which represent the I/O pin is capable to output PWM signal, needed to support IR transmission.). For the exercise here, we use pin 10 on the EduCake. (The "#define TIMER_PWM_PIN 10" statement is part of the "IRremote.h" file, which is included as part of the 86Duino Coding IDE, under the "\hardware\86Duino\x86\libraries\IRremote\" folder.)

In the "loop()" function, the "Serial.available()" function is call to detect data sent from the Serial Monitor. When data from the Serial Monitor is detected, the "char data = Serial.read();" statement is called to read and store the received-data to the data variable. The MP3 IR remote control we are using for this exercise transmit 32 bit data. The "cmd = (DevicdAddr<<16)|(unsigned long)data;" statement place the address location for the data on the left 16-bit of the data packet, and the actual data value on the right 16-bit of the data packet. You can modify the code in this section to transmit different IR data value to trigger different control to better understand how the code function.

After composing the desired command, use the library's IR_send object to send the data using NEC protocol, "IR_send.sendNEC(unsign long command, int bits)". Since we are working with 32 bit data in this exercise, the "IR_send.sendNEC()" function is called with 32 as data length. If you changed the code to work with different data length, be sure to change the data length value to match.

To support IR transmitter/receiver from different manufacturers, in addition to the sendNEC function, the IRremote library also includes function to support others, which include sendSony, sendRC5, sendRC6, sendDISH, sendSharp, sendPanasonic, sendJVC and etc.

If you like to establish your own IR communication protocol and module the data at a different carrier frequency, you can use the following function provided as part of the IRremote library:

- sendRaw(unsigned int buf[], int len, int Khz)

Following is an example for the above “sendRaw()” function:

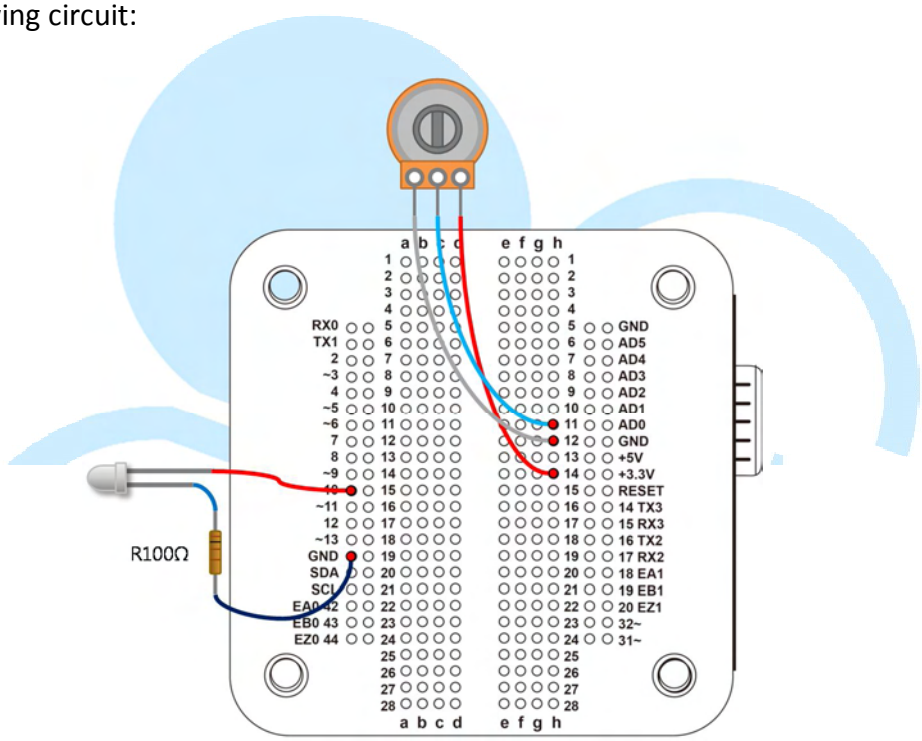
```
unsigned int cmdBuf[ ] = {
     8900, 4450,// H L
     550, 600, 550, 500,// H L H L
     600, 550, 550, 550,
     550, 550, 600, 500,
     550, 600, 550, 550,
     600, 1650, 550, 1650,
     600, 1650, 550, 1650,
     550, 1700, 550, 1650,
     600, 1650, 550, 1700,
     500, 600, 550, 1650,
     600, 1650, 600, 500,
     500, 1700, 600, 550,
     500, 600, 600, 550,
     550, 1650, 600, 500,
     600, 550, 550, 1650,
     600, 500, 600, 1650,
     600, 1650, 550, 1650,
     600
    };// 67
    int cmdLength = 67;
    IR_send.sendRaw( cmdBuf, cmdLength, 38 );
```

In the above code, the “cmdBuf[]” array is used to store a series of HIGH/LOW data sequences, starting with signal HIGH. The “sendRaw(unsigned int buf[], int len, int Khz)” function is called with parameters that include data array, with data value in µs, data length and modulation frequency in KHz. Using this function, you can transmit data to match just about any IR transmission protocol.

7. Fourth exercise: IR communication between 2 EduCake

In the previous section, we covered the basic usage for the IRremote library. In this exercise, we will increase the complexity a little bit, using 2 86Duino EduCake devices and establish IR communication between the 2 devices.

One of the EduCake function as receiver, using the same circuitry as the one in Exercise 3 part-2. The other EduCake function as transmitter, using the following circuit:



From the 86Duino Coding IDE, enter the following codes for the transmitting device:

```
#include <IRremote.h>

int ID_send_pin = 10;// Pin attached to IR tranmitter
int VR_pin = A0;

IRsend IR_send;// IRsend object for the IRremote library

void setup( )
{
  Serial.begin( 115200 );

  IR_send.outPin( ID_send_pin );
  // Note:  pin 10 on EduCake with PWM output capability is
  // used to send signal to the IR transmitter

}
void loop( ) {

  // Read resistor value from the variable resistor
  // (range = 0 ~ 1023)
  unsigned int VRvalue = analogRead( VR_pin );

  // Both transmitting and receiving devices
  // must use the same address
  unsigned long DeviceAddr = 0x00AA;

  // prepare data to be send
  unsigned long cmd = ( DeviceAddr<<16 ) | ( unsigned long )VRvalue;

  // Transmit data using NEC protocol
  IR_send.sendNEC( cmd, 32 );

  Serial.print( "VR value: " );
  Serial.println( VRvalue, DEC );
  Serial.print( "Send IR command in NEC format = " );
  Serial.println( cmd, HEX );

  delay( 200 );
}
```

For the EduCake device function as the receiver, use the following code:

```
#include <IRremote.h>
#include <Servo.h>

int IR_rec_pin = 2;// I/O pin attached to IR receiver
int servo_pin = 3;// I/O pin attached to Servo

IRrecv IRrecver( IR_rec_pin );// IR receiver object

decode_results results;// variable to store decoded result

Servo servo_0;// Servo object
```

```
// Output successfully decoded data for observation
void Print_IRdecodeResult( decode_results &decodeResults )
{
  // Device address from received data stream
  unsigned int DeviceAddr = ( unsigned int )( ( decodeResults.value
& 0xFFFF0000 )>>16 );

  // data value from received data stream
  unsigned int VRvalue = ( unsigned int )( decodeResults.value &
0x0000FFFF );

  if( decodeResults.decode_type == NEC && DeviceAddr == 0x00AA )
  {
    // Value range 0~1023 mapped to 1000~2000
    int ServoPosition = map( VRvalue, 0, 1023, 1000, 2000 );

    // Limit servo movement within safe range
    ServoPosition = constrain( ServoPosition, 1100, 1900 );

    // set servo position
    servo_0.writeMicroseconds( ServoPosition );

    Serial.print( "IR receive OK, raw data = " );
    Serial.print( VRvalue, DEC );
    Serial.print( "ServoPosition = " );
    Serial.println( ServoPosition, DEC );
  }
}

void setup( )
{
  Serial.begin( 115200 );
  IRrecver.enableIRIn( );// Initialize receiver object
  servo_0.attach( servo_pin );// configure I/O attached to servo
}

void loop( ) {
  if ( IRrecver.decode( &results ) )
  {
    Print_IRdecodeResult( results );

    // Resume receiving data after decoding current received data
    IRrecver.resume( );
  }
}
```

After the codes are compiled and uploaded to both transmitting and receiving EduCake devices, you can control movement of the servo attached to the EduCake device that function as the IR receiver by changing the variable resistor on the EduCake that function as the IR transmitter.

On the IR transmitting EduCake, there are codes that acquire data from the analog to digital converter periodically, combines the acquired data with device address 0x00AA and transmit the combined data.

The code for the IR receiver is similar to exercise 3 part-2 earlier, with the following functions to decode device address and data value from the received IR data stream:

```
unsigned int DeviceAddr = (unsigned int)((decodeResults.value &
0xFFFF0000 )>>16 )
unsigned int VRvalue = (unsigned int)(decodeResults.value &
0x0000FFFF)
```

Then, after the data is successfully decoded, based on NEC IR communication protocol, VRvalue (value from the transmitting device's variable resistor) is used to set the servo's position, within a min and max limiting value to keep servo movement within a safe range.

Once you understand IR communication introduced in this application note, using an 86Duino EduCake, you can simulate large variety of IR remote control. Even for the remote control without documentation, it's possible for you to record IR data stream from the remote, analyze the recorded data and figure out the protocol and commands.