Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

IR Kit (SKU:DFR0107)

Contents

Introduction

IR is widely used in remoter control. With this IR receiver, the Arduino project is able to receive command from any IR remoter controller if you have the right decoder. Well, it will be also easy to make your own IR controller using IR transmitter.

Sample Code

```
// 0.1 by pmalmsten http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1176098434

// 0.2 by farkinga

// 0.3 by farkinga - adds cool behaviors

/* 0.4 by pstrobl

   changed the original program to use on IR Kit For Arduino Model: DFR0107 32 bit controller.

   All keypresses are sent to the serial monitor at 9600 baud.

   pulseIn is always HIGH. The phototransistor in the kit does not invert the signal.

   uses pin 13 for heartbeat debug
```

```
   32 bits requires a long variable, so divided up into two 15 bit so can use
integer variables

   use the first 15 bits of the 32 bits for remote and data stream verificati
on. This code is always the same for every button press

   use the last 15 of the 32 bits for button press selection. This code chang
es for each button.

   ignore the middle 2 bits, it never changes.

*/


#define IR_BIT_LENGTH 32    // number of bits sent by IR remote

#define FirstLastBit 15     // divide 32 bits into two 15 bit chunks for inte
ger variables. Ignore center two bits. they are all the same.

#define BIT_1 1500          // Binary 1 threshold (Microseconds)

#define BIT_0 450           // Binary 0 threshold (Microseconds)

#define BIT_START 4000      // Start bit threshold (Microseconds)


#define IR_PIN 2            // IR Sensor pin

#define LED_PIN 13          // LED goes off when signal is received


int debug = 0;              // flag as 1 to output raw IR pulse data stream l
ength in microseconds

int output_verify = 0;      // flag as 1 to print decoded verification intege
rs. same number for all buttons

int output_key = 0;         // flag as 1 to print decoded key integers

int remote_verify = 16128;  // verifies first bits are 11111100000000 differe
nt remotes may have different start codes


void setup() {

  pinMode(LED_PIN, OUTPUT);    //This shows when ready to recieve

  pinMode(IR_PIN, INPUT);

  digitalWrite(LED_PIN, LOW);

  Serial.begin(9600);

}


void loop() {

  digitalWrite(LED_PIN, HIGH);
```

```
  int key = get_ir_key();

  digitalWrite(LED_PIN, LOW);  // turn LED off while processing response
  do_response(key);
  delay(130);                  // 2 cycle delay to cancel duplicate keypresse
s
}

/*
  wait for a keypress from the IR remote, and return the
  integer mapping of that key (e.g. power button on remote returns
  the integer 1429)
*/

int get_ir_key()
{
  int pulse[IR_BIT_LENGTH];
  int bits[IR_BIT_LENGTH];

  do {} //Wait for a start bit
  while(pulseIn(IR_PIN, HIGH) < BIT_START);

  read_pulse(pulse);
  pulse_to_bits(pulse, bits);
  RemoteVerify(bits);
  return bits_to_int(bits);
}



/*
  use pulseIn to receive IR pulses from the remote.
  Record the length of these pulses (in ms) in an array
*/
```

```
void read_pulse(int pulse[])
{
  for (int i = 0; i < IR_BIT_LENGTH; i++)
  {
    pulse[i] = pulseIn(IR_PIN, HIGH);
  }
}

/*
  IR pulses encode binary "0" as a short pulse, and binary "1"
  as a long pulse.  Given an array containing pulse lengths,
  convert this to an array containing binary values
*/

void pulse_to_bits(int pulse[], int bits[])
{
  if (debug) { Serial.println("-----"); }
  for(int i = 0; i < IR_BIT_LENGTH; i++)
  {
    if (debug) { Serial.println(pulse[i]); }
    if(pulse[i] > BIT_1) //is it a 1?
    {
      bits[i] = 1;
    }
    else if(pulse[i] > BIT_0) //is it a 0?
    {
      bits[i] = 0;
    }
    else //data is invalid...
    {
      Serial.println("Error");
    }
  }
}
```

```
/*
  check returns proper first 14 check bits
*/

void RemoteVerify(int bits[])
{
  int result = 0;
  int seed = 1;

  //Convert bits to integer
  for(int i = 0 ; i < (FirstLastBit) ; i++)
  {
    if(bits[i] == 1)
    {
        result += seed;
    }

    seed *= 2;
  }
        if (output_verify)
      {
        Serial.print("Remote ");
        Serial.print(result);
        Serial.println(" verification code");
      }
 if (remote_verify != result) {delay (60); get_ir_key();} //verify first grou
p of bits. delay for data stream to end, then try again.
}



/*
  convert an array of binary values to a single base-10 integer
*/
```

```
int bits_to_int(int bits[])
{
  int result = 0;
  int seed = 1;

  //Convert bits to integer
  for(int i = (IR_BIT_LENGTH-FirstLastBit) ; i < IR_BIT_LENGTH ; i++)
  {
    if(bits[i] == 1)
    {
        result += seed;
    }
    seed *= 2;
  }
  return result;
}



/*
  respond to specific remote-control keys with different behaviors
*/

void do_response(int key)
{

  if (output_key)
   {
      Serial.print("Key ");
      Serial.println(key);
   }

  switch (key)
  {
```

```
    case 32640:  // turns on UUT power
      Serial.println("POWER");
      break;

    case 32385:  // FUNC/STOP turns off UUT power
      Serial.println("FUNC/STOP");
      break;

    case 32130:  // |<< ReTest failed Test
      Serial.println("|<<");
      break;

    case 32002:  // >|| Test
      Serial.println(">||");
      break;

    case 31875:  // >>| perform selected test number
      Serial.println(">>|");
      break;

    case 32512:  // VOL+ turns on individual test beeper
      Serial.println("VOL+");
      break;

    case 31492:  // VOL- turns off individual test beeper
      Serial.println("VOL-");
      break;

    case 31620:  // v scroll down tests
      Serial.println("v");
      break;

    case 31365:  // ^ scroll up tests
      Serial.println("^");
```

```
  break;

case 30982:  // EQ negative tests internal setup
  Serial.println("EQ");
  break;

case 30855:  // ST/REPT Positive tests Select Test and Repeat Test
Serial.println("ST/REPT");
  break;

case 31110:  // 0
  Serial.println("0");
  break;

case 30600:  // 1
  Serial.println("1");
  break;

case 30472:  // 2
  Serial.println("2");
  break;

case 30345:  // 3
  Serial.println("3");
  break;

case 30090:  // 4
  Serial.println("4");
  break;

case 29962:  // 5
  Serial.println("5");
  break;
```

```
    case 29835:  // 6
      Serial.println("6");
      break;

    case 29580:  // 7
      Serial.println("7");
      break;

    case 29452:  // 8
      Serial.println("8");
      break;

    case 29325:  // 9
      Serial.println("9");
      break;

    default:
      {
        Serial.print("Key ");
        Serial.print(key);
        Serial.println(" not programmed");
      }
    break;
  }
}
```

Trouble shooting

More question and cool idea, visit DFRobot Forum