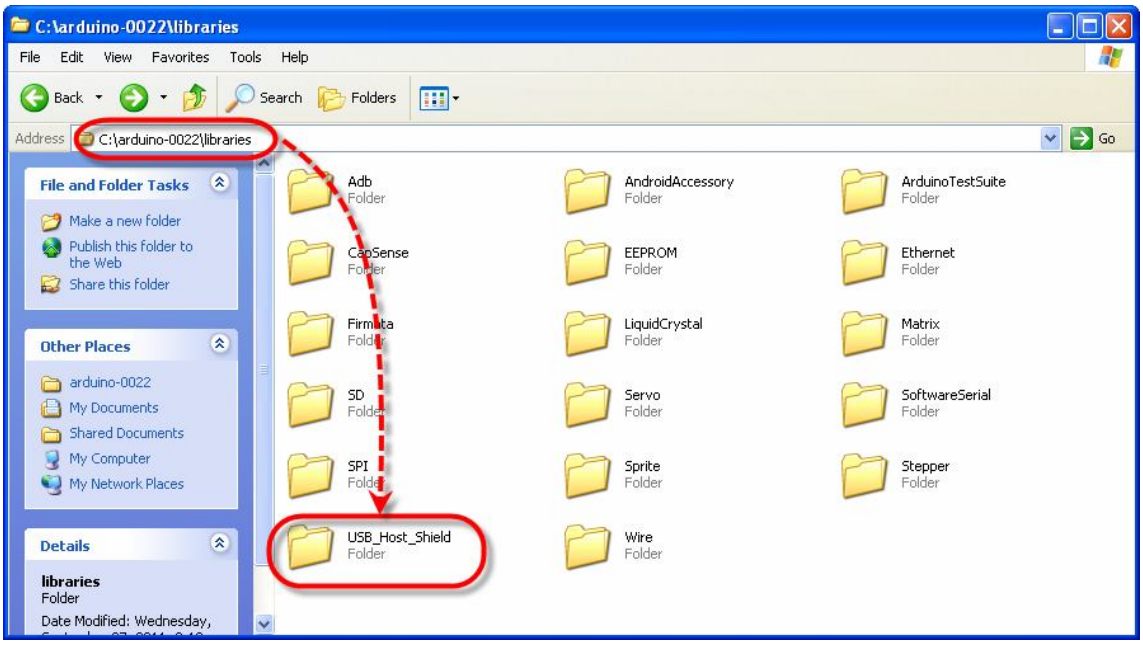
ตัวอย่างการเชื่อมต่อ USB Host กับ USB HID Keyboard

สำหรับตัวอย่างนี้จะนำเสนอ ตัวอย่างการนำบอร์ด ET-MEGA2560-ADK ไปประยุกต์ใช้งานเพื่อเชื่อมต่อกับ USB HID Keyboard ทางพอร์ต USB Host ซึ่ง ตัวอย่างนี้ ดัดแปลงมาจากโครงงานซึ่งได้รับการคิดค้นและเผยแพร่ไว้ที่ “http://www.circuitsathome.com/” ซึ่งเป็นเวปไซต์ที่นำเสนอโครงงานและตัวอย่างการประยุกต์ใช้งาน เกี่ยวกับการเชื่อมต่อ USB Host กับ อุปกรณ์ USB Device มากมายหลายรูปแบบ ซึ่งโครงงาน USB HID Keyboard ก็เป็นอีกโครงงานหนึ่งที่ถูกนำเสนอไว้ โดยรายละเอียดโครงงานอยู่ที่ “http://www.circuitsathome.com/mcu/how-to-drive-usb-keyboard-from-arduino#more-1739” โดยทางอีทีที ได้นำตัวอย่างดังกล่าวมาทำการดัดแปลงแก้ไขใหม่ เพื่อให้สอดคล้องกับระบบฮาร์ดแวร์ของบอร์ด ET-MEGA2560-ADK โดยผลการทำงานของตัวอย่างนี้ จะแสดงให้เห็นขั้นตอนและวิธีการเชื่อมต่อกับ USB HID Keyboard โดยแสดงผลการทำงานทางพอร์ตสื่อสารอนุกรม (USB HID Comport FT232R) ของบอร์ด เป็นค่ารหัส ASCII ของคีย์ที่กด

สำหรับตัวอย่างนี้ ในส่วนของฮาร์ดแวร์ จำเป็นต้องใช้แหล่งจ่ายไฟจากภายนอกจ่ายให้กับบอร์ดด้วย โดยต่อ USB HID Keyboard เข้ากับพอร์ต USB Host ของบอร์ด ET-MEGA2560-ADK และต่อสาย USB Device (USB Mini จาก FT232R) ของบอร์ดเข้ากับพอร์ต USB Host ของคอมพิวเตอร์ PC โดยผลการทำงานของโปรแกรมจะแสดงผลทางพอร์ตสื่อสารอนุกรม UART1 ของบอร์ด (USB Mini จาก FT232R) สำหรับด้าน Software นั้นต้องทำการ Copy ไฟล์ Library ของ "USB_Host_Shield" ที่ทำการ Modify แล้วไปเพิ่มไว้ในโฟลเดอร์ "..\arduino-0022\libraries" ดังตัวอย่าง



ซึ่ง Library ของ "USB_Host_Shield" จะใช้ต้นฉบับจาก http://www.circuitsathome.com มาทำการปรับแก้ส่วนของขาสัญญาณที่ใช้ในการเชื่อมต่อระหว่าง MAX3421E กับ ATMEGA2560 ให้ตรงกับระบบฮาร์ดแวร์ของบอร์ด ET-MEGA2560-ADK คือ

- MAX3421E RESET เชื่อมต่อกับ PJ2 ของ ATMEGA2560
- MAX3421E GPX เชื่อมต่อกับ PJ3 ของ ATMEGA2560
- MAX3421E SS เชื่อมต่อกับ PH7 ของ ATMEGA2560
- MAX3421E INT เชื่อมต่อกับ PE6 ของ ATMEGA2560
- MAX3421E MISO เชื่อมต่อกับ PB3(D50) ของ ATMEGA2560
- MAX3421E MOSI เชื่อมต่อกับ PB2(D51) ของ ATMEGA2560
- MAX3421E SCK เชื่อมต่อกับ PB1(D52) ของ ATMEGA2560

ตัวอย่างโปรแกรม

```
/*
 * Examples  : Arduino USB Host Examples By....ETT CO.,LTD
 * Program   : USB_HID_Keyboard
 * Hardware  : ET-MEGA2560-ADK(Arduino)
 * Function  : Demo USB HID Keyboard Interface
 */

#include <Max3421e.h>
#include <Usb.h>

/* keyboard data taken from configuration descriptor */
#define KBD_ADDR        1
#define KBD_EP          1
#define KBD_IF          0
#define EP_MAXPKTSIZE   8
#define EP_POLL         0x0a

//****************************************************************************
//  macros to identify special charaters(other than Digits and Alphabets)
//****************************************************************************
//USB HID Keyboard Scancode = Page(0x07)
//Document:Universal Serial Bus HID Usage Tables Page[53..60]
#define BANG          (0x1E)  // 1 !
#define AT            (0x1F)  // 2 @
#define POUND         (0x20)  // 3 #
#define DOLLAR        (0x21)  // 4 $
#define PERCENT       (0x22)  // 5 %
#define CAP           (0x23)  // 6 ^
#define AND           (0x24)  // 7 &
#define STAR          (0x25)  // 8 *
#define OPENBKT       (0x26)  // 9 (
#define CLOSEBKT      (0x27)  // 0 )

#define RETURN        (0x28)  // Enter
#define ESCAPE        (0x29)  // Esc
#define BACKSPACE     (0x2A)  // Backspace
#define TAB           (0x2B)  // Tab
#define SPACE         (0x2C)  // Spacebar
#define HYPHEN        (0x2D)  // - _
#define EQUAL         (0x2E)  // = +
#define SQBKTOPEN     (0x2F)  // [ {
#define SQBKTCLOSE    (0x30)  // ] }
#define BACKSLASH     (0x31)  // \ |
#define SEMICOLON     (0x33)  // ; :
#define INVCOMMA      (0x34)  // ' "
#define TILDE         (0x35)  // ~
#define COMMA         (0x36)  // , <
#define PERIOD        (0x37)  // . >
#define FRONTSLASH    (0x38)  // / ?
#define DELETE        (0x4c)  //

/* "Sticky keys */
#define CAPSLOCK      (0x39)  // Cap Lock
#define SCROLLLOCK    (0x47)  // Scroll Lock
#define NUMLOCK       (0x53)  // Num Lock

/* Modifier masks. One for both modifiers */
#define CTRL          0x11    // Ctrl
#define SHIFT         0x22    // Shift
#define ALT           0x44    // Alt
#define GUI           0x88    // GUI

/* Sticky keys output report bitmasks */
// Output Report = 1 Byte
// ->Bit[0] = NUM LOCK
// ->Bit[1] = CAPS LOCK
// ->Bit[2] = SCROLL LOCK
// ->Bit[3] = COMPOSE
// ->Bit[4] = KANA
// ->Bit[5..7] = CONSTANT
#define bmNUMLOCK     0x01
#define bmCAPSLOCK    0x02
#define bmSCROLLLOCK  0x04
```

```
EP_RECORD ep_record[ 2 ];   //endpoint record structure for the keyboard
char buf[ 8 ] = { 0 };      //keyboard buffer
char old_buf[ 8 ] = { 0 };  //last poll

/* Sticky key state */
bool numLock = false;
bool capsLock = false;
bool scrollLock = false;

void setup();
void loop();

MAX3421E Max;
USB Usb;

void setup()
{
  Serial.begin(115200);
  Serial.println("Start ET-MEGA2560-ADK : Demo USB HID Keyboard Interface");
  Max.powerOn();
  delay( 200 );
}

void loop()
{
  Max.Task();
  Usb.Task();

  //wait for addressing state
  if( Usb.getUsbTaskState() == USB_STATE_CONFIGURING )
  {
    kbd_init();
    Usb.setUsbTaskState( USB_STATE_RUNNING );
  }

  //poll the keyboard
  if( Usb.getUsbTaskState() == USB_STATE_RUNNING )
  {
    kbd_poll();
  }
}

/* Initialize keyboard */
void kbd_init( void )
{
  byte rcode = 0;  //return code

  /* Initialize data structures */
  ep_record[ 0 ] = *( Usb.getDevTableEntry( 0,0 ));  //copy endpoint 0 parameters
  ep_record[ 1 ].MaxPktSize = EP_MAXPKTSIZE;
  ep_record[ 1 ].Interval  = EP_POLL;
  ep_record[ 1 ].sndToggle = bmSNDTOG0;
  ep_record[ 1 ].rcvToggle = bmRCVTOG0;
  Usb.setDevTableEntry( 1, ep_record );              //plug kbd.endpoint parameters

  /* Configure device */
  rcode = Usb.setConf( KBD_ADDR, 0, 1 );
  if( rcode )
  {
    Serial.print("Error attempting to configure keyboard. Return code :");
    Serial.println( rcode, HEX );
    while(1);  //stop
  }

  /* Set boot protocol */
  rcode = Usb.setProto( KBD_ADDR, 0, 0, 0 );
  if( rcode )
  {
    Serial.print("Error attempting to configure boot protocol. Return code :");
    Serial.println( rcode, HEX );
    while( 1 );  //stop
  }

  Serial.println("Keyboard initialized OK");
  Serial.print(">");
}
```

```
/* Poll keyboard and print result */
// USB Input Report Format = 8 Byte
// Byte[0] = Modifier Keys
// 8.3 Report Format for Array Items (HID1_11.pdf p56)
// ->Bit[0] = LEFT CTRL
// ->Bit[1] = LEFT SHIFT
// ->Bit[2] = LEFT ALT
// ->Bit[3] = LEFT GUI
// ->Bit[4] = RIGHT CTRL
// ->Bit[5] = RIGHT SHIFT
// ->Bit[6] = RIGHT ALT
// ->Bit[7] = RIGHT GUI

// Byte[1] = Reserve
// Byte[2] = Keycode[1]
// Byte[3] = Keycode[2]
// Byte[4] = Keycode[3]
// Byte[5] = Keycode[4]
// Byte[6] = Keycode[5]
// Byte[7] = Keycode[6]

/* buffer starts at position 2, 0 is modifier key state and 1 is irrelevant */
void kbd_poll( void )
{
  char i;
  static char leds = 0;
  byte rcode = 0;     //return code

  /* poll keyboard */
  rcode = Usb.inTransfer( KBD_ADDR, KBD_EP, 8, buf );
  if( rcode != 0 )
  {
    return;
  }//if ( rcode..

  for( i = 2; i < 8; i++ )
  {
    if( buf[ i ] == 0 )
    {  //end of non-empty space
      break;
    }
    if( buf_compare( buf[ i ] ) == false )
    {   //if new key
      switch( buf[ i ] )
      {
        case CAPSLOCK:    capsLock =! capsLock;
                          leds = ( capsLock ) ? leds |= bmCAPSLOCK :
                          leds &= ~bmCAPSLOCK;        // set or clear bit 1 of LED report byte
                          break;

        case NUMLOCK:     numLock =! numLock;
                          leds = ( numLock ) ? leds |= bmNUMLOCK :
                          leds &= ~bmNUMLOCK;         // set or clear bit 0 of LED report byte
                          break;

        case SCROLLLOCK:  scrollLock =! scrollLock;
                          leds = ( scrollLock ) ? leds |= bmSCROLLLOCK :
                          leds &= ~bmSCROLLLOCK;  // set or clear bit 2 of LED report byte
                          break;

        case DELETE:      break;

        case RETURN:      Serial.println();
                          Serial.print(">");
                          break;

        default:          Serial.print(HIDtoA( buf[ i ], buf[ 0 ] ));
                          break;
      }//switch( buf[ i ...

      rcode = Usb.setReport( KBD_ADDR, 0, 1, KBD_IF, 0x02, 0, &leds );
      if( rcode )
      {
        Serial.print("Set report error: ");
        Serial.println( rcode, HEX );
      }//if( rcode ...
```

```
    }//if( buf_compare( buf[ i ] ) == false ...
  }//for( i = 2...

  for( i = 2; i < 8; i++ )
  {                    //copy new buffer to old
    old_buf[ i ] = buf[ i ];
  }
}

/* compare byte against bytes in old buffer */
bool buf_compare( byte data )
{
  char i;
  for( i = 2; i < 8; i++ )
  {
    if( old_buf[ i ] == data )
    {
      return( true );
    }
  }
  return( false );
}

/* HID to ASCII converter. Takes HID keyboard scancode, returns ASCII code */
byte HIDtoA( byte HIDbyte, byte mod )
{
  /* upper row of the keyboard, numbers and special symbols */
  if( HIDbyte >= 0x1e && HIDbyte <= 0x27 ) // 1..9,0
  {
    if(( mod & SHIFT ) || numLock )
    {    //shift key pressed
      switch( HIDbyte )
      {
        case BANG:        return( 0x21 );   // 1 = !
        case AT:          return( 0x40 );   // 2 = @
        case POUND:       return( 0x23 );   // 3 = #
        case DOLLAR:      return( 0x24 );   // 4 = $
        case PERCENT:     return( 0x25 );   // 5 = %
        case CAP:         return( 0x5e );   // 6 = ^
        case AND:         return( 0x26 );   // 7 = &
        case STAR:        return( 0x2a );   // 8 = *
        case OPENBKT:     return( 0x28 );   // 9 = (
        case CLOSEBKT:    return( 0x29 );   // 0 = )
      }//switch( HIDbyte...
    }

    // 1..9,0
    else
    {
      if( HIDbyte == 0x27 )
      {
        return( 0x30 );                     // 0
      }
      else
      {
        return( HIDbyte + 0x13 );           // 0x1E..0x26 = 0x31(1)..0x39(9)
      }
    }//numbers
  }//if( HIDbyte >= 0x1e && HIDbyte <= 0x27

  /**/
  /* number pad. Arrows are not supported */
  //End(1) Down(2) PageDown(3) Left(4) 5 Right(6) Home(7) Up(8) PageUp(9)
  if(( HIDbyte >= 0x59 && HIDbyte <= 0x61 ) && ( numLock == true )) {  // numbers 1-9
    return( HIDbyte - 0x28 );               // 0x59..0x61 = 0x31(1)..0x39(9)
  }

  //Insert(0)
  if(( HIDbyte == 0x62 ) && ( numLock == true ))
  {
    return( 0x30 );                         // 0
  }

  /* Letters a-z */
  if( HIDbyte >= 0x04 && HIDbyte <= 0x1d )
  {
    if((( capsLock == true ) && ( mod & SHIFT ) == 0 ) ||
```

```
     (( capsLock == false ) && ( mod & SHIFT )))
  {
    return( HIDbyte + 0x3d );      // A..Z
  }
  else
  {
    return( HIDbyte + 0x5d );      // a..z
  }
}//if( HIDbyte >= 0x04 && HIDbyte <= 0x1d...

/* Other special symbols */
//Spacebar -(_) =(+) [({) ])}) \(|) # ;(:) '(") ~ ,(") .(>) /(?)
if( HIDbyte >= 0x2c && HIDbyte <= 0x38 )
{
  switch( HIDbyte )
  {
    case SPACE:       return( 0x20 );                // Space Bar

    case HYPHEN:      if(( mod & SHIFT ) == false )
                      {
                        return( 0x2d );              // -
                      }
                      else
                      {
                        return( 0x5f );              // _
                      }

     case EQUAL:      if(( mod & SHIFT ) == false )
                      {
                        return( 0x3d );              // =
                      }
                      else
                      {
                         return( 0x2b );             // +
                      }

    case SQBKTOPEN:   if(( mod & SHIFT ) == false )
                      {
                        return( 0x5b );              // [
                      }
                      else
                      {
                        return( 0x7b );              // {
                      }

    case SQBKTCLOSE:  if(( mod & SHIFT ) == false )
                      {
                        return( 0x5d );              // ]
                      }
                      else
                      {
                        return( 0x7d );              // }
                      }

    case BACKSLASH:   if(( mod & SHIFT ) == false )
                      {
                        return( 0x5c );              // ]
                      }
                      else
                      {
                        return( 0x7c );              // }
                      }

    case SEMICOLON:   if(( mod & SHIFT ) == false )
                      {
                        return( 0x3b );              // ;
                      }
                      else
                      {
                        return( 0x3a );              // :
                      }

    case INVCOMMA:    if(( mod & SHIFT ) == false )
                      {
                        return( 0x27 );              // ,
                      }
                      else
```

```
                        {
                          return( 0x22 );               // "
                        }

      case TILDE:       if(( mod & SHIFT ) == false )
                        {
                          return( 0x60 );               // `
                        }
                        else
                        {
                          return( 0x7e );               // ~
                        }

      case COMMA:       if(( mod & SHIFT ) == false )
                        {
                          return( 0x2c );               // ,
                        }
                        else
                        {
                          return( 0x3c );               // <
                        }

      case PERIOD:      if(( mod & SHIFT ) == false )
                        {
                          return( 0x2e );               // .
                        }
                        else
                        {
                          return( 0x3e );               // >
                        }

      case FRONTSLASH:  if(( mod & SHIFT ) == false )
                        {
                          return( 0x2f );               // /
                        }
                        else
                        {
                          return( 0x3f );               // ?
                        }

      default:          break;
    }//switch( HIDbyte..
  }//if( HIDbye >= 0x2d && HIDbyte <= 0x38..
  return( 0 );
}
```