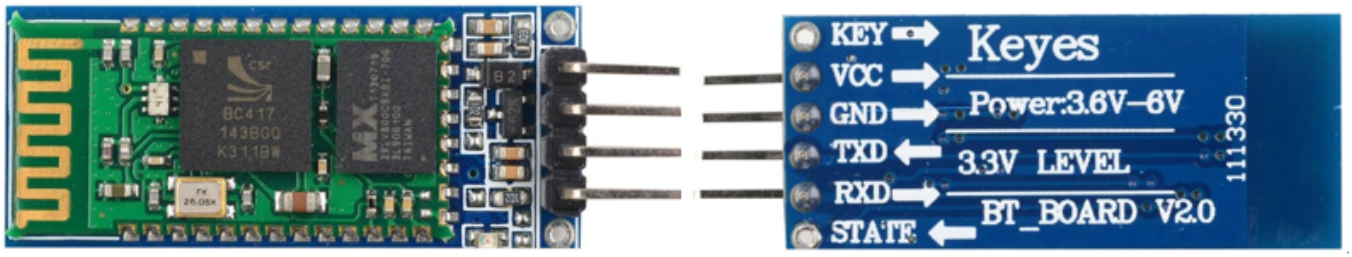
UART Lab Report

Bluetooth UART Lab

We are given a Bluetooth HC-06 UART device where we can talk between divices and send data in UART (Universal Asychronous Reciver Transmitter) format over a bluetooth connection.



Front Back

The device has 4 pins, VCC, GND, TX, RX. We connect the power to a 5V port on our board, because HC-06 accepts up to 6 volts. To make sure our connection is sound, the RX recive pin on one device should connect to the TX transmit pin on another device, so that information can pass through between them.

There are dedicated UART connections on our discovery board.

Pins\AF	AF8
PF6	UART7_RX
PF7	UART7_TX

These ports need to be placed in Alternative Function mode, but luckily we don't have to do all the manual initialization and handling of the pins and ports. Keil provides ARM drivers to access the different UART ports on the discovery board.

After enabling the CMSIS USART drivers, we can include the USART drivers in our C file and use the

```
1 extern ARM_DRIVER_USART Driver_USART7;
```

structure to interact with the connection.

Digital Bluetooth Picture Frame

With the UART bluetooth device, we'll make a wireless display that can show images and pictures. The idea is to send the image data across the UART bluetooth serial connection, and draw the image on the discovery board's LCD screen.

Client Side

On the embedded device, after establishing a connection, we need to recive the bytes, and draw those bytes to the

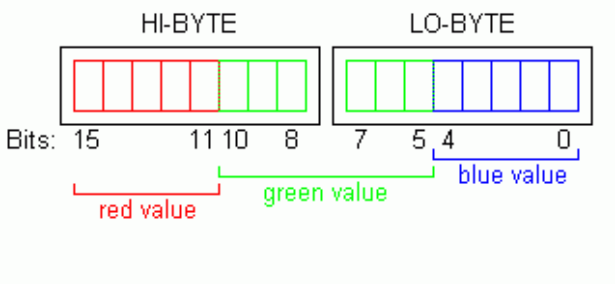
screen, but lets do a back and forth handshake to make sure the connection works.

```
Driver_USART7.Send("Hello World!\n\r", 14);
```

This means that the client will send the initiation, and the server will wait for the exact string, `Hello World! <clrf>`. The server would send back data, and we will verify it is the string `go to next 1`, and jump to the next stage in a state machine.

```
Driver_USART7.Receive(&recv, 12);
if (strcmp(recv, "go to next 1") == 0) ...
```

The next stage is where the server sends the pixel data to the client, and the client has to go in the same order to draw out the pixels. The LCD on the board uses rgb565 color format, so when we recive a byte at a time, we have to combine them two at a time and make it into 16 bits for the forground color.




```
    rgb = (rg << 8) | gb;
```

We draw row by column, and use the color data to draw a single pixel.

```
for (int y = 0; y < 272; y++) {
  for (int x = 0; x < 480; x++) {
  draw_pixel(x, y);
  }
}
```

And at the very end, we jump back to the beginning of the loop to wait for another image.

Master Side

The bluetooth master is our host mac computer with a python3 script. The python script uses the pyserial, and pillow (image manipulation) librarys to send the data to the client. The script takes any image format, scale it, and turns it into two bytes that is in rgb565 format and sends it to the client.

Since we are sending an image, that is a lot of data and would take a long time if we send it at slow speeds. The command `stty -f /dev/cu.Embedded2-DevB 115200` changes supposingly the speed of our virtual serial over bluetooth connections. The host code waits for the client hello:

```
greet = ser.read(14)
if greet != b'Hello World!\n\r':
    print('Wrong handshake')
    exit(1)
```

And then proceeds to randomely select a file or multiple files from a folder to send once or multiple times.

```
if args.count and args.folder:
    for i in range(args.count):
        send_picture(args.folder + '/' + random.choice(os.listdir(args.folder)))
else:
    send_picture(args.image)
```

Full code

Embedded code:

```
#include "stm32f7xx_hal.h"              // Keil::Device:STM32Cube HAL:Common
#include "Driver_USART.h"               // ::CMSIS Driver:USART
#include "Board_GLCD.h"                 // ::Board Support:Graphic LCD
#include "stm32f7xx_hal.h"              // Keil::Device:STM32Cube HAL:Common
#include "GLCD_Config.h"                // Keil.STM32F746G-Discovery::Board Support:Graphic
LCD
#include <stdlib.h>
#include <string.h>

/* USART Driver */
extern ARM_DRIVER_USART Driver_USART7;
extern GLCD_FONT GLCD_Font_16x24;

static void SystemClock_Config(void);

char rg, gb;
short rgb;

void draw_pixel(int x, int y) {
    Driver_USART7.Receive(&rg, 1);
    while(Driver_USART7.GetRxCount() != 1);
    Driver_USART7.Receive(&gb, 1);
    while(Driver_USART7.GetRxCount() != 1);
    rgb = (rg << 8) | gb;
    GLCD_SetForegroundColor(rgb);
    GLCD_DrawPixel(x, y);
}

int main(void) {
    SystemClock_Config();
    GLCD_Initialize();
    Driver_USART7.Initialize(NULL);

    Driver_USART7.PowerControl(ARM_POWER_FULL);
    Driver_USART7.Control(ARM_USART_MODE_ASYNCHRONOUS |
                          ARM_USART_DATA_BITS_8 |
                          ARM_USART_PARITY_NONE |
                          ARM_USART_STOP_BITS_1 |
                          ARM_USART_FLOW_CONTROL_NONE, 115200);
    Driver_USART7.Control (ARM_USART_CONTROL_TX, 1);
    Driver_USART7.Control (ARM_USART_CONTROL_RX, 1);

    GLCD_SetBackgroundColor(GLCD_COLOR_BLUE);
    GLCD_SetForegroundColor(GLCD_COLOR_YELLOW);
    GLCD_SetFont(&GLCD_Font_16x24);
    GLCD_ClearScreen();

    Driver_USART7.Send("Hello World!\n\r", 14);

    char recv[256] = {0}, fmtdisp_str[32] = {0};
    short mode = 0;

```

```
52      while(1) {
53          switch(mode) {
54              case 0:
55                  // hello world! -> hello device
56                  Driver_USART7.Receive(&recv, 12);
57                  while(Driver_USART7.GetRxCount() != 12) {
58                      sprintf(fmtdisp_str, "RX Count: %02i", Driver_USART7.GetRxCount());
59                      GLCD_DrawString(50, 75, fmtdisp_str);
60                  }
61  //              Driver_USART7.Send(recv, 80);
62                  recv[13] = '\0';
63                  GLCD_DrawString(50, 100, recv);
64                  if (strcmp(recv, "go to next 1") == 0) {
65                      GLCD_ClearScreen();
66                      mode = 0x100;
67                  } else {
68  //                  GLCD_ClearScreen();
69  //                  GLCD_DrawString(50, 50, "Ending.");
70  //                  exit(0);
71                  }
72                  break;
73              case 1 ... 0xfe:
74  //              Driver_USART7.Receive(&mode, 2);
75  //              if (mode == 0xff) {
76  //                  mode = 0x100;
77  //                  break;
78  //              }
79  //              recv[mode] = '\0';
80  //              Driver_USART7.Receive(&recv, mode);
81  //              GLCD_DrawString()
82  //              Driver_USART7.Send(recv, mode);
83                  break;
84              case 0x100:
85                  for (int y = 0; y < 272; y++) {
86  //                  if (y % 20 == 0)
87  //                      Driver_USART7.Send("line", 4);
88                      for (int x = 0; x < 480; x++) {
89                          draw_pixel(x, y);
90                      }
91                  }
92                  mode = 0x100;
93                  break;
94  //          case 0x101:
95  //              for (int x = 0; x < 480; x++) {
96  //                  if (x % 40 == 0)
97  //                      Driver_USART7.Send("line", 4);
98  //                  for (int y = 0; y < 272; y++) {
99  //                      draw_pixel(x, y);
100 //                  }
101 //              }
102 //              mode = 0x102;
103 //              break;
104 //          case 0x102:
105 //              for (int y = 271; y > -1; y++) {
106 //                  if (y % 20 == 0)
107 //                      Driver_USART7.Send("line", 4);
```

```
//                  for (int x = 0; x < 480; x++) {
//                      draw_pixel(x, y);
//                  }
//              }
//              mode = 0x103;
//              break;
//          case 0x103:
//              for (int x = 479; x > -1; x++) {
//                  if (x % 40 == 0)
//                      Driver_USART7.Send("line", 4);
//                  for (int y = 0; y < 272; y++) {
//                      draw_pixel(x, y);
//                  }
//              }
//              mode = 0x100;

            break;
        default:
            break;
        }
    }
}

void SysTick_Handler(void) {
    HAL_IncTick();
}

static void SystemClock_Config(void) {
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_OscInitTypeDef RCC_OscInitStruct;
    HAL_StatusTypeDef ret = HAL_OK;

    /* Enable HSE Oscillator and activate PLL with HSE as source */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 25;
    RCC_OscInitStruct.PLL.PLLN = 432;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 9;

    ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
    if (ret != HAL_OK) {
        while (1) { ; }
    }

    /* Activate the OverDrive to reach the 216 MHz Frequency */
    ret = HAL_PWREx_EnableOverDrive();
    if (ret != HAL_OK) {
        while (1) { ; }
    }

    /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks
dividers */
    RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
```

```
RCC_CLOCKTYPE_PCLK1 |
                    RCC_CLOCKTYPE_PCLK2);
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
    if (ret != HAL_OK) {
        while (1) { ; }
    }
}

```

Python script for Master:

```
#!/usr/bin/env python3
from PIL import Image
import serial, time, argparse, os, random, PIL

size = 480, 272

def send_picture(image_loc):
    print('sending', image_loc)
    im = Image.open(image_loc)
    im = im.resize(size, PIL.Image.BICUBIC)

    elapsed = -time.perf_counter()
    for i, px in enumerate(im.getdata()):
        time.sleep(0.0002)
#        if i % (480*20) == 0:
#            read = ser.read(4)
#            print(read, i//480)
#            if read != b'line':
#                print('line failure')
#                exit(1)
        rg =   px[0] & 0b11111000
        rg |= (px[1] & 0b11100000) >> 5
        gb =  (px[1] & 0b00011100) << 3
        gb |= (px[2] & 0b11111000) >> 3
    #    print('%02x%02x' % (rg, gb), end=' ')
        ser.write(bytes([rg, gb]))
    elapsed += time.perf_counter()
    print('time elapsed', elapsed)

parser = argparse.ArgumentParser()
parser.add_argument('-c', '--count', type=int, default=1, help='number of pictures to
randomly select')
parser.add_argument('-f', '--folder', type=Path, help='folder to pick random images from')
parser.add_argument('-i', '--image', default='framed.bmp', help='single image to send')
args = parser.parse_args()

ser = serial.Serial('/dev/cu.Embedded2-DevB', 115200)  # open serial port
print(ser.name)         # check which port was really used
os.system('stty -f /dev/cu.Embedded2-DevB 115200')
print('set stty speed')

```

```
print('waiting for greet')
greet = ser.read(len(b'hello world!\n\r'))
print(greet)
if greet != b'Hello World!\n\r':
    print('Wrong handshake')
    exit(1)
print('greeted')

print('responding')
ser.write(b'hello device')
time.sleep(1)
ser.write(b'go to next 1')


if args.count and args.folder:
    for i in range(args.count):
        send_picture(args.folder + '/' + random.choice(os.listdir(args.folder)))
else:
    send_picture(args.image)

time.sleep(3)
ser.close()

print('finished')

```