GUI-O IoT Example
GUI-O master – GUI-O slave (STM)

Table of Contents

Required hardware....3
Required software....3
Step 1. Setup the environment....4
Setup Keil environment....4
Step 2. STM32F103C8T6 programming....5
Wiring connections....5
The code....7
Upload the code....11
Step 3. Application setup....11
At local Android device (LOCAL GUI-O app)....11
At remote Android device/s (REMOTE GUI-O app/s)....12

The example shows how the LOCAL GUI-O application communicates with STM32F103 controller using the HC-06 Bluetooth module and REMOTE GUI-O using the MQTT protocol. The concept is introduced by simply toggling a LED from the application side.

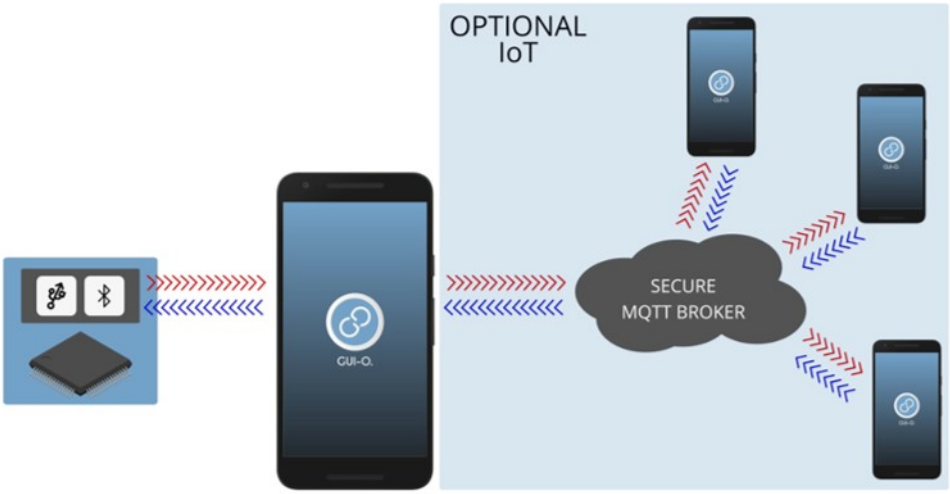

Figure 1: Schematic representation of Bluetooth, Bluetooth LE and USB connection with optional IoT. ***Left side*** *– any device with Bluetooth, Bluetooth LE or USB capabilities;* ***Middle*** *– local application;* ***Right side*** *– remote applications.*

Required hardware

- HC-06 Bluetooth module
- STM32F103C8T6 Blue pill Arduino module
- ST-Link V2 Emulator Downloader Programming
- LED integrated on STM32F103C8T6 module
- Wires for connections

Required software

- Keil MDK-ARM IDE
- minimal STM32F103 startup SW
- GUI-O application

Step 1. Setup the environment

Setup Keil environment

1. Download MDK-ARM from https://www.keil.com/demo/eval/arm.htm
2. Open new project: *Project → New uVision Project*
3. Set:
   - *Pack Installer → Devices → STMicroelectronics → STM32F103C8* and check for updates.
   - *Manage Run-time Environment* and select the following:
     - *CMSIS → CORE*
     - *DEVICE → Startup*
     - *DEVICE → StdPeriph Drivers → Framework, GPIO, RCC and USART* (see Figure 1).
   - ST-LINK debugger: *Projects -> Options for target* -> tab *Debug* -> use *ST-Link Debugger.*

Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
Board Support		MCBSTM32C	2.0.0	Keil Development Board MCBSTM32C
CMSIS				Cortex Microcontroller Software Interface Components
CORE	☑		5.4.0	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M
DSP	☐	Source	1.8.0	CMSIS-DSP Library for Cortex-M, SC000, and SC300
NN Lib	☐		1.3.0	CMSIS-NN Neural Network Library
RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
RTOS2 (API)			2.1.3	CMSIS-RTOS API for Cortex-M, SC000, and SC300
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
DMA	☐		1.2	DMA driver used by RTE Drivers for STM32F1 Series
GPIO	☐		1.3	GPIO driver used by RTE Drivers for STM32F1 Series
Startup	☑		1.0.0	System Startup for STMicroelectronics STM32F1xx device series
StdPeriph Drivers				
ADC	☐		3.5.0	Analog-to-digital converter (ADC) driver for STM32F10x
BKP	☐		3.5.0	Backup registers (BKP) driver for STM32F10x
CAN	☐		3.5.0	Controller area network (CAN) driver for STM32F1xx
CEC	☐		3.5.0	Consumer electronics control controller (CEC) driver for STM32F1xx
CRC	☐		3.5.0	CRC calculation unit (CRC) driver for STM32F1xx
DAC	☐		3.5.0	Digital-to-analog converter (DAC) driver for STM32F1xx
DBGMCU	☐		3.5.0	MCU debug component (DBGMCU) driver for STM32F1xx
DMA	☐		3.5.0	DMA controller (DMA) driver for STM32F1xx
EXTI	☐		3.5.0	External interrupt/event controller (EXTI) driver for STM32F1xx
FSMC	☐		3.5.0	Flexible Static Memory Controller (FSMC) driver for STM32F11x
Flash	☐		3.5.0	Embedded Flash memory driver for STM32F1xx
Framework	☑		3.5.1	Standard Peripherals Drivers Framework
GPIO	☑		3.5.0	General-purpose I/O (GPIO) driver for STM32F1xx
I2C	☐		3.5.0	Inter-integrated circuit (I2C) interface driver for STM32F1xx
IWDG	☐		3.5.0	Independent watchdog (IWDG) driver for STM32F1xx
PWR	☐		3.5.0	Power controller (PWR) driver for STM32F1xx
RCC	☑		3.5.0	Reset and clock control (RCC) driver for STM32F1xx
RTC	☐		3.5.0	Real-time clock (RTC) driver for STM32F1xx
SDIO	☐		3.5.0	Secure digital (SDIO) interface driver for STM32F1xx
SPI	☐		3.5.0	Serial peripheral interface (SPI) driver for STM32F1xx
TIM	☐		3.5.0	Timers (TIM) driver for STM32F1xx
USART	☑		3.5.0	Universal synchronous asynchronous receiver transmitter (USART) driver fo...

Figure 2: Run-Time Environment

Step 2. STM32F103C8T6 programming

STM32F103C8T6 will be communicating with GUI-O over software serial interface (i.e., emulated serial interface) with HC-06 module. Messages sent to the HC-06 module will be forwarded to the GUI-O application. In turn, all messages received from the GUI-O application will be forwarded to the STM32F103C8T6.

Wiring connections

1. Wire the STM32F1C8T6 board and ST-LinkV2 as shown in the Figure 3 and table below.

STM32F103C8T6	ST-Link V2
GND	GND (6)
SWCLK	SWCLK
SWDIO	SWDIO
5V	5V

2. Wire the STM32F1C8T6 board and HC-06 as shown in the Figure 3 and table below.

STM32F103C8T6	HC-06
A2 (PA2 – USART2 Tx)	RXD
A3 (PA3 – USART2 Rx)	TXD
SGND	GND
5V	VCC

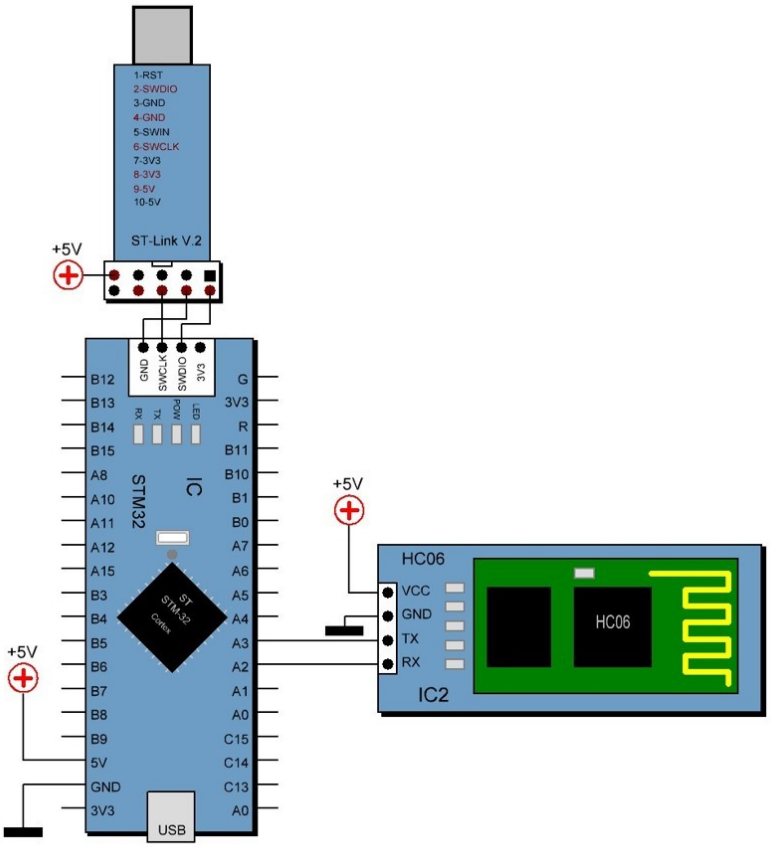

Figure 3: Wiring diagram – ST-Link V2 – STM32F103C8T6 - HC-06

The code

After including headers and setting up global variables, two functions are defined. One initializes general input/output, which interacts with the LED and the other sets communication and parses data:

```
// Initialize GPIOC PIN13 as push-pull output
void GPIOC_Init(void)
{
/* Bit configuration structure for GPIOC PIN13 */
GPIO_InitTypeDef gpioc_init_struct = { GPIO_Pin_13, GPIO_Speed_50MHz,
                                       GPIO_Mode_Out_PP };

/* Enable PORT C clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
/* Initialize GPIOC: 50MHz, PIN13, Push-pull Output */
GPIO_Init(GPIOC, &gpioc_init_struct);

/* Turn off LED to start with */
GPIO_SetBits(GPIOC, GPIO_Pin_13);
}


// Initialize USART2 for Bluetooth module: enable interrupt on reception
of a character
void USART2_Init(void)
{
/* USART configuration structure for USART2 */
USART_InitTypeDef usart2_init_struct;
/* Bit configuration structure for GPIOA PIN2 and PIN3 */
GPIO_InitTypeDef gpioa_init_struct;

/* Enable clock for USART2, AFIO and GPIOA */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

/* GPIOA PIN2 alternative function Tx */
gpioa_init_struct.GPIO_Pin = GPIO_Pin_2;
gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
gpioa_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &gpioa_init_struct);
/* GPIOA PIN3 alternative function Rx */
gpioa_init_struct.GPIO_Pin = GPIO_Pin_3;
gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
gpioa_init_struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &gpioa_init_struct);

/* Enable USART1 */
USART_Cmd(USART2, ENABLE);
/* Baud rate 38400, 8-bit data, One stop bit
* No parity, Do both Rx and Tx, No HW flow control*/
usart2_init_struct.USART_BaudRate = 38400;
usart2_init_struct.USART_WordLength = USART_WordLength_8b;
usart2_init_struct.USART_StopBits = USART_StopBits_1;
```

```
usart2_init_struct.USART_Parity = USART_Parity_No ;
usart2_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
usart2_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
/* Configure USART2 */
USART_Init(USART2, &usart2_init_struct);  //ttt
/* Enable RXNE interrupt */
USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
/* Enable USART2 global interrupt */
NVIC_EnableIRQ(USART2_IRQn);
}
```

Next, we setup USART2 interrupt request handler with HC-06 board. In USART2 interrupt request handler we read any data available on the serial interface.

```
//USART2 interrupt request handler: on reception of character from BlueTooth
modules characters load to input buffer

void USART2_IRQHandler(void)
{
unsigned int character; /* input character */
/* RXNE handler */
if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
  {
  character = (char)USART_ReceiveData(USART2);   /* get character direct from
data register    */
  if(character!=0)          /* continue only for non-NULL characters */
    {
     switch (character)
         {
        default:
        UART2_string[strcnt2++] = character;
/* write characters in the input buffer */
        break;
        case '\n': break;       /* new line */
        case '\r':              /* enter    */
        UART2_string[strcnt2] = 0;
/* replace enter with the 'null terminator' at the end of the line */
        parseData(&UART2_string[0], ' ');
/* Splitting a string according to given delimiters */
        processing();          /* creating actions */
        strcnt2 = 0;         /* empty buffer*/
        break;
        }  //switch
    }      //if(character!=0)
   }
}
```

When the GUI-O application sends the initialization request (**@init**), the command to create a toggle widget is sent as a response:

```
//Function Name  : processing
//Description    : carrying out actions recognizing the right argument
//Input          : None
//Output         : None
//Return         : None

void processing()
{
unsigned int argnr;          /* for deleting arguments */

//  GUI-O INTERPRETER
if ( !strcmp(argument2[0],"@init")  )     /* START BUTTON touch response    */
             {
/* GUI-O initialization block */
   sendstr2("@cls\n\r");  /* clear full screen - all local GUI-O elements */
   sendstr2("|TG UID:tg1 X:50 Y:30\n\r"); /* toggle with minimum parameters to
local      */
   sendstr2("@cls PUB:\"\"\n\r");   /* clear full screen - all remote GUI-O
elements*/
   sendstr2("|TG UID:tg1 X:50 Y:30 PUB:\"\"\n\r");  /* toggle with minimum
parameters to remote     */
             /* other graphical elements */
             }

else if (!strcmp(argument2[0],"@tg1"))       /* toggle touch response    */
             {
    if (!strcmp(argument2[1],"1"))
                  {
                   sendstr2("@tg1 EN:1\n\r");
                   sendstr2("@tg1 EN:1 PUB:\"\"\n\r");
                   led_toggle();       /* toggle LED with GUI-O */
                  }
    else if (!strcmp(argument2[1],"0"))
                  {
                   sendstr2("@tg1 EN:0\n\r");
                   sendstr2("@tg1 EN:0 PUB:\"\"\n\r");
                   led_toggle();       /* toggle LED with GUI-O */
                   }
             }


//***************** end commands and actions from UART2 ****************

for (argnr = 0; argnr < 5; argnr++)  argument2[argnr][0]=0;
// delete arguments after processing

}  //processing()
```

The essence of the code is string processing. Initialization is requested in both - LOCAL GUI-O and REMOTE GUI-O app (e.g., when initialization button is pressed both send the following command @init DPW:601 DPH:962\n\r). This can be simplified by using only one @init. However, in this case the information of the source (local or

remote) is lost and hence, the whole interface must be re-initialized in both GUI-O apps. Once the first initialization is done, it has to be cleared each time to avoid errors – duplicated elements. In the case when user interacts with toggle, again the same element is used for local and remote and therefore, the widget must be cleared each time.

TIP: At any start of GUI-O application, toggle is initialized with the state "0". This means that e.g., when REMOTE GUI-O starts it switches off the toggle that LOCAL GUI-O has switched on. Instead of using the following command for toggle initialization

```
  sendstr2("|TG UID:tg1 X:50 Y:30\n\r");          /* toggle init to local  */
```

use

```
  sendstr2("|TG UID:tg1 X:50 Y:30 EN:");    /* init toggle to local GUI-O */
  sendnr2(toggle);                          /* according to the state of LED
*/
  sendstr2("\n\r");
```

and check the previous status of toggle with EN:state.

For more information regarding the communication with more remote devices/users refer to the GUI-O developer manual.

When the user interacts with the widget on the LOCAL or REMOTE GUI-O application side, the appropriate function is called, which toggles the LED on the PC13 port:

```
//Toggle LED
void led_toggle(void)
{
/* Read LED output (GPIOA PIN8) status */
uint8_t led_bit = GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_13);

/* If LED output set, clear it */
if(led_bit == (uint8_t)Bit_SET)
    {
     GPIO_ResetBits(GPIOC, GPIO_Pin_13);
    }
    /* If LED output clear, set it */
else
    {
    GPIO_SetBits(GPIOC, GPIO_Pin_13);
    }
}
```

For more information see the comments in the full source code: **stm_led_bluetooth_mqtt.c**

Upload the code

1. Connect the STM32F103 to the PC via ST-LINK v2.
2. HC-06 module must be present / inserted into the STM32F103 board.
3. Download from the website **stm_led_bluetooth_mqtt.c** and save it to a location in the uVision project.
4. Add source to the project *Project -> Manage -> Project Items -> Add Files* **stm_led_control_bluetooth.c** or click on 
5.  Rebuild SW
6.  Download SW to STM32F103
7. After the code upload is complete, the STM32F103 built-in LED is turned off.

Step 3. Application setup

At local Android device (LOCAL GUI-O app)

1. Start the GUI-O application, go to *Settings* > *Connections* > *Bluetooth and IoT*.
2. Tap *Available devices* and wait for device search to complete.
3. Tap on your Bluetooth device when discovered and wait for successful connection.
   - If you set a toggle on *Settings* > *Connections* > *Bluetooth and IoT* > *Autoconnect*, after the application GUI-O starts, it will connect to the Bluetooth module automatically (see GUI-O developer manual for default settings).
4. Select *Settings* > *Quick pair* > *Generate QR code* > enter *Device name* and *user name.*
5. Scan the generated QR code with your remote device. After the confirmation, the generated data is saved.

   NOTE: to add a new user, repeat the steps 4 and 5.

At remote Android device/s (REMOTE GUI-O app/s)

1. Start the GUI-O application and select *IoT* connection under the connection settings (*Settings* > *Connections* > *Default connection* > *IoT*).
2. Select *Settings* > *Quick pair* > *Scan QR code,* after scanning the QR code from a local device, the generated data is saved.
3. Select *Settings* > *Connections* > *IoT* > *Connect*
   - If you set a toggle on *Settings* > *Connections* > *IoT* > *Autoconnect*, after the application GUI-O starts, it will connect to the IoT automatically (see GUI-O developer manual for default settings).

   NOTE: as described above there can be more remote devices, so steps 1 to 3 must be repeated for each remote device.
4. Close the settings menu and press the initialize button.
5. Interact with the toggle and observe the LED.

NOTE: For more information about the steps described here, see GUI-O developer manual.