Laboratorio di Architetture e Programmazione dei Sistemi Elettronici Industriali

Prof. Luca Benini <luca.benini@unibo.it>

Simone Benatti <simone.benatti@unibo.it>

Filippo Casamassima<filippo.casamassima@unibo.it>

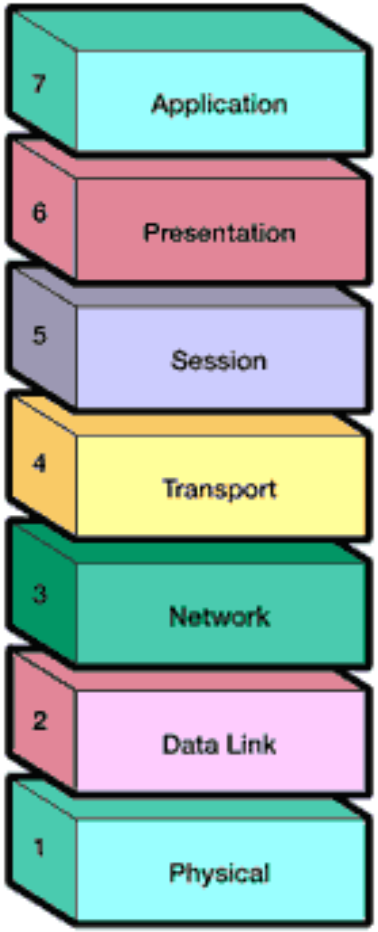
Bluetooth

Protocol layers

User application, usually, are built over Network layer

WSN protocols define lower levels

- Physical
- Data Link (MAC)
- Network



User specific

Protocols specific

Bluetooth: Intro

- Create a cable replacement standard for personal area network
- Handle simultaneously both data and voice between a wide range of devices

Figure 1: Personal Area Networking (PAN)

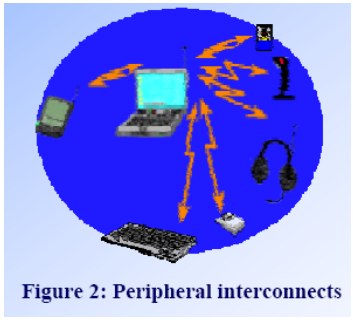


Figure 2: Peripheral interconnects

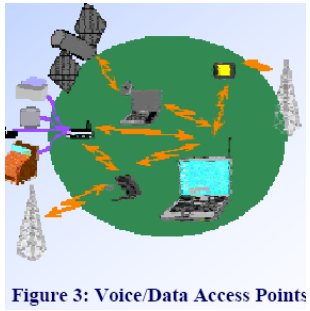
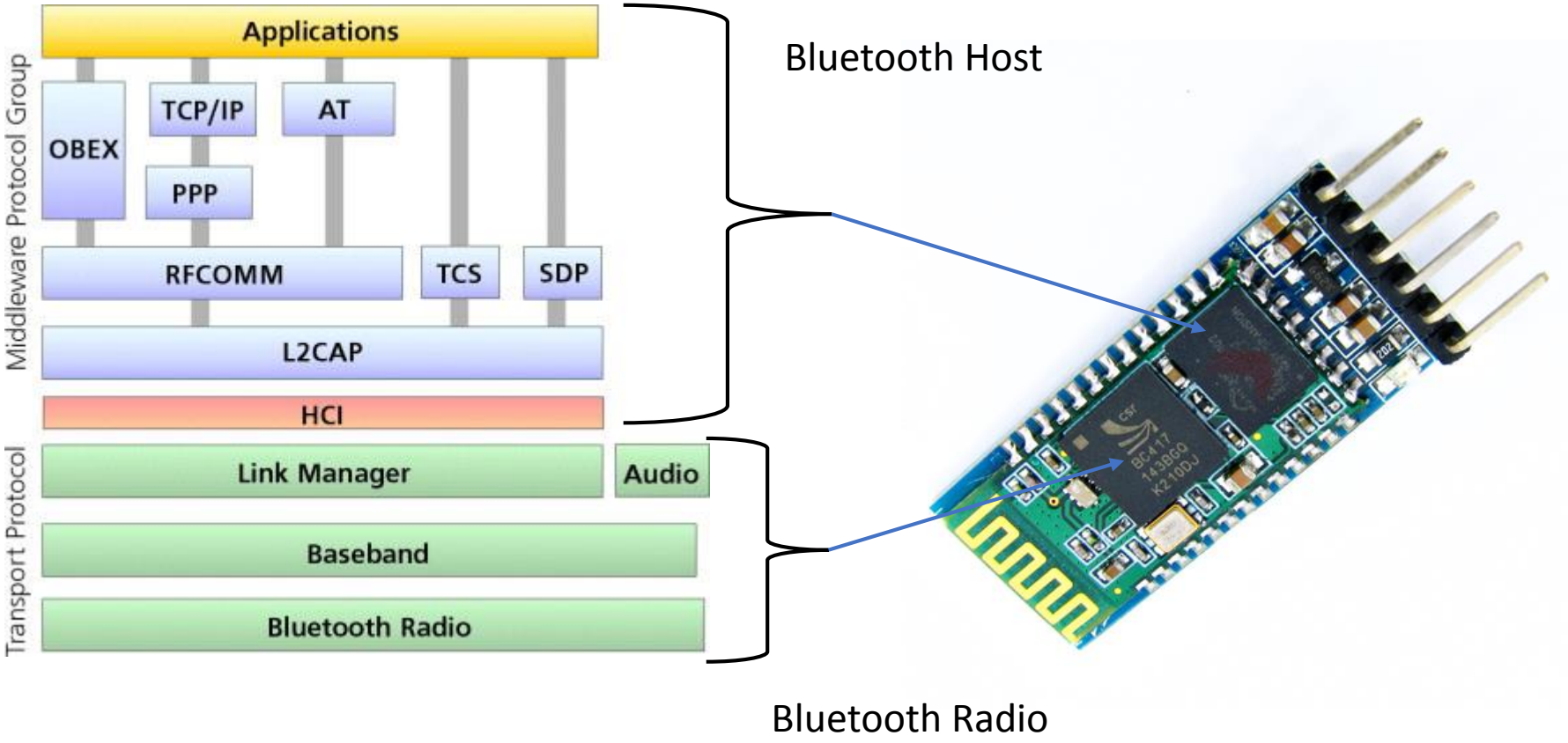


Figure 3: Voice/Data Access Points

Key features:

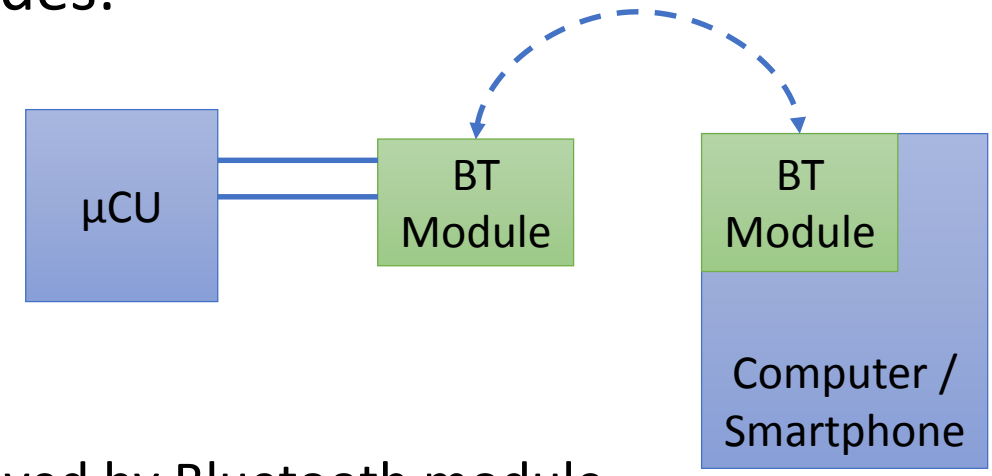
- **Master – Slave communication**
- **Network called Piconet (max 1 Master and 7 Slaves)**
- Interoperability
- Low complexity of use
- Low power (ca. 50mA while transmitting)
- Low cost ($10 per device)
- Small form factor (9mm^2 single chip)

Bluetooth Stack



Bluetooth Interface

- 2 different working modes:
  - Command mode
  - Data mode

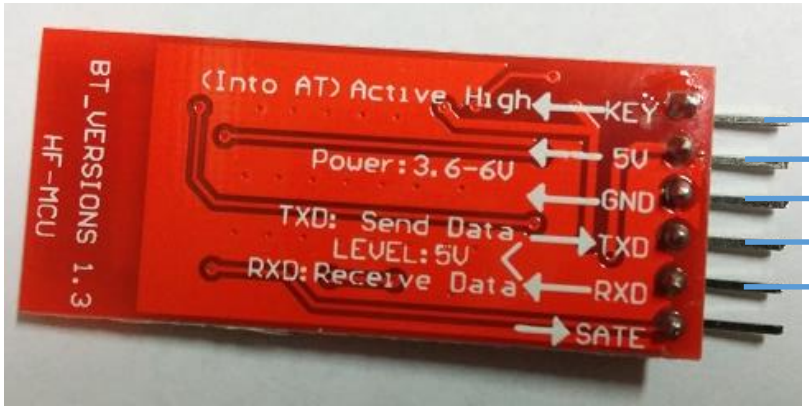


- Command mode:
  - AT commands are received by Bluetooth module (no Bluetooth connection)

- Data Mode (BT Connection estabilished):
  - Different Bluetooth profiles:
    - A2DP, ATT, GAP, HDP, HID, SPP
  - Serial Port Profile (SPP):
    - Emulates a serial cable to provide a simple substitute for existing RS-232

Bluetooth: Connection with the Board

Module model: HC - 05



BT Module	Discovery Board
Key	Any GPIO
5V	5v
Gnd	Gnd
TXD	PA3
RXD	PA2

Switch between command and Data mode

- The Bluetooth works in **Data Mode** when the **KEY** pin is pulled to logic 0 level or is left unconnected. Transparent UART data transfer with a connected remote device occurs only while in Data Mode.

- The Bluetooth switches to ***Command Mode*** if ***KEY*** pin is set to ***logic HIGH***. Command mode must be invoked to setup and configure. (Send AT Commands)

- If Bluetooth is in ***Command Mode*** (powered ON with KEY pin to +3.3v) the UART is set to **38400bps**, 8 data bits, 1 stop bit, no parity, no handshake.

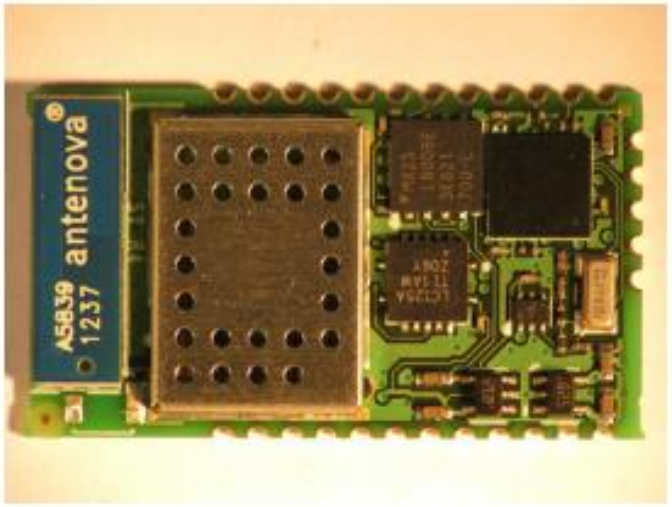
Exercise:

- Configure the Bluetooth (only one project, insert delays between commands):
  1. Start the Bluetooth in Command Mode
  2. Send AT command (“AT\r\n”)
  3. Look in the Appendix and send following commands
     1. **Get firmware version and save in a buffer**
     2. **Get device name, and change device name to your surname**
     3. **Change baud to 115200**
     4. **Reset the module**
- Send and Receive data (we will connect a PC to your Bluetooth)
  1. Periodically send your name to connected device
  2. Save incoming data in a buffer (read \r to detect end of data)

WiFi

Module presentation:

- Applications:
  - Smart appliances
  - Industrial control
  - Home automation
  - Wireless sensors
  - Cable replacement
  - Medical equipment
  - Machine-to-machine communication



SPWF01SA

- Description:
  - The SPWF01SA and the SPWF01SC intelligent Wi-Fi modules represent a plug and play and standalone 802.11 b/g/n solution for easy integration of wireless Internet connectivity features into existing or new products.

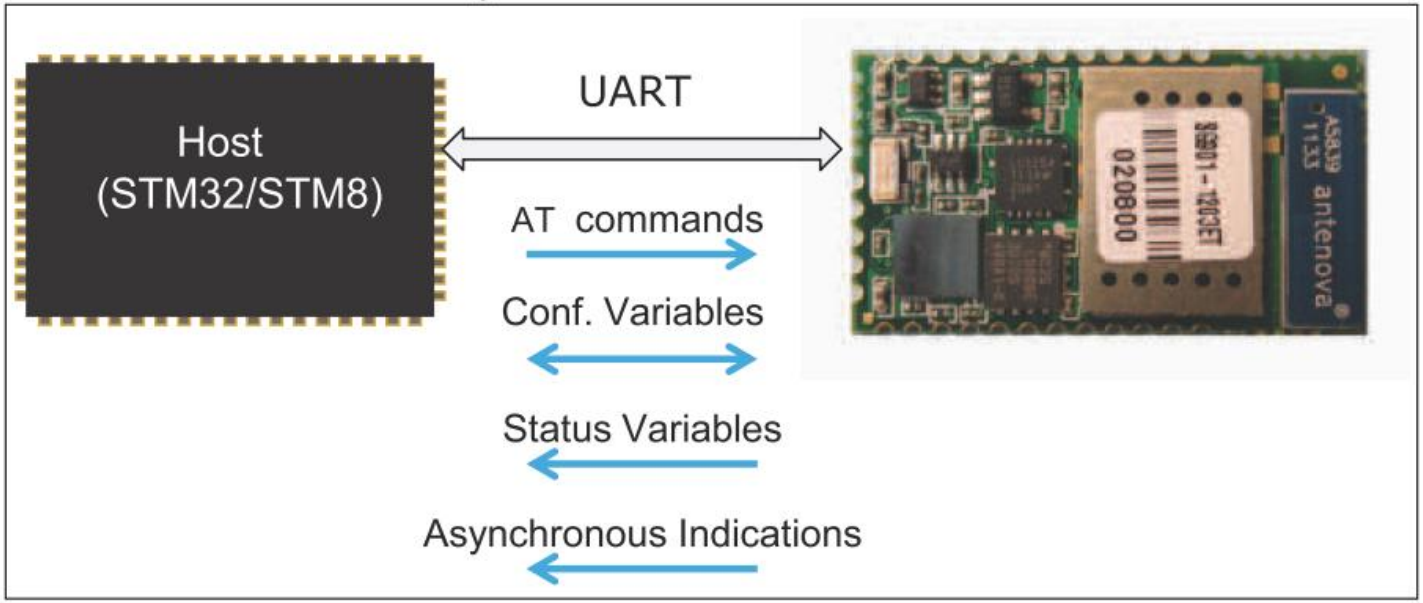
Module block diagram and interface:

The SPWF01Sx.11 Module command interface consists of a set of:

- AT-style commands,
- configuration variables,
- status variables, and
- asynchronous indications (also known as unsolicited responses or WINDs).

The communication of commands, variables, and asynchronous indications is executed via the serial port and implies the integration of the SPWF01Sx modules with a host processor as indicated in figure:

Figure 1. SPWF01Sx.11 interface



WiFi module communication

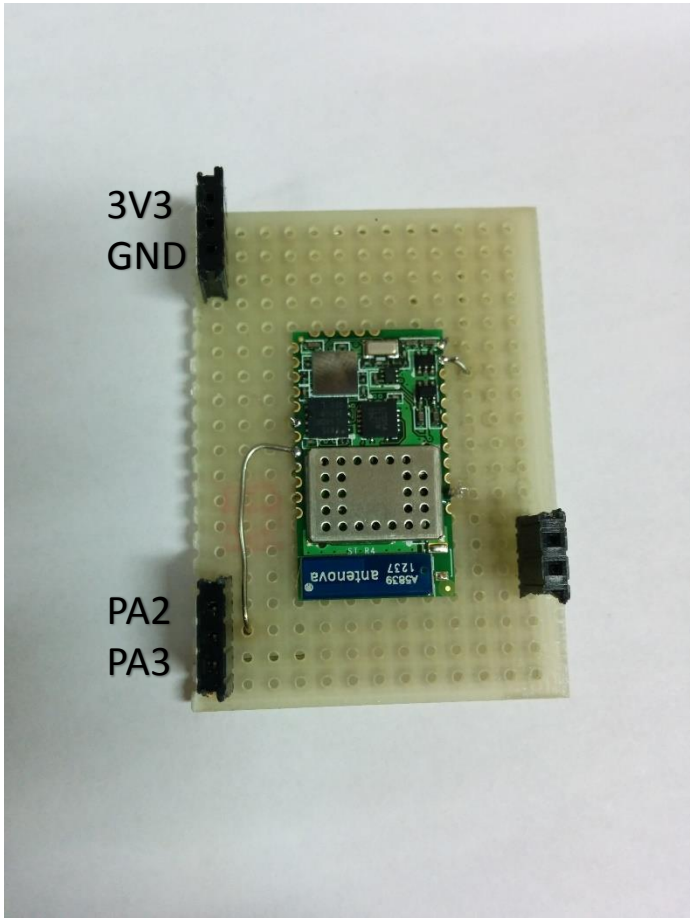
Asynchronous indications may arrive at any time (except as noted below), and have the format:
<cr><lf>+WIND:<number>:<descriptive string><cr><lf>
The <number> field of each asynchronous indication type is unique. The descriptive string may be safely ignored.

AT commands are always in the form of:
AT<cmd><cr>
<zero or more response lines>
<cr><lf><responsecode><cr><lf>

Example:
AT: Attention
AT, by itself, is a null command that always returns an OK result code. It is useful for testing the module interface for readiness.

AT<cr> *(invio)*
<cr><lf>OK<cr><lf> *(risposta)*

Module block diagram and interface:



Exercise:

- Configure the WiFi Module (check OK answer and insert delay between commands)
  1. Send AT command **AT\r**
  2. Set SSID name: **AT+S.SSIDTXT=GTVrouter\r**
  3. Set the password
     AT+S.SCFG=wifi_wpa_psk_text,aaaaaaaa\r
  4. Set the network privacy mode
     AT+S.SCFG=wifi_priv_mode,2\r
  5. Set the network mode (1 = BSS, 3 = MiniAP)
     AT+S.SCFG=wifi_mode,1\r
  6. Save Settings
     AT&W\r
  7. Reset the module:
  8. **AT+CFUN=1\r**

Exercise:

- Check result after reset (save result in a buffer and view in debugger (wait 10 seconds before check with breakpoint)):

+WIND:13:ST SPWF01SA1 IWM: Copyright (c) 2012-2014 STMicroelectronics, Inc. All rights Reserved.

+WIND:3:Watchdog Running

+WIND:0:Console active

+WIND:46:WPA: Crunching PSK...

+WIND:32:WiFi Hardware Started

+WIND:21:WiFi Scanning

+WIND:35:WiFi Scan Complete (0x0)

+WIND:19:WiFi Join: 34:08:04:0F:9D:FE

+WIND:25:WiFi Association with 'GTVrouter' successful

+WIND:51:WPA Handshake Complete

+WIND:24:WiFi Up: 192.168.0.101

Exercise:

- Perform a POST request with your name and the variable that will be assigned:
  - *at+s.httppost=**192.168.0.1xx**,/post.php,**studentN**=yourname*
  - ***192.168.0.1xx*** will be told during lesson
    a different **N** will be assigned to each group
- Perform a GET request:
  - AT+S.HTTPGET=192.168.1.1xx,/testget.html
  - Search your surname in RxBuffer, if found turn on a led

Appendix: example CODE and Buetooth command list

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include <stdio.h>
#include <string.h>
// EDIT FILE stm32f10x_conf.h to add USART includes


/* Private typedef -----------------------------------------------------------*/
typedef enum { FAILED = 0, PASSED = !FAILED} TestStatus;

/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
const u16 RxBufferSize1 = 2048;
const u16 TxBufferSize1 = 128;
USART_InitTypeDef USART_InitStructure;
char RxBuffer1[RxBufferSize1];
char* TxBuffer1;
bool byteReceived = FALSE;
volatile uint32_t TimingDelay;

/* Private function prototypes -----------------------------------------------*/


/* Private function prototypes -----------------------------------------------*/
void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void Delay(__IO uint32_t nTime);
// TestStatus Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength);


/* Private functions ---------------------------------------------------------*/

/**
  * @brief   Main program
  * @param  None
  * @retval None
  */
int main(void)
{
 char* String1 = "AT\r\n";
 char* strToFind = "OK\r";
 bool StingFound = FALSE;
  /* System Clocks Configuration */
  RCC_Configuration();

  /* NVIC configuration */
  NVIC_Configuration();

  /* Configure the GPIO ports */
  GPIO_Configuration();

  USART_InitStructure.USART_BaudRate = 9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
```

```
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  /* Configure USART2 */
  USART_Init(USART2, &USART_InitStructure);

  /* Enable USART2 Receive and Transmit interrupts */
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

  /* Enable the USART2 */
  USART_Cmd(USART2, ENABLE);

//Systick is used to generate precise delays
    SysTick_Config(SystemCoreClock/1000);

    //Use the following function to insert a Delay
    //Delay(int milliseconds)
    Delay(500);

    //Assign to TxBuffer the string you want to write and then enable tx interrupt
    TxBuffer1 = String1;
    USART_ITConfig(USART2, USART_IT_TXE, ENABLE);

  while (1)
  {
        if (byteReceived == TRUE){
            byteReceived = FALSE;

            //Search for a determined string in the RX Buffer (change strToFind)
            if(strstr(RxBuffer1,strToFind)){
                StingFound = TRUE;
            }
        }
    }
}

/**
  * @brief  Configures the different system clocks.
  */
void RCC_Configuration(void)
{
  /* Enable GPIO clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
  /* Enable USART clock*/
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
}

/**
  * @brief  Configures the different GPIO ports.
  */
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Configure USART2 Tx as alternate function push-pull */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
```

```
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Configure USART2 Rx as input floating */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}

/**
  * @brief  Configures the nested vectored interrupt controller.
  */
void NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure the NVIC Preemption Priority Bits */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

  /* Enable the USART2 Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

}

/**
  * @brief  Wait untill variable is decremented by systick
  */
void Delay(__IO uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}
```

```c
/* Includes ------------------------------------------------------------------*/
#include "stm32f10x_it.h"


/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
extern char* TxBuffer1;
extern char RxBuffer1[];
u16 TxCounter1;
u16 RxCounter1;
extern u16 RxBufferSize1;
extern bool byteReceived;
extern volatile uint32_t TimingDelay;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M3 Processor Exceptions Handlers                         */
/******************************************************************************/

/**
  * @brief  This function handles NMI exception.
  * @param  None
  * @retval None
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
```

```
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
  * @retval None
  */
void DebugMon_Handler(void)
{
}

/**
  * @brief  This function handles PendSV_Handler exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
{
}

/**
  * @brief  This function handles SysTick Handler.
```

```
  * @param  None
  * @retval None
  */
void SysTick_Handler(void)
{
       if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}




/******************************************************************************/
/*            STM32F10x Peripherals Interrupt Handlers                        */
/******************************************************************************/

/**
  * @brief  This function handles USART2 global interrupt request.
  * @param  None
  * @retval None
  */

void USART2_IRQHandler(void)
{

  if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
  {
        RxBuffer1[RxCounter1++] = USART_ReceiveData(USART2);
        byteReceived = TRUE; //Set a flag to be used in main
        //Add the received character to a buffer
        if (RxCounter1 == (RxBufferSize1-1))
            RxCounter1 =0;

  }

  if(USART_GetITStatus(USART2, USART_IT_TXE) != RESET)
  {
        //Check if the character to be sent is the last one
        if(TxBuffer1[TxCounter1] == 0)
    {
      /* Disable the USART2 Transmit interrupt */
      USART_ITConfig(USART2, USART_IT_TXE, DISABLE);
            TxCounter1 = 0;
    }
        else{
            /* Write one byte to the transmit data register */
            USART_SendData(USART2, TxBuffer1[TxCounter1++]);
        }
  }
}



/******************************************************************************/
/*                 STM32F10x Peripherals Interrupt Handlers                   */
```

```
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f10x_xx.s).                                            */
/******************************************************************************/

/**
  * @brief  This function handles PPP interrupt request.
  * @param  None
  * @retval None
  */
/*void PPP_IRQHandler(void)
{
}*/


/****************** (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```

EGBT-045MS PIN CONFIGURATION

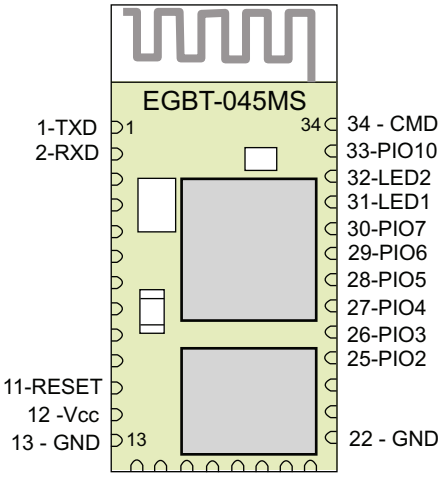


Figure 5. *EGBT-045MS pin layout*

Table 1. EGBT-045MS Pin Description

PIN	ID	DESCRIPTION
1	TXD	UART TXD Output
2	RXD	UART RXD Input
11	RESET	RESET Input
12	Vcc	+3.1 to 4.2VDC Power Input
13	GND	Common Ground
22	GND	Common Ground
25	PIO2	User programmable I/O
26	PIO3	User programmable I/O
27	PIO4	User programmable I/O
28	PIO5	User programmable I/O
29	PIO6	User programmable I/O
30	PIO7	User programmable I/O
33	PIO10	User programmable I/O
31	LED1	LED Status Indicator
32	LED2	LED Status Indicator
34	CMD	Command Mode

Note:
All unassigned pins must be left unconnected.

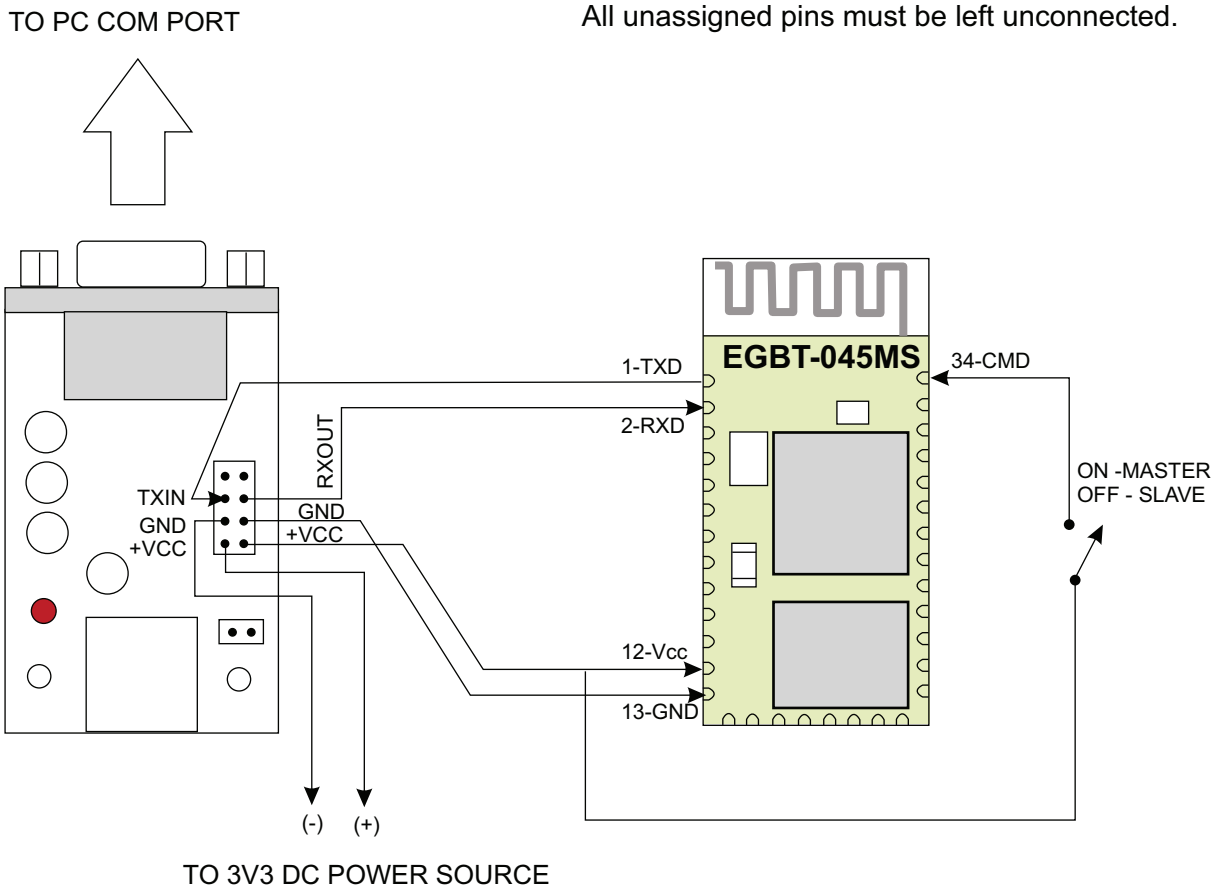


Figure 6. *Connecting the EGBT-045MS to a PC for test and configuration. To connect to a PC COM port, a RS-232C to TTL converter is needed. This figure shows a wiring example using e-Gizmo RS-232 to TTL converter kit.*

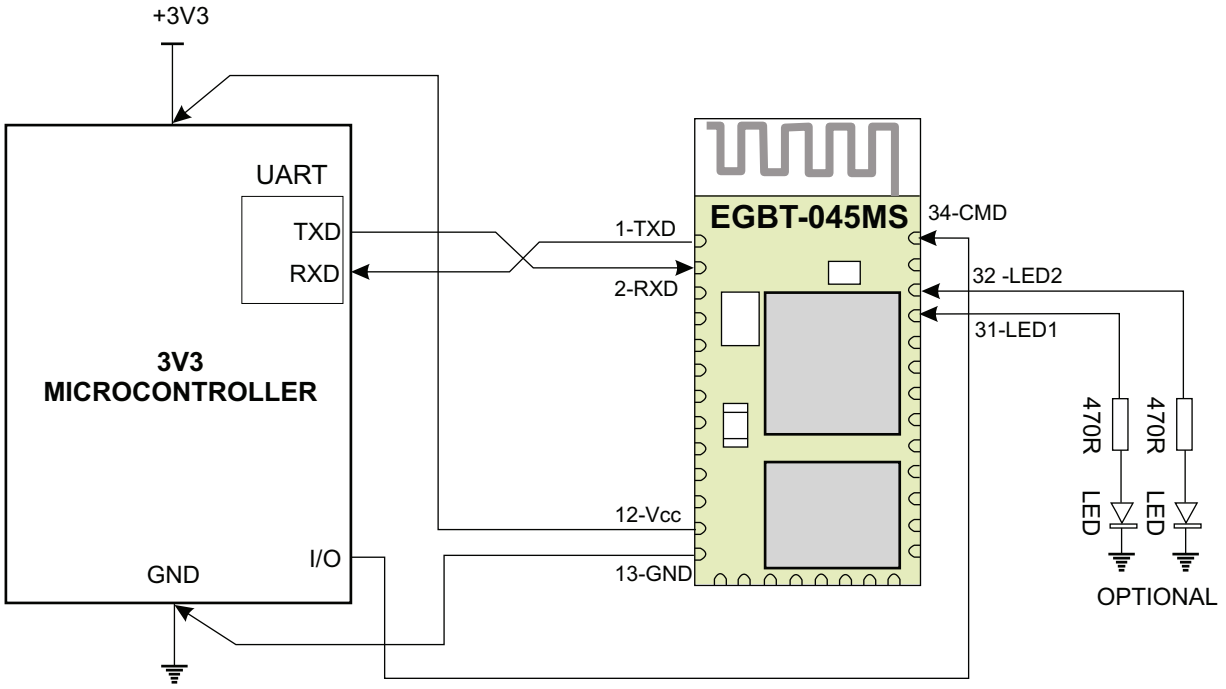


Figure 7. *EGBT-045MS wiring example with a 3v3 host microcontroller. The 470R resistors and LEDs are for status indication, and may be omitted if not needed. The microcontroller can switch the EGBT-045MS between command mode and data mode at any time by setting the CMD pin accordingly.*

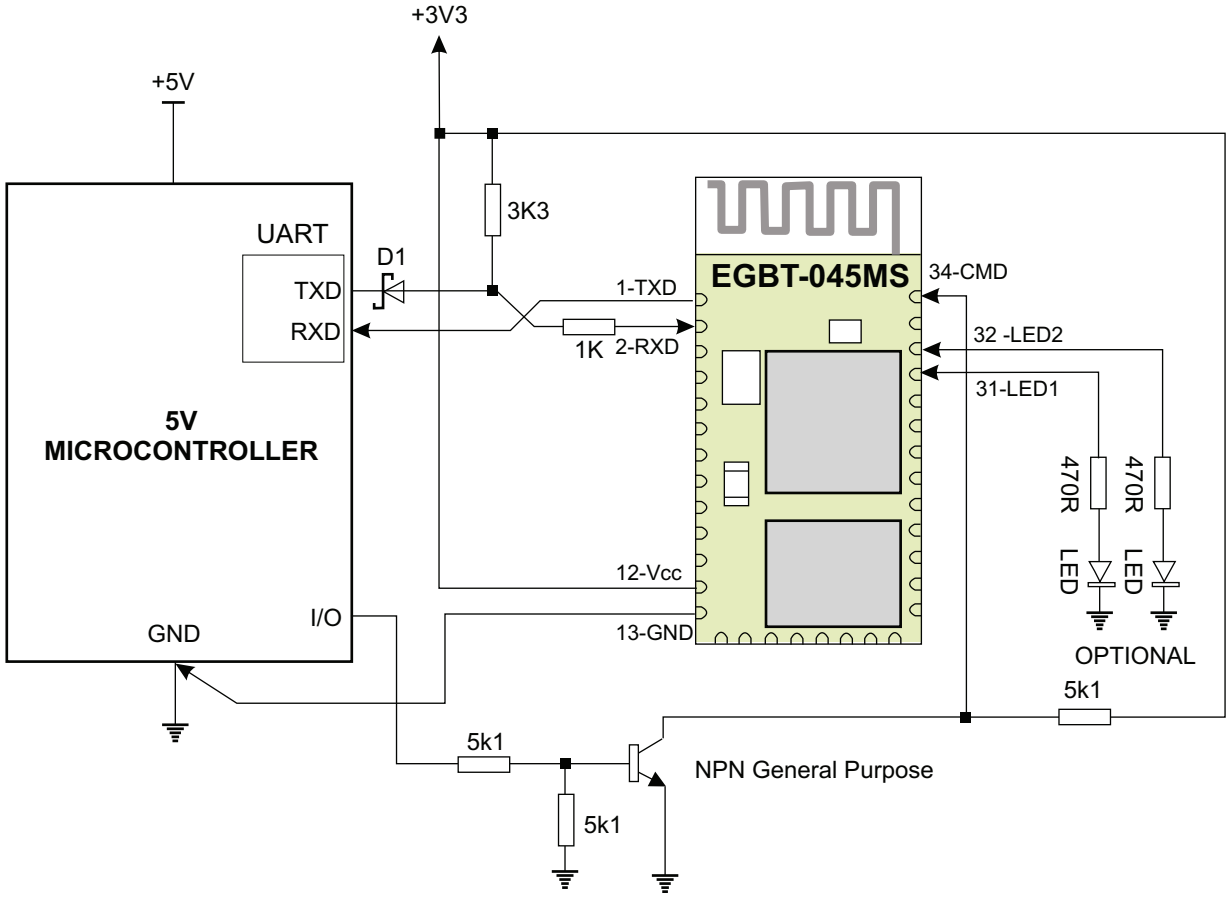


Figure 8. *EGBT-045MS RX input is not 5V tolerant. A schottky diode connected as shown will keep 5V voltages out of the Bluetooth module when operated with a 5V host microcontroller. The transistor circuit takes on a similar function for the CMD pin.*

PREPARATION FOR USE

The EGBT-045MS is pre-configured as a slave Bluetooth device. As shipped, it works under the following data mode default configuration:

Baud Rate: 38400/9600 bps – see “Entering Command Mode” section for details
Data : 8 bits
Stop Bits: 1 bit
Parity : None
Handshake: None

Passkey: 1234
Device Name: HC-05

If the default configuration suits your application, then you can use EGBT-045MS immediately. Once it is paired to a master Bluetooth device, its operation becomes transparent to the user. No user code specific to the Bluetooth module is needed at all in the user microcontroller program.

The EGBT-045MS works in Data Mode when the CMD pin (pin 34) is pulled to logic 0 level or is left unconnected. Transparent UART data transfer with a connected remote device occurs only while in Data Mode. The EGBT-045MS switches to Command Mode if CMD pin is set to logic HIGH. Command mode must be invoked to setup and configure EGBT-045MS. Pairing and connection to a remote Bluetooth slave device can be initiated only while the EGBT-045MS is in command mode (and configured in role as a Master) by entering a string of AT style commands. Any changes made to system parameters (e.g. password, baud rate, etc) will be retained even after power is removed, hence device configuration setup must not be repeated unless new changes need to be made.

Entering Command Mode

EGBT-045MS is put in Command Mode by setting the CMD pin to logic High. There is one important point the user should be aware of when setting the device in Command Mode – the baud rate may assume a different value depending on the instance the CMD pin is switched to high:

• If EGBT-045MS is powered ON with CMD pin to +Vcc, the UART is set to 38400bps, 8 data bits, 1 stop bit, no parity, no handshake.

• If the EGBT-045MS is powered ON with the CMD pin at logic low (or open circuit), and then pulled High a moment later, the UART parameters assumes a set of values that was previously fixed using the AT+UART command, with 9600bps, 8 data bits, 1 stop bit, no parity, and no handshake as default parameter values.

You can do configuration setup using the host controller itself (the microcontroller in your own circuit), or a PC running a terminal software using a serial to TTL (or USB to Serial TTL) converter. See Figure 6 for connection details.

EGBT-045MS AT COMMAND REFERENCE

Symbols and Nomenclatures.

<xxxx> Descriptive ID of parameters that must be entered by the user or reported back by the EGBT-045MS.

Example:
AT+INQM=<inq1>,<inq2>,<inq3>
May in actual use appear as
AT+INQM=0,1,15
When <inq1> =0, <inq2>=1, and <inq3>=15

↵ = Carriage Return followed by Line Feed character<CR><LF>

Example:

OK↵ = OK<CR><LF>

Important note: All commands must be terminated by <CR><LF>. If the host controller send a <CR> only, EGBT-045MS will repeatedly send a respond that will stop only when <LF> is issued by the host controller

AT COMMAND LISTING

	COMMAND	FUNCTION
1	AT	Test UART Connection
2	AT+RESET	Reset Device
3	AT+VERSION	Querry firmware version
4	AT+ORGL	Restore settings to Factory Defaults
5	AT+ADDR	Query Device Bluetooth Address
6	AT+NAME	Query/Set Device Name
7	AT+RNAME	Query Remote Bluetooth Device's Name
8	AT+ROLE	Query/Set Device Role
9	AT+CLASS	Query/Set Class of Device CoD
10	AT+IAC	Query/Set Inquire Access Code
11	AT+INQM	Query/Set Inquire Access Mode
12	AT+PSWD	Query/Set Pairing Passkey
13	AT+UART	Query/Set UART parameter
14	AT+CMODE	Query/Set Connection Mode
15	AT+BIND	Query/Set Binding Bluetooth Address
16	AT+POLAR	Query/Set LED Output Polarity
17	AT+PIO	Set/Reset a User I/O pin
18	AT+MPIO	Set/Reset multiple User I/O pin
19	AT+MPIO?	Query User I/O pin
20	AT+IPSCAN	Query/Set Scanning Parameters
21	AT+SNIFF	Query/Set SNIFF Energy Savings Parameters
22	AT+SENM	Query/Set Security & Encryption Modes
23	AT+RMSAD	Delete Authenticated Device from List
24	AT+FSAD	Find Device from Authenticated Device List
25	AT+ADCN	Query Total Number of Device from Authenticated Device List
26	AT+MRAD	Query Most Recently Used Authenticated Device
27	AT+STATE	Query Current Status of the Device
28	AT+INIT	Initialize SPP Profile
29	AT+INQ	Query Nearby Discoverable Devices
30	AT+INQC	Cancel Search for Discoverable Devices
31	AT+PAIR	Device Pairing
32	AT+LINK	Connect to a Remote Device
33	AT+DISC	Disconnect from a Remote Device
34	AT+ENSNIFF	Enter Energy Saving mode
35	AT+EXSNIFF	Exit Energy Saving mode

ERROR CODES

ERROR CODE	VERBOSE
0	Command Error/Invalid Command
1	Results in default value
2	PSKEY write error
3	Device name is too long (>32 characters)
4	No device name specified (0 lenght)
5	Bluetooth address NAP is too long
6	Bluetooth address UAP is too long
7	Bluetooth address LAP is too long
8	PIO map not specified (0 lenght)
9	Invalid PIO port Number entered
A	Device Class not specified (0 lenght)
B	Device Class too long
C	Inquire Access Code not Specified (0 lenght)
D	Inquire Access Code too long
E	Invalid Iquire Access Code entered
F	Pairing Password not specified (0 lenght)
10	Pairing Password too long (> 16 characters)
11	Invalid Role entered
12	Invalid Baud Rate entered
13	Invalid Stop Bit entered
14	Invalid Parity Bit entered
15	No device in the Pairing List
16	SPP not initialized
17	SPP already initialized
18	Invalid Inquiry Mode
19	Inquiry Timeout occured
1A	Invalid/zero lenght address entered
1B	Invalid Security Mode entered
1C	Invalid Encryption Mode entered

1. Test UART connection

COMMAND	RESPONSE
AT↵	OK↵

2. Reset Device

COMMAND	RESPONSE
AT+RESET↵	OK↵

3. Querry firmware version

COMMAND	RESPONSE
AT+VERSION?↵	+VERSION:<VER>↵ OK↵

where <VER> = Version Number

4. Restore settings to Factory Defaults

COMMAND	RESPONSE
AT+ORGL↵	OK↵

Restore to the following settings:
Device Class: 0
Inquiry Code: 0x009e8b33
Device Mode: Slave
Binding Mode: SPP
UART: 38400bps, 8 bit, 1 stop bit, no parity
Pairing Code: 1234
Device Name: H-C-2010-06-01

5. Query EGBT-045MS Bluetooth Address

COMMAND	RESPONSE
AT+ADDR?↵	+ADDR:nn:uu:ll↵ OK↵

Bluetooth Address Format:
nn - NAP (16 bit Non-significant Address Portion)
uu - UAP (8 bit Upper Address Portion)
ll - LAP (24 bit Lower Address Portion)
Returned bluetooth address

Example: Querry EGBT-045MS Bluetooth Address

From Host controller:
AT+ADDR?↵
EGBT-045MS Response
+ADDR:11:6:230154↵
OK↵

Bluetooth Address is 11:06:23:01:54

6. Query/Set Device Name

COMMAND	RESPONSE
AT+NAME?↵	+NAME:<name>↵ OK↵
AT+NAME=<name>↵	OK↵

where <name> = Device Name (31 characters max.)

Example: Query device name

From Host controller:
AT+NAME?↵
EGBT-045MS Response
+NAME:HC-05↵
OK↵

Example: Set device name to “e-Gizmo”

From Host controller:
AT+NAME=e-Gizmo↵
EGBT-045MS Response
OK↵

Example: Set device name to “supercalifragilisticexpialidocious”

From Host controller:
AT+NAME=supercalifragilisticexpialidocious↵
EGBT-045MS Response
ERROR:(3)↵ *name too long (>31 characters)*

7. Query Remote Bluetooth Device’s Name

COMMAND	RESPONSE
AT+RNAME?<addr>↵	+NAME:<name>↵ OK↵

where <name> = Device name
<addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Example: Query remote Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+RNAME?0002,72,0A3C7F↵
EGBT-045MS response if remote device name is “HC-05”
+NAME:HC-05↵
OK↵
EGBT-045MS response if remote device name is unresolved
FAIL↵

8. Query/Set Device Role

COMMAND	RESPONSE
AT+ROLE?↵	+ROLE:<role>↵ OK↵
AT+ROLE=<role>↵	OK↵

where <role>
0 - Slave (default)
1 - Master
2 - Slave-Loop

Slave - EGBT-045MS acts as discoverable wireless UART device ready for transparent data exchange.
Master - Scans for a remote bluetooth (slave) device, pairs, and setup connection for a transparent data exchange detween devices
Slave-Loop - Data loop-back Rx-Tx. Used mainly for testing.

Example: Set EGBT-045MS in master role

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+ROLE=1↵
EGBT-045MS response
+ROLE:1↵
OK↵

9. Query/Set Class of Device CoD

COMMAND	RESPONSE
AT+CLASS?↵	+CLASS:<class>↵ OK↵
AT+CLASS=<class>↵	OK↵

where <class>
0 - (default)

The Class of Device identifier. For more info, see the Bluetooth_Code_Definition.pdf file included with the product documentation of this kit.

10. Query/Set Inquire Access Code

COMMAND	RESPONSE
AT+IAC?↵	+IAC:<iac>↵ OK↵
AT+IAC=<iac>	OK↵ / FAIL↵

where <iac> = Inquire Access Code
9e8b33 - default value

11. Query/Set Inquire Access Mode

COMMAND	RESPONSE
AT+INQM?↵	+INQM:<inq1>,<inq2>,<inq3>↵ OK↵
AT+INQM=<inq1>,<inq2>,<inq3>↵	OK↵ / FAIL↵

where <inq1> Inquire Access Mode
0 - standard
1 - rssi (default)

<inq2> Maximum number of devices response
0 to 32000
1 (default)

<inq3> Inquire timeout
1 to 48
48 (default)

Maximum number of devices response - EGBT-045MS will stop inquiring once the number of devices that responded reaches this value.
Inquire Timeout - Multiply this number by 1.28 to get the maximum time in seconds the EGBT-045MS will wait for a respond to an inquiry call.

Example: Set EGBT-045MS Inquire Access Mode at
Inquire access mode - 1 (rssi)
Number of devices - 3
Timeout - 10 (10*1.28 = 12.8 seconds)

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+INQM=1,3,10↵
EGBT-045MS response
OK↵

12. Query/Set Pairing Passkey

COMMAND	RESPONSE
AT+PSWD?↵	+PSWD:<password>↵ OK↵
AT+PWSD=<password>↵	OK↵

where <password> = Alphanumeric password 16 characters max.
1234 - (default)

Example: Set EGBT-045MS Password to “e-Gizmo”

From Host controller:
AT+PSWD=e-Gizmo↵
EGBT-045MS response
OK↵

13. Query/Set UART parameter

COMMAND	RESPONSE
AT+UART?↵	+UART:<baud>,<stop>,<parity>↵ OK↵
AT+UART=<baud>,<stop>,<parity>↵	OK↵

where <baud> =baud rate, any one of the following
4800
9600 (default)
19200
38400
57600
115200
234000
460800
921600
1382400

<stop> = number of stop bits
0 - 1 bit (default)
1 - 2 bits

<parity> = Parity bit
0 - None (default)
1 - Odd parity
2 - Even Parity

Example: Set EGBT-045MS UART parameter to 115200bps, 2 stop bits, even parity

From Host controller:
AT+UART=115200,1,2↵
EGBT-045MS response
OK↵

14. Query/Set Connection Mode

COMMAND	RESPONSE
AT+CMODE?↵	+CMODE:<mode>↵ OK↵
AT+CMODE=<mode>↵	OK↵

where <mode>
0 - Connect to a specified Bluetooth device only (default). See related command Command 15.
1 - Can connect with any other Bluetooth device.
2 - Test mode

15. Query/Set Binding Bluetooth Address

COMMAND	RESPONSE
AT+BIND?↵	+BIND:<addr>↵ OK↵
AT+BIND=<addr>↵	OK↵

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

Example: Bind with Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA,UAP,LAP format = 0002,72,0A3C7F

From Host controller:
AT+BIND=0002,72,0A3C7F↵
EGBT-045MS response
OK↵

16. Query/Set LED Output Polarity

COMMAND	RESPONSE
AT+POLAR?↵	+POLAR:<led1>,<led2>↵ OK↵
AT+POLAR=<led1>,<led2>↵	OK↵

where <led1> = LED1 (pin 31) Polarity
0 - LED1 output active low
1 - LED1 output active high (default)

<led2> = LED2 (pin 32) Polarity
0 - LED2 output active low
1 - LED2 output active high (default)

LED 1
Flashes once each seconds to indicate EGBT-045MS is in Command Mode. Flashes two times per second when EGBT-045MS is in data mode.

LED2
Turns ON when EGBT-045MS remote connection is successfuly opened.

17. Set/Reset a User I/O pin

COMMAND	RESPONSE
AT+PIO=<pn>,<value>↵	OK↵

where <pn> = port number. Available port are as follows
2 - PIO2
3 - PIO3
4 - PIO4
5 - PIO5
6 - PIO6
7 - PIO7
10 - PIO10

<value>
0 - Logic Low
1 - Logic High

Example: Set PIO2 to logic High

From Host controller:
AT+PIO=2,1↵
EGBT-045MS response
OK↵

18. Set/Reset multiple User I/O pin

COMMAND	RESPONSE
AT+MPIO=<iomap>↵	OK↵

where
<iomap> =12-bit I/O map presented in hexadecimal

x	PIO10	X	X	PIO7	PIO6	PIO5	PIO4	PIO3	PIO2	x	x

x - don't care/reserved

Example:
Set PIO2 and PIO6 to logic High, all others to logic 0

Bit pattern is 0000 0100 0100 = 44 hexadecimal

From Host controller:
AT+MPIO=44↵
EGBT-045MS response
OK↵

19. Query User I/O pin

COMMAND	RESPONSE
AT+MPIO?↵	+MPIO:<iomap>↵ OK↵

where
<iomap> =12-bit I/O map presented in hexadecimal

x	PIO10	X	X	PIO7	PIO6	PIO5	PIO4	PIO3	PIO2	x	x

x - reserved -used by system, may assume any values

Example:
Read PIO inputs

From Host controller:
AT+MPIO?↵
EGBT-045MS response
+MPIO:944↵
OK↵
Returned value in binary: 1001 0100 0100

In this example, the PIO are previously set in command 18 with PIO2 and PIO6 set. The returned value also shows reserved bits 11 and 8 set by the system.

20. Query/Set Scanning Parameters

COMMAND	RESPONSE
AT+IPSCAN?↵	+IPSCAN:<int>,<dur>,<pint>,<pdur>↵ OK↵
AT+IPSCAN=<int>,<dur>,<pint>,<pdur>↵	OK↵

where <int> = inquire scan time interval
1024 - default
<dur> = inquire scan time duration
512 - default
<pint> = page scan time interval
1024 - default
<pdur> = page scan time duration
512 - default

All parameters must be represented with decimal integer value.

21. Query/Set SNIFF Energy Savings Parameters

COMMAND	RESPONSE
AT+SNIFF?↵	+SNIFF:<tmax>,<tmin>,<retry>,<timeout>↵ OK↵
AT+SNIFF=<tmax>,<tmin>,<retry>,<timeout>↵	OK↵

where <tmax> = maximum time
0 - default
<tmin> = minimum time
0 - default
<retry> = retry time
0 - default
<timeout> = timeout
0 - default

All parameters must be represented with decimal integer value.

22. Query/Set Security & Encryption Modes

COMMAND	RESPONSE
AT+SENM?↵	+SENM:<mode>,<encrypt>↵ OK↵
AT+SENM=<mode>,<encrypt>↵	OK↵

where <mode> = Security Mode
0 - sec_mode_off (default)
1 - sec_mode1_non-secure
2 - sec_mode2-service
3 - sec_mode3_link
4 - sec_mode_unknown

<encrypt> = encryption mode
0 - hci_enc_mode_off (default)
1 - hci_enc_mode_pt_to_pt
2 - hci_enc_mode_pt_to_pt_and_bcast

23. Delete Authenticated Device from List

COMMAND	RESPONSE
AT+RMSAD=<addr>↵	OK↵

where <addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Example: Remove from Authenticated Device list a Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA,UAP,LAP format = 0002,72,0A3C7F

From Host controller:
AT+RMASD=0002,72,0A3C7F↵
EGBT-045MS response if deletion is successful
OK↵
EGBT-045MS response if remote device address is not in the list
ERROR(15)↵

Caution:

Entering
AT+RMASD↵
will delete ALL authenticated device from the list!

24. Find Device from Authenticated Device List

COMMAND	RESPONSE
AT+FSAD=<addr>↵	OK↵

where <addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Note: AT+FSAD returns a FAIL response if device is not in the authenticated list

25. Query Total Number of Device from Authenticated Device List

COMMAND	RESPONSE
AT+ADCN?↵	+ADCN:<total>↵ OK↵

where
<total> = total number of devices in the authenticated device list

26. Query Most Recently Used Authenticated Device

COMMAND	RESPONSE
AT+MRAD?↵	+MRAD:<addr>↵ OK↵

where <addr> = 48 bit bluetooth address
in NAP:UAP:LAP format

27. Query Current Status of the Device

COMMAND	RESPONSE
AT+STATE?↵	+STATE:<stat>↵ OK↵

where
<stat> = Current Status, any one of the following:

INITIALIZED
READY
PAIRABLE
PAIRED
INQUIRING
CONNECTING
CONNECTED
DISCONNECTED
UNKNOWN

28. Initialize SPP Profile

COMMAND	RESPONSE
AT+INIT↵	OK↵ / FAIL↵

29. Query Nearby Discoverable Devices

COMMAND	RESPONSE
AT+INQ↵	+INQ: <addr>,<class>,>rssi>↵ OK↵

where <addr> = 48 bit bluetooth address
in NAP:UAP:LAP format
<class> = Device Class
<rssi> = RSSI

This command will scan and report all nearby dicoverable Bluetooth devices. The same device may be reported more than once.

This command will work only if EGBT-045MS is set to work as a master device, i.e. AT+ROLE=1, and after AT+INIT is executed.

Example: Discover nearby devices

From Host controller: Set device role as master

AT+ROLE=1↵

EGBT-045MS response

OK↵

From Host controller: Initialize SPP

AT+INIT↵

EGBT-045MS response

OK↵

From Host controller: Set inquire mode as RSSI, look for 9 devices, and 48 as timeout

AT+INQM=1,9,48↵

EGBT-045MS response

OK↵

From Host controller: Start Device Discovery

AT+INQ↵

EGBT-045MS response (sample only, actual report will vary)

+INQ:101D:C0:2E7B54,5A0204,7FFF↵

+INQ:25:48:21AD1A,5A020C,7FFF↵

OK↵

In this example, EGBT-045MS found only two discoverable devices. It will quit searching after the timeout time specified by the AT+INQM command or if the number of discovered devices equals the number of specified devices.

30. Cancel Search for Discoverable Devices

COMMAND	RESPONSE
AT+INQC↵	OK↵

AT+INQ can be stopped at anytime by executing this command.

31. Device Pairing

COMMAND	RESPONSE
AT+PAIR=<addr>,<timeout>↵	OK↵ / FAIL↵

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

<timeout> = Timeout time in sec

32. Connect to a Remote Device

COMMAND	RESPONSE
AT+LINK=<addr>↵	OK↵ / FAIL↵

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

33. Disconnect from a Remote Device

COMMAND	RESPONSE
AT+DISC↵	+DISC:<results>↵

where

<results> = Disconnection results, any one of the following:

SUCCESS
LINK_LOSS
NO_SLC
TIMEOUT
ERROR

34. Enter Energy Saving mode

COMMAND	RESPONSE
AT+ENSNIFF=<addr>↵	OK ↵

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

35. Exit Energy Saving mode

COMMAND	RESPONSE
AT+EXSNIFF=<addr>↵	OK↵

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format