Kernel Masters

UART Class Notes & Lab Experiments

Embedded C & RTOS

Kernel Masters
1/24/2023

Contents

Communication Protocols and Interfacing with External Devices

1. Introduction

The modern day microcontrollers are enabled with several communication protocols to achieve the requirement of communication. These communication protocols can be wire line or wireless. And it is not new to have communication protocols on microcontrollers. The 8 bit microcontroller can communicate with personal computers using **serial communication interface (RS-232)**. In last three decades several wire line and wireless communication protocols are evolved for embedded systems.

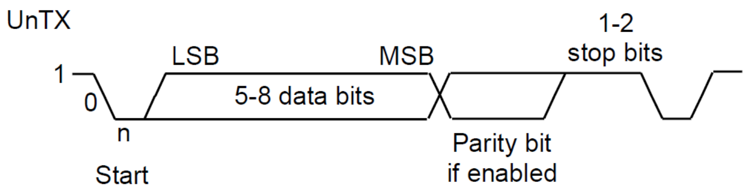
Communication between the microcontroller and other peripheral devices can be implemented in two ways: parallel and serial. In **parallel communication**, 8-bit or 16-bit bus is connected with external device with other appropriate control signal to establish meaningful communication. The simple example is interfacing memory with microcontroller. The bus (8 or 16 or 32 bit) connected directly with memory chip and ALE (Address Latch Enable) decides, whether bus carries address or data to write into memory and read from memory. Parallel communication was commonly used in the early days of microprocessor based systems.

As complexity in electronic systems arises, it became a challenge for the designer to optimize I/O‟s available with microcontroller which can connect with several other devices. It was not possible to connect microcontroller with other components parallel, because I/O‟s were not available. And also it increases the physical dimensions of PCB board which will directly increase the cost of production of electronic systems. In the mid 80‟s, Philips was facing similar problem with I/O‟s and cost of production, motivated Philips to invented a serial communication protocol: **Inter-IC protocol (I2C)**. To solve the same problem, Motorola came up with **Serial Peripheral Interface (SPI).** Several other serial communication protocols are developed in last two decades for different applications in automotive, industrial control and consumer electronics. Examples: **Controller Area Network (Automotive), Inter-IC sound (I2S)** etc. The parallel communication is easy to understand and implement. To optimize the I/O‟s and cost of production, most of the modern day electronic systems designs use different variety of serial communication protocols. Similarly, modern microcontrollers support wireless communication protocols like Bluetooth, ZigBee etc.

2. UART (Universal Asynchronous Receiver Transmitter) Controller

3. UART Basics

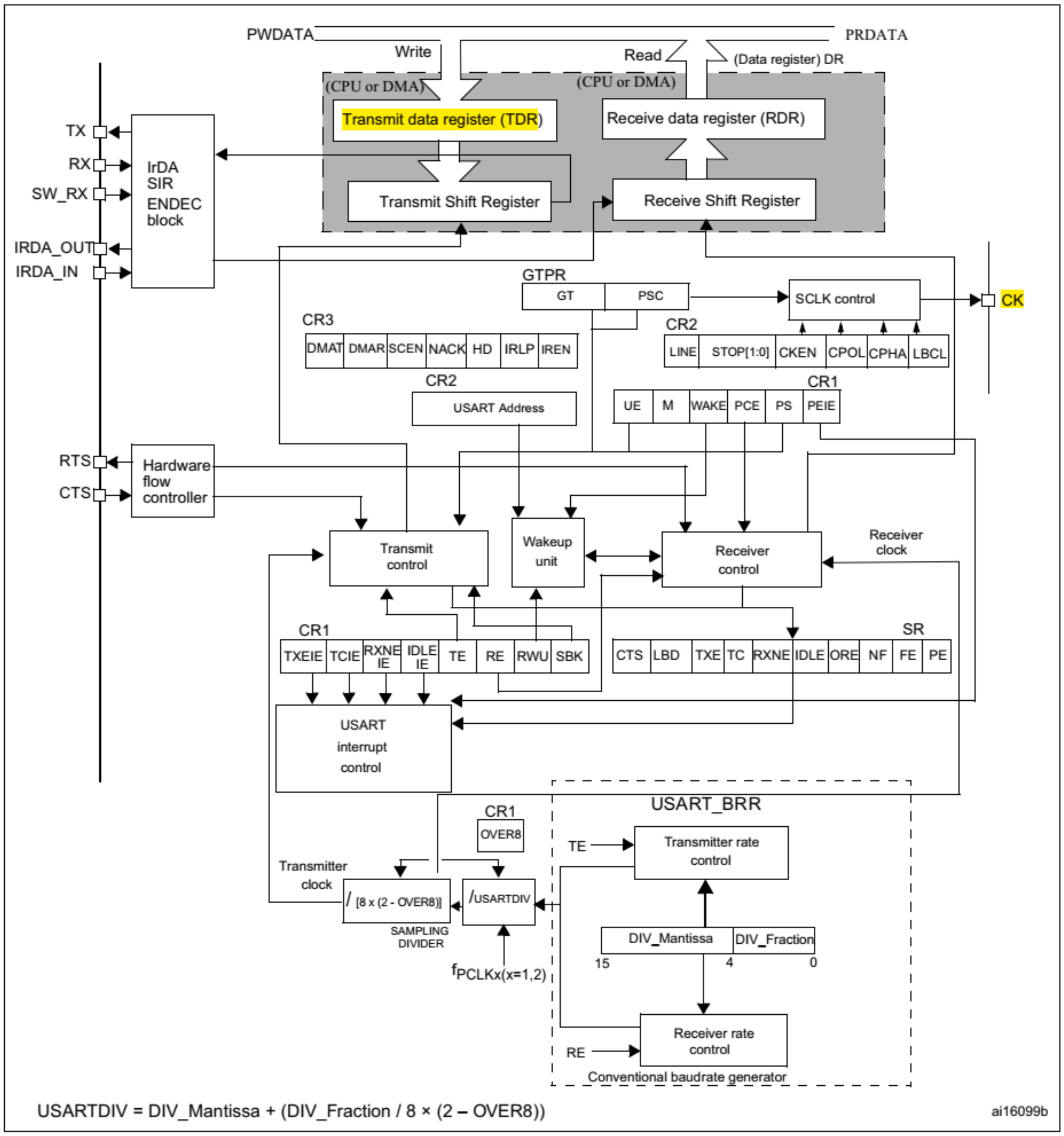
This serial port allows the microcontroller to communicate with devices such as other computers, printers, input sensors, and LCDs. Serial transmission involves sending one bit at a time, such that the data is spread out over time.
The total number of bits transmitted per second is called the **baud rate**. The reciprocal of the baud rate is the **bit time**, which is the time to send one bit.
Most microcontrollers have at least one UART.
Each UART will have a baud rate control register, which we use to select the transmission rate.
Each device is capable of creating its own serial clock with a transmission frequency approximately equal to the serial clock in the computer with which it is communicating.
A frame is the smallest complete unit of serial transmission.



4. STM32F401RBT6 USART Controller

- Full duplex, asynchronous communications.
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance
- Fractional baud rate generator systems
- Common programmable transmit and receive baud rate (refer to the datasheets for the value of the baud rate at the maximum APB frequency.
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
- 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission
- IrDA SIR encoder decoder Support for 3/16 bit duration for normal mode
- Smartcard emulation capability
- The Smartcard interface supports the asynchronous protocol Smartcards as defined in the ISO 7816-3 standards
- 0.5, 1.5 stop bits for Smartcard operation
- Single-wire half-duplex communication
- Configurable multibuffer communication using DMA (direct memory access)
- Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receive
- Transfer detection flags:
    - Receive buffer full
    - Transmit buffer empty
    - End of transmission flags
- Four error detection flags:
    - Overrun error
    - Noise detection
    - Frame error
    - Parity error

5. STM32F401RBT6 USART Functional Block Diagram



UART Lab Assignments:

1. UART Controller

Three USART Controllers Pin mapping details show the below table.

2.0/3.0			4.0/4.1		
GPIO Pin	**Pin Function**	**On-Board Function**	**GPIO Pin**	**Pin Function**	**On-Board Function**
PA9	GPIO	UART1_TX(Wi-Fi)	PA9	GPIO	UART1_TX
PA10	GPIO	UART1_RX(Wi-Fi)	PA10	GPIO	UART1_RX
PA2	GPIO	UART2_TX	PA2	GPIO	UART2_TX
PA3	GPIO	UART2_RX	PA3	GPIO	UART2_RX
PC6	GPIO	UART6_TX	PC6	GPIO	UART6_TX (Wi-Fi)
PC7	GPIO	UART6_RX	PC7	GPIO	UART6_RX (Wi-Fi)

1.1. UART With polling:

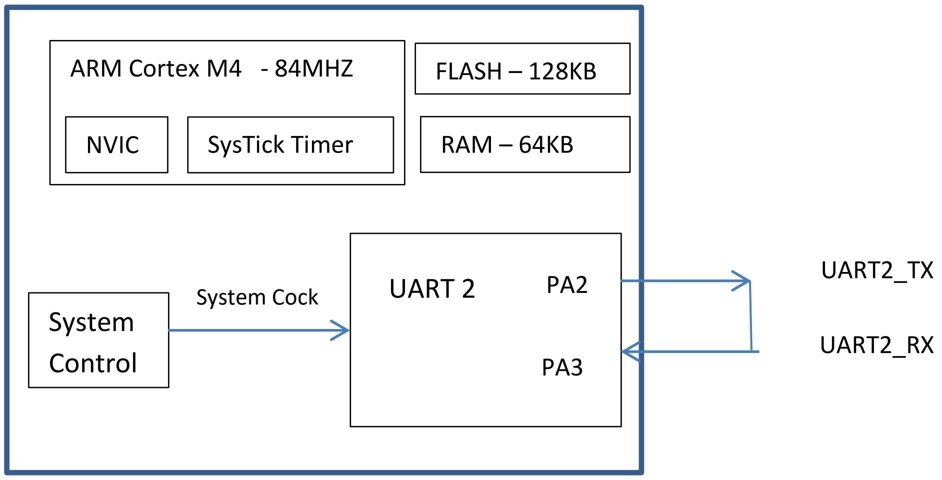
Experiment 1: USART Polling

Write a pseudo code and Embedded C program to **Configure UART2 with 9600 8N1.** And transmit character 'A' and Receive same character using hardware Loop back.

a. System Clock Initialization to 16MHZ.
  b. UART Communication protocol specifications –
    9600 8N1 = 1 start bit+8 word length +parity(0 bit)+ 1 stop bit = 10 bits.
  c. Enable polling (nothing but disable UART Interrupts)

Baud Rate = System clock/8*(2-0)*USARTDIV
USARTDIV = System clock/8*(2-0)*BaudRate = 16000000/16*9600 = 16000000/(16*9600)=104.1875 ==> 104+0.18==>68(integral part)

Step 1: Block Diagram



Step 2: Pseudo Code

UART Initialization: UART2_Init

- Enable Port A Clock
- Enable PA2 and PA3 as alternate functions (load ‘10’ in GPIO_PORTA_MODE Register)
- Select UART2_Tx and UART2_RX functionality using Mux register (Load ‘7’ in GPIO_PORTA_AFRH)
- Enable UART2 clock

UART Configuration: UART2_Config:

- Set 9600 baud rate in BRR (0x683)
- Set 13th, 2nd and 3rd bits in CR1 register to enable USART 2 enable

UART Operation:

```
Infinite loop:
        Transmit character:
        Loop:   Read USART2_SR register and check 7th bit (TXE) value, If TXE=0, go to loop
                otherwise next statement.
        Load ‘A’ in to USART2_DR register.
        void UART2_OutChar (unsigned char)
        {

        }

        Receive character:
        Loop:   Read USART2_SR register and check 5th bit (RXNE) value, If RXNE=0 (empty), go
                to loop otherwise next statement.
        Read USART2_DR registers and store in to local variable.
        unsigned char UART2_InChar (void)
        {

        }

Go to infinite loop
```

1.2. UART Blocking/Non-Blocking/Timeout Functions

Experiment 2: Write an Embedded C Program and implement USART Non-Blocking and USART TIMEOUT Functions.

USART Rx Functions

1. UART Rx Blocking function

unsigned char USART2_InChar (void);

UART2_InChar () function is called a Blocking function because this function continues in block state until data is available in RDR.
In this situation microprocessor doesn't handle any other task, so blocking function is not recommended.

Solution for this problem is to implement Non-Blocking and TIMEOUT function.

2. UART Rx Non-Blocking function

unsigned char USART2_InChar_NonBlock (void);
UART2_InChar_NonBlock () function is called a Non-Blocking function because this function fails if data is not available in RDR otherwise data will be stored in global variable.

3. UART Rx TIMEOUT function

int USART2_InChar_TIMEOUT (unsigned int);

This function waits for a pre-defined number of iterations mentioned in arguments. If data is not received before the completion of iterations, this function returns -1 to indicate TIMEOUT ERROR, otherwise data will be stored in global variable and return value is zero to indicate SUCCESS.

1.3. UART Rx Interrupts

Experiment 3: USART RX interrupts
Write a Pseudo code and Embedded C program to **USAR2 Configuration u**sing the below conditions.
a. System Clock Initialization to 16MHZ
b. 9600 8N1 (1+8+0+1=10 bits)
Assign a Hardware breakpoint in UART2_ISR and run program. Program execution time sends 'A' character from host system using Tera term application. Verify results in ISR.

1.4. Communication between two Raayan mini Board Using UART Protocol

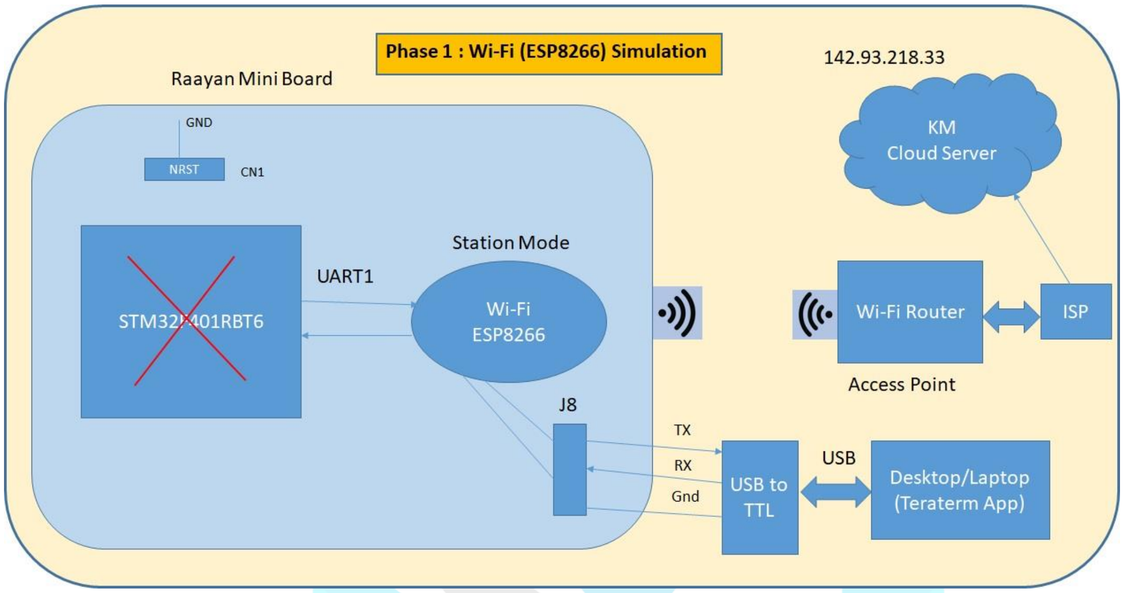
Experiment 4: Develop a Communication between two Raayan mini boads using UART2 protocol.
The switches (PC8 and PC9) are inputs, LEDs (PA8, PC5) are outputs, and the UART2 is used to communicate

Raayan Mini Board 1	**Raayan Mini Board 2/Laptop**
UP_SW ON & DN_SW OFF	RED LED ON (or) Check 'R' in Tera term window.
UP_SW OFF & DN_SW ON	GREEN LED ON (or) Check 'G' in Tera term window.
GREEN	UP_SW OFF & DN_SW ON (or) Type 'G' in Tera term window.
RED	UP_SW ON & DN_SW OFF (or) Type 'R' in Tera term window.

UART Project – Wi-Fi module interface with STM32F401RBT6:

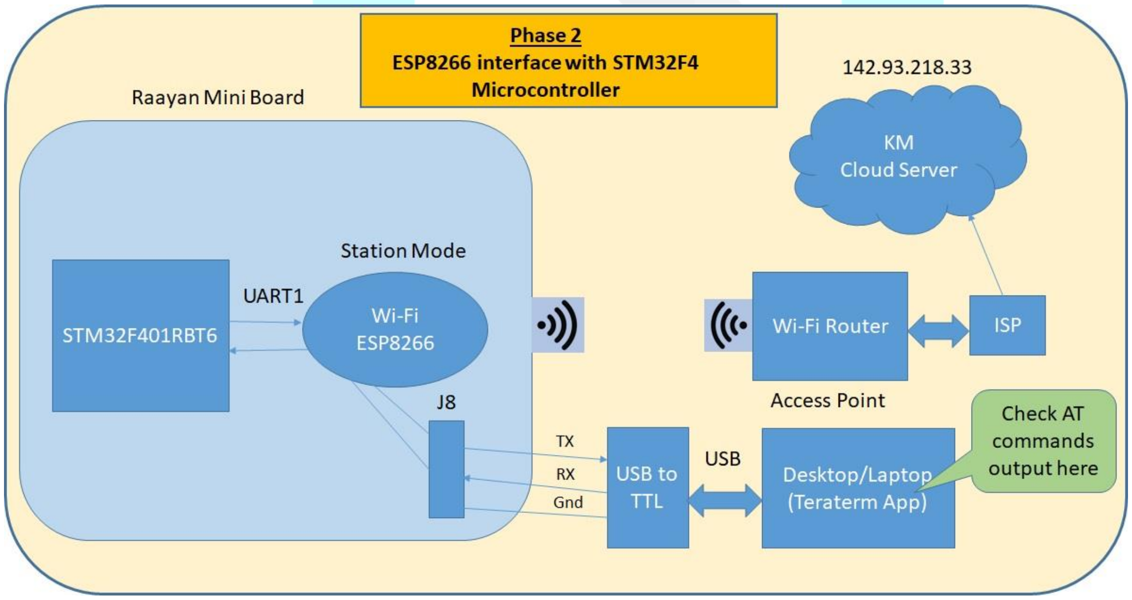
1. Phase 1: Disable STM32F401RBT6 microcontroller and connect Wi-Fi module with USB to TTL Converter.

Send Temperature, Humidity and Device ID to Kernel Masters server using AT Commands.
Run AT commands in “Docklight” serial communication protocol tool.



2. Phase 2: Enable STM32F401RBT6 microcontroller.

Write an embedded C program and send temperature, humidity and device ID to kernel masters server using UART1 communication protocol. Configure UART1 to 115200 8N1.



Kernel Masters Server IP Address: 142.93.218.33
Kernel Masters Webserver: http://142.93.218.33/km/weathermonitor.php

IoT Project - Smart Weather Monitoring System V1.0

Draw Block Diagram, pseudo code, Flowchart and write an Embedded C program and read temperature value from LM35 Temperature sensor every 5 sec delay and send to LCD and Kernel Masters webserver?

Hint: Write separate source file for every peripheral.
Example: SysTickTimer.c, lcd.c, adc.c , wifi.c , uart.c, main.c