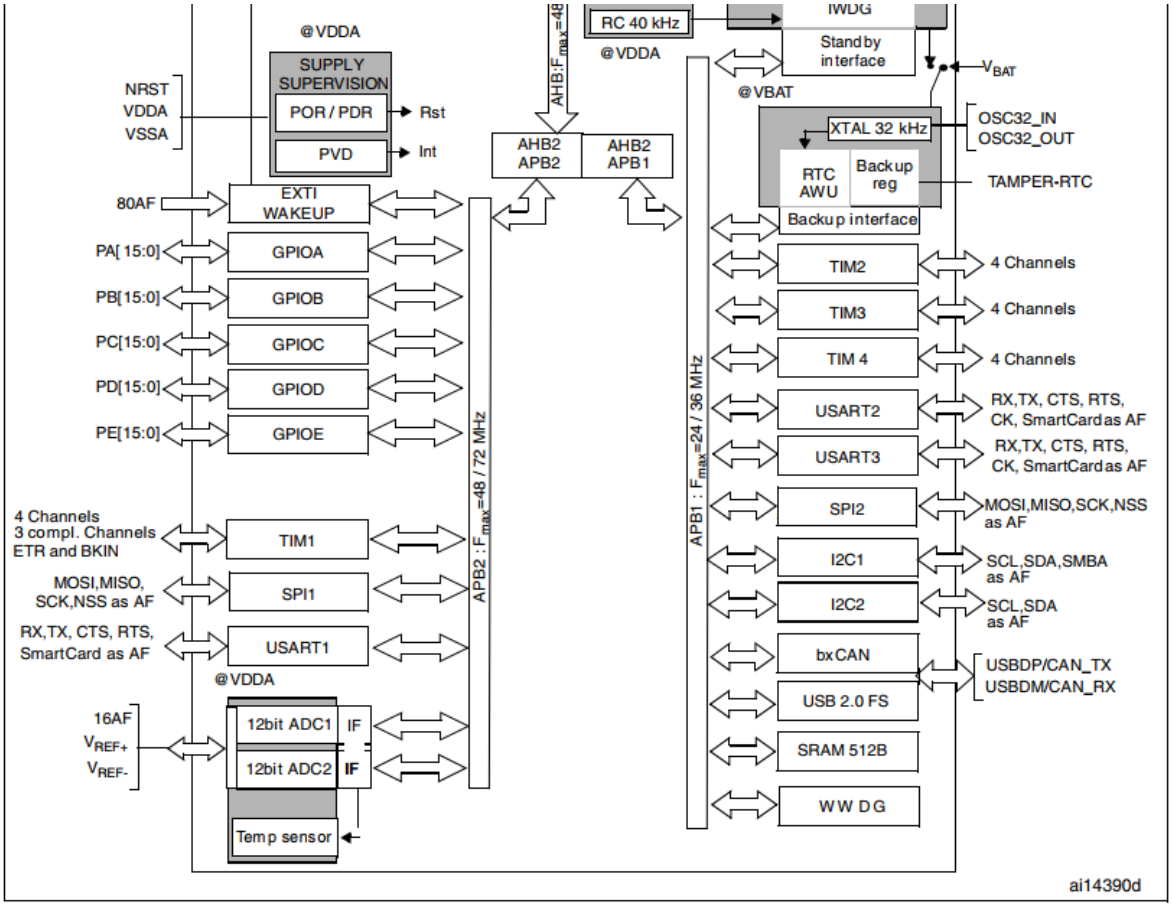
Bài 7: UART Software & UART Hardware

7.1 Phần Cứng USART



Bảng chân kết nối ra GPIO của USART1.

D9	D11	29	D7	41	67	20	PA8	I/O	FT	PA8	USART1_CK/ TIM1_CH1(9)/ MCO	-
C9	D10	30	C7	42	68	21	PA9	I/O	FT	PA9	USART1_TX(9)/ TIM1_CH2(9)	-
D10	C12	31	C6	43	69	22	PA10	I/O	FT	PA10	USART1_RX(9)/ TIM1_CH3(9)	-
C10	B12	32	C8	44	70	23	PA11	I/O	FT	PA11	USART1_CTS/ CANRX(9)/ USBDM/ TIM1_CH4(9)	-
B10	A12	33	B8	45	71	24	PA12	I/O	FT	PA12	USART1_RTS/ CANTX(9) /USBDP TIM1_ETR(9)	-

7.1.2 Các Thanh Ghi của USART

Table 198. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																							DR[8:0]								
	Reset value																								0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CP2102: USART To USB

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) trong các vi điều khiển thường bao gồm nhiều thanh ghi khác nhau để điều khiển hoạt động của module này. Dưới đây là sự khác nhau giữa các thanh ghi bạn đã đề cập:

1. **USART_SR (Status Register)**: Đây là thanh ghi trạng thái. Nó lưu trữ thông tin về trạng thái hiện tại của USART như các lỗi, cờ trạng thái gửi nhận, v.v. Một số cờ tiêu biểu trong thanh ghi này là:
    - TXE: Trạng thái bộ đệm truyền dữ liệu.
    - RXNE: Trạng thái bộ đệm nhận dữ liệu.
    - TC: Hoàn thành truyền dữ liệu.
2. **USART_DR (Data Register)**: Đây là thanh ghi dữ liệu. Dữ liệu sẽ được truyền đi hoặc nhận về thông qua thanh ghi này. Khi bạn gửi dữ liệu, bạn viết vào thanh ghi này. Khi bạn nhận dữ liệu, bạn đọc từ thanh ghi này.
3. **USART_BRR (Baud Rate Register)**: Thanh ghi này được sử dụng để thiết lập tốc độ baud (baud rate) của USART. Tốc độ baud xác định tốc độ truyền dữ liệu giữa vi điều khiển và thiết bị ngoại vi.
4. **USART_CR (Control Register)**: Đây là các thanh ghi điều khiển. Tùy thuộc vào vi điều khiển cụ thể, có thể có nhiều thanh ghi điều khiển như CR1, CR2, CR3, v.v. Các thanh ghi này chứa các bit điều khiển để bật/tắt USART, thiết lập chế độ truyền, điều chỉnh giao thức, v.v.
5. **USART_GTPR (Guard Time and Prescaler Register)**: Thanh ghi này được sử dụng để thiết lập thời gian bảo vệ và bộ chia tần số. Thời gian bảo vệ có thể hữu ích trong các ứng dụng cần thời gian bảo vệ giữa các byte dữ liệu.
6. **Làm thế nào để sử dụng các thanh ghi**

```c
#include "stm32f4xx.h"  // Thư viện tiêu chuẩn của STM32F4

void USART2_Init(void) {
    // 1. Bật clock cho USART2 và GPIOA
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    // 2. Cấu hình chân PA2 và PA3 cho USART2
    GPIOA->MODER |= (2 << (2 * 2)) | (2 << (3 * 2)); // Chọn chế độ alternate function
    GPIOA->AFR[0] |= (7 << (2 * 4)) | (7 << (3 * 4)); // Chọn alternate function cho PA2, PA3 là USART2

    // 3. Cấu hình Baud rate
    USART2->BRR = 0x0683; // 9600 baud rate với clock 16 MHz

    // 4. Bật USART và cấu hình các thông số truyền nhận
    USART2->CR1 |= USART_CR1_TE | USART_CR1_RE; // Bật truyền và nhận
    USART2->CR1 |= USART_CR1_UE; // Bật USART
}

void USART2_Write(int ch) {
    // Đợi cho đến khi TXE (Transmit Data Register Empty) được set
    while(!(USART2->SR & USART_SR_TXE)) {}
    USART2->DR = (ch & 0xFF);
```

```
30. }
31.
32. int USART2_Read(void) {
33.     // Đợi cho đến khi RXNE (Read Data Register Not Empty) được set
34.     while(!(USART2->SR & USART_SR_RXNE)) {}
35.     return USART2->DR & 0xFF;
36. }
37.
38. int main(void) {
39.     USART2_Init(); // Khởi tạo USART2
40.
41.     while(1) {
42.         USART2_Write('H'); // Gửi ký tự 'H'
43.         for(int i = 0; i < 1000000; i++) {} // Đợi một chút
44.     }
45. }
46.
```

7.1.3.Cách USART Truyền Thông

USART (Universal Synchronous and Asynchronous Receiver-Transmitter) truyền và nhận dữ liệu qua các bước sau:

1. **Truyền dữ liệu**:
    - Dữ liệu từ vi điều khiển được đưa vào bộ đệm truyền (transmit buffer) của USART.
    - Bộ đệm truyền chuyển đổi dữ liệu song song thành dữ liệu nối tiếp.
    - Dữ liệu nối tiếp được gửi ra ngoài qua chân truyền dữ liệu (Tx).
2. **Nhận dữ liệu**:
    - Dữ liệu nối tiếp từ thiết bị gửi được nhận qua chân nhận dữ liệu (Rx).
    - Bộ đệm nhận của USART chuyển đổi dữ liệu nối tiếp thành dữ liệu song song.
    - Dữ liệu song song sau đó được truyền đến vi điều khiển để xử lý.
3. **Đồng bộ hóa dữ liệu**:
    - Để xác định được đầu và cuối của một gói dữ liệu, USART sử dụng các bit start và stop.
    - Bit start (bit bắt đầu) báo hiệu sự bắt đầu của một gói dữ liệu.
    - Bit stop (bit kết thúc) báo hiệu sự kết thúc của gói dữ liệu.
4. **Điều khiển tốc độ**:
    - USART có thể hoạt động ở nhiều tốc độ baud khác nhau. chẳng hạn như 9600, 19200, 38400, 57600, 115200 baud
    - Tốc độ baud là số bit truyền trong một giây.
5. **Uart truyền dữ liệu nối tiếp, theo 1 trong 3 chế độ**:
    - Simplex: Chỉ tiến hành giao tiếp một chiều
    - Half duplex: Dữ liệu sẽ đi theo một hướng tại 1 thời điểm
    - Full duplex: Thực hiện giao tiếp đồng thời đến và đi từ mỗi master và slave
6. Chân Tx (truyền) của một chip sẽ kết nối trực tiếp với chân Rx (nhận) của chip khác và ngược lại. Quá trình truyền dữ liệu thường sẽ diễn ra ở 3.3V hoặc 5V. Uart là một

giao thức giao tiếp giữa một master và một slave. Trong đó 1 thiết bị được thiết lập để tiến hành giao tiếp với chỉ duy nhất 1 thiết bị khác.

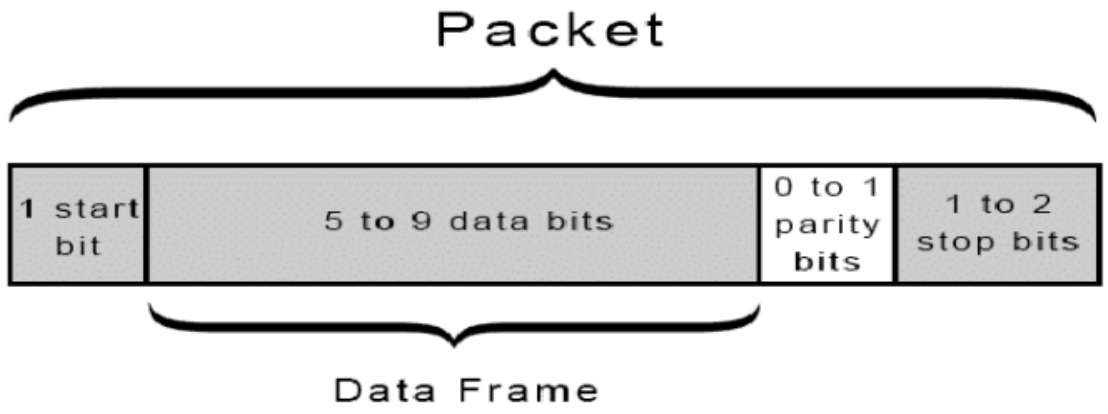
USART khởi tạo truyền dữ liệu và kết thúc truyền dữ liệu qua các bước sau:

1. Khởi tạo truyền dữ liệu:

- **Ở trạng thái Idle set đường tín hiệu luôn ở mức cao (hoặc thấp)**
- **Kích hoạt USART**: Trước khi bắt đầu truyền dữ liệu, bạn cần kích hoạt bộ USART và cấu hình các tham số như tốc độ baud, số bit dữ liệu, bit kiểm tra chẵn lẻ (parity bit), và số bit dừng.
- **Viết vào bộ đệm truyền (Transmit Buffer)**: Dữ liệu cần truyền được ghi vào bộ đệm truyền của USART.
- **Gửi bit start**: USART tự động thêm một bit start vào trước dữ liệu để báo hiệu bắt đầu truyền dữ liệu.
- **Gửi dữ liệu**: USART bắt đầu truyền từng bit dữ liệu, từ LSB (Least Significant Bit) đến MSB (Most Significant Bit).

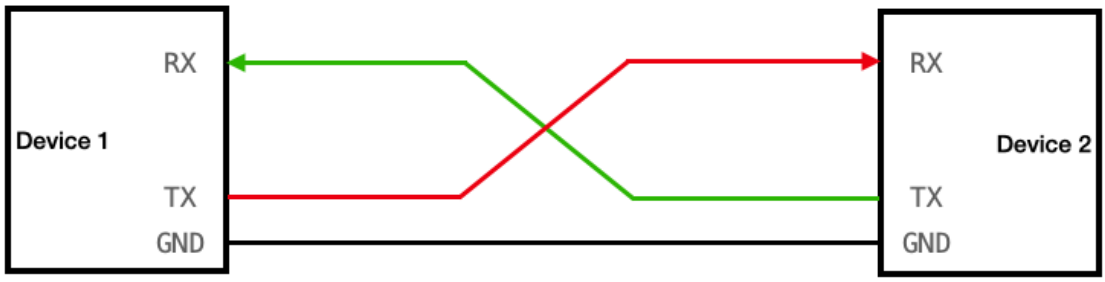
2. Kết thúc truyền dữ liệu:

- **Gửi bit stop**: Sau khi dữ liệu được truyền hoàn tất, USART tự động thêm một hoặc nhiều bit stop để báo hiệu kết thúc truyền dữ liệu.
- **Xác nhận dữ liệu truyền xong**: Sau khi bit stop được gửi đi, USART báo hiệu rằng quá trình truyền dữ liệu đã hoàn thành và sẵn sàng truyền dữ liệu tiếp theo.

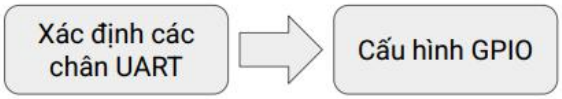


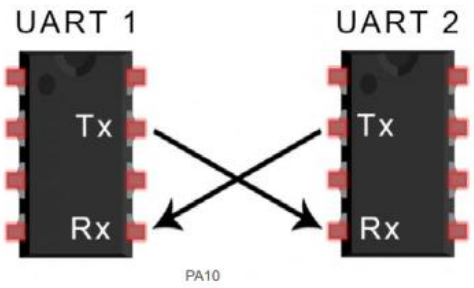
7.2.Cấu hình USART Software

UART chỉ sử dụng 2 chân để truyền, đó là TX và RX.

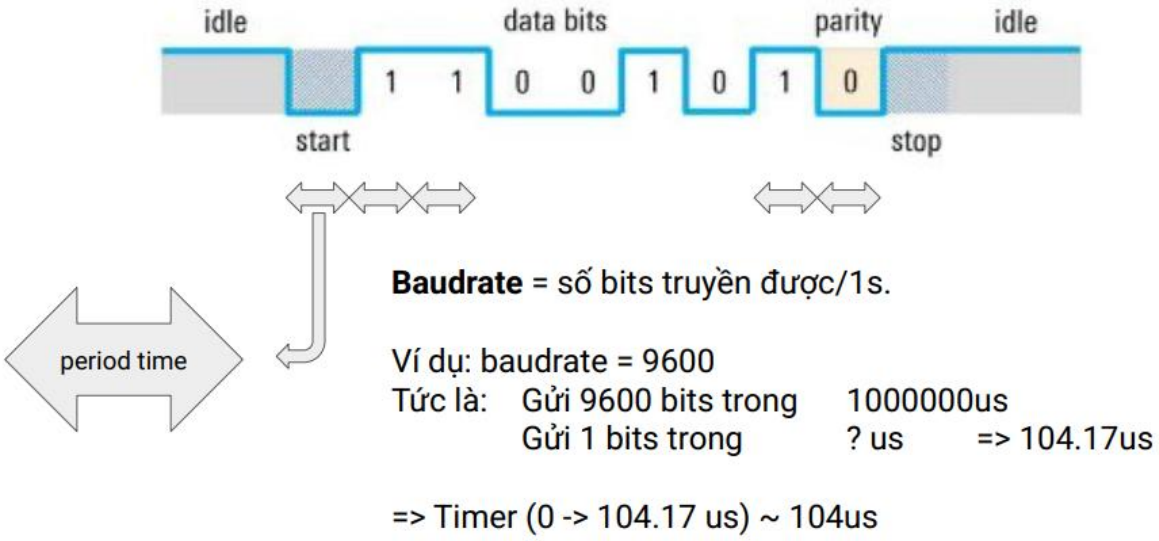


Xác định các chân sử dụng cho UART là bước đầu tiên. UART soft không yêu cầu các chân cụ thể nên ta có thể sử dụng chân bất kì:





7.2.1 Xác định thời gian truyền 1 bit => cấu hình timer



Tốc độ baudrate được xác định bởi thời gian truyền đi 1 bit. Ở bài này ta dùng tốc độ phổ thông 9600, ứng với mỗi bit là 105us => xây hàm delay 105us truyền 1 bit.

Timer hoạt động tần số 72MHz=> 1 xung clock = 1/72 us => chọn thang đếm Prescaler = 72-1 => cứ 1us thì timer đếm lên 1 lần.

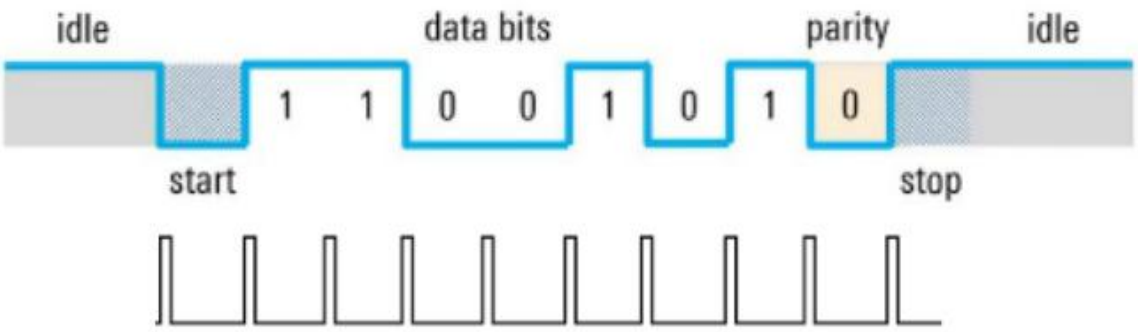
7.2.2 Chế độ nghỉ (Không truyền), đường TX sẽ được giữ ở mức cao. hàm UART_Config() thiết lập chế độ nghỉ cho đường truyền:

```
void UART_Config(){
	GPIO_SetBits(UART_GPIO, TX_Pin);
	delay_us(1);
}
```

7.2.3 Hàm truyền:

Hàm truyền sẽ truyền lần lượt 8 bit trong byte dữ liệu, sau khi tín hiệu start được gửi đi.

- Tạo start, delay 1 period time.
  - Truyền bit dữ liệu. mỗi bi truyền trong 1 period time.
  - Dịch 1 bit.
- Tạo stop, delay tương ứng với số bit stop




```
void UART_Transmit(const char DataValue)
{
	// Send Start Bit
	GPIO_WriteBit(UART_GPIO, RX_Pin, Bit_RESET);
	delay_us(BRateTime);

	for ( unsigned char i = 0; i < 8; i++ ){
```

```
                if( ((DataValue>>i)&0x1) == 0x1 ){
                        GPIO_WriteBit(UART_GPIO, RX_Pin, Bit_SET);
                } else{
                        GPIO_WriteBit(UART_GPIO, RX_Pin, Bit_RESET);
                }
        delay_us(BRateTime);
        }
        // Send Stop Bit
        GPIO_WriteBit(UART_GPIO, RX_Pin, Bit_SET);
        delay_us(BRateTime);
}
```

Data truyền đi sẽ được thêm bit parity tùy theo cấu hình parity bit là chẵn/lẻ hay không dùng pairty bit:

```
typedef enum{
        Parity_Mode_NONE,
        Parity_Mode_ODD,
        Parity_Mode_EVENT
}Parity_Mode;
```

Có thể tạo bit parity bằng cách đếm số bit 1, sau đó thêm vào cuối chuỗi bit bit 0 hoặc 1 tương ứng:

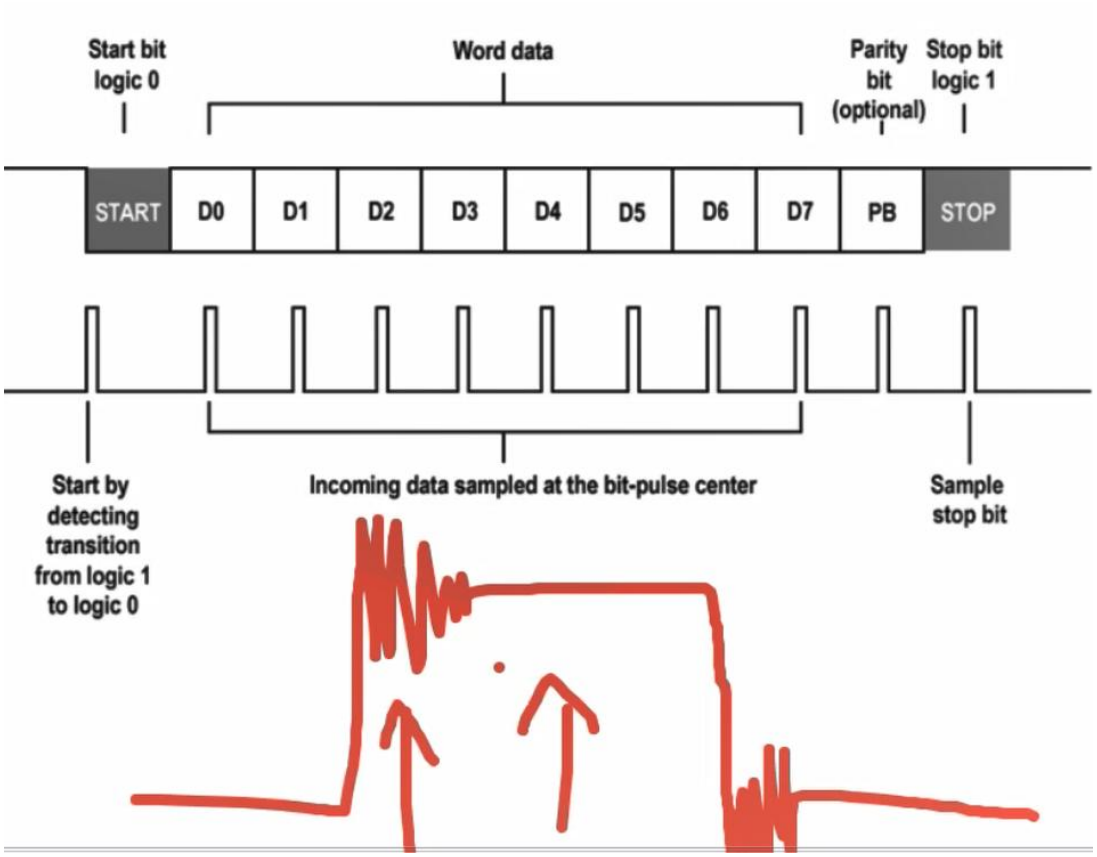
```
uint8_t Parity_Generate(uint8_t data, Parity_Mode Mode){
        uint8_t count =0;
        for(int i=0; i< 8; i++){
                if(data & 0x01){
                        count++;
                }
                data>>=1;
        }
        switch(Mode){
                case Parity_Mode_NONE:
                        return data;
                        break;
```

```
                case Parity_Mode_ODD:
                        if(count%2){
                                return ((data<<1)|1);
                        } else {
                                return (data<<1);
                        }
                        break;
                case Parity_Mode_EVENT:
                        if(!(count%2)){
                                return ((data<<1)|1);
                        } else {
                                return (data<<1);
                        }
                        break;
                default:
                        return data;
                        break;
        }
}
```

7.2.4.Hàm nhận:

Hàm nhận sẽ nhận lần lượt nhân 8 bit.

- Nếu baud=9600 chẳng hạn, thì thời gian của bit data = 104 us. Ta không cần thiết phải đọc dữ liệu ngay khi có tín hiệu đến. Do sườn lên và sườn xuống của xung data thường nhiễu. bởi vậy ta sẽ đọc ở vị chí chính giữa xung data.
- Chờ tín hiệu start từ thiết bị gửi (start bit: TX từ mức cao => mức thấp =0)
- Delay 1,5 period time (để đọc chính giữa bít xung data)
    - Đọc data trên RX, ghi vào biến.
    - Dịch 1 bit.
    - Delay 1 period time.
- Delay 0,5 period time và đợi stop bit (là bit keo từ mức 0->1: duy trì trong 1 thời gian dài).




```
unsigned char UART_Receive(void)
{
	unsigned char DataValue = 0;
	while(GPIO_ReadInputDataBit(UART_GPIO, RX_Pin) == 1);

	delay_us(BRateTime);
	delay_us(BRateTime/2);
	for ( unsigned char i = 0; i < 8; i++ )
	{
		if ( GPIO_ReadInputDataBit(UART_GPIO, RX_Pin) == 1 ){
			DataValue += (1<<i);
		}
		delay_us(BRateTime);
	}
		if ( GPIO_ReadInputDataBit(UART_GPIO, RX_Pin) == 1 ){
			delay_us(BRateTime/2);
			return DataValue;
		}
```

}

Sau khi nhận được data, có thể tiến hành kiểm tra chẵn/lẻ:

```
uint8_t Parity_Check(uint8_t data, Parity_Mode Mode){
	uint8_t count =0;
	for(int i=0; i< 8; i++){
		if(data & 0x01){
			count++;
		}
		data>>=1;
	}
	switch(Mode){
		case Parity_Mode_NONE:
			return 1;
			break;
		case Parity_Mode_ODD:
			return (count%2);
			break;
		case Parity_Mode_EVENT:
			return (!(count%2));
			break;
		default:
			return 0;
			break;
	}
}
```

```
#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"
#define TX_Pin GPIO_Pin_9
#define RX_Pin GPIO_Pin_10
#define UART_GPIO GPIOA  // set cổng ra USART Là GPIOA
#define BRateTime 104 // baudrate (9600 baud) = 9600 bit/s => delay truyen 1
bit = 104.16 us (làm tròn =104 us)
void RCC_Config() {
    //Cấp clock cho GPIOA, TIM2
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
}
void GPIO_Config() {
    //Cau hinh chan TX, RX cho GPIO,
    //RX nhận tín hiệu nên để mode: Input-floating, mode này không có điện trở kéo lên hoặc kéo xuống.
    //TX Truyền dữ liêu, chọn mode: Out_PP, Có điện trở kéo lên mức cao để tín hiệu ổn định
GPIO_InitTypeDef GPIOInitStruct;
GPIOInitStruct.GPIO_Pin = RX_Pin;
GPIOInitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(UART_GPIO, &GPIOInitStruct);
GPIOInitStruct.GPIO_Pin = TX_Pin;
GPIOInitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIOInitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(UART_GPIO, &GPIOInitStruct); //Nạp cấu hình cho biến struct mới khai báo
}
void TIM_Config() {
    //Cau hinh Timer
TIM_TimeBaseInitTypeDef TIM_InitStruct;
TIM_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_InitStruct.TIM_Prescaler = 72 - 1; // Cấu hình 1 us : Timer đếm lên 1 lần.
TIM_InitStruct.TIM_Period = 0xFFFF;
TIM_InitStruct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_InitStruct);
TIM_Cmd(TIM2, ENABLE);
}
void delay_us(uint32_t time) {
    // Hàm delay thời gian micro second
TIM_SetCounter(TIM2, 0);
while (TIM_GetCounter(TIM2) < time) {}
}
void clock() {  //Ham tao xung chu kỳ = 104 us (tương đương 9600 baud)
delay_us(BRateTime); // = delay_us(104) = 104 us truyen 1 bit.
}
void UARTSoftware_Init() { // Hàm set chế độ idle cho chân TX, lúc rảnh TX ở mức cao
GPIO_SetBits(GPIOA, TX_Pin);// Set TX_pin = 1.
delay_us(1);
}
void UARTSoftware_Transmitt(char c) {
// Ham truyen data (char c)
GPIO_ResetBits(GPIOA, TX_Pin); // Keo chan TX xuong 0 cho bit bat dau, bat dau truyen data.
clock(); // delay 104us
```

```
// Truyen 8 bit du lieu (tu LSB sang MSB)
for (int i = 0; i < 8; i++) {
    // Dùng mặt nạ Mask (1<<i):bx0000 00i0: vi trí thứ i, bit set lên 1. bit 1
lần lượt dịch từ bit LSB->MSB
    // Hàm điều kiên if(condition) = Nếu giá trị ở mức cao (1) thí set TX_pin
lên 1, ngược lại reset TX_pin về 0
    if (c & (1 << i)) {
GPIO_SetBits(GPIOA, TX_Pin); // Gui bit 1
} else {
GPIO_ResetBits(GPIOA, TX_Pin); // Gui bit 0
}
clock(); // Delay 104 us (9600 baud)
}
// stop bit- bit dung
GPIO_SetBits(GPIOA, TX_Pin); // Kéo chân TX lên mức cao =1 : stop bit
clock(); // Đợi thời gian 104 us để 1 thành bit stop.
}
char UARTSoftware_Receive() {
    // Ham nhan data
char c = 0;

    while (GPIO_ReadInputDataBit(GPIOA, RX_Pin) == 1); // wait Cho den khi
RX_pin = 0, thoat vong lap. (start bit =0)
// Cho 1.5 baudrate de lay mau du lieu chinh giua bit data, de tranh noise o
suon len va suon xuong cua bit
delay_us(BRateTime + BRateTime / 2);
// Doc 8 bit du lieu (LSB truoc)
for (int i = 0; i < 8; i++) {
if (GPIO_ReadInputDataBit(GPIOA, RX_Pin)) {
    c |= (1 << i); // mask (1<<i) : cho phep ghi bit thu i cua bien c len 1.
}
clock(); // Doi theo thoi gian cho bit tiep theo
}
// Cho bit dung (nno phai cao)
delay_us(BRateTime / 2); // Doi thoi gian 0.5 baudrate sau khi doc du lieu
return c;
}
char data[] = {'H','A','I'};
int main() {
RCC_Config();
GPIO_Config();
TIM_Config();
UARTSoftware_Init(); // set trang thai idle cho chan TX
while (1) {
for (int i = 0; i < 3; i++) {
UARTSoftware_Transmitt(data[i]);
delay_us(1000);
}
```

```
char received = UARTSoftware_Receive();
UARTSoftware_Transmitt(received);
}
}
```

7.3. Lập Trình trên UART Hardware

7.3.1.Cấu hình GPIO

I/O	FT	PA9	USART1_TX(9)/ TIM1_CH2(9)
I/O	FT	PA10	USART1_RX(9)/ TIM1_CH3(9)

Các bộ UART trong STM32F1 được xác định sẵn các chân GPIO

Tương tự Software, TX sẽ là OUTPUT và RX sẽ là INPUT.

```
void GPIO_Config() {
    GPIO_InitTypeDef GPIO_InitStruct;
    // PA9 TX, mode: Alternative Function_Push Pull (chế độ cho usart,
spi,i2c)
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
    // PA10 RX, mode : thả nổi điện áp để nhận xung data
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
    }
```

1. **Cấu hình UART**

Tương tự các ngoại vi khác, các tham số Uart được cấu hình trong Struct USART_InitTypeDef:

- USART_Mode: Cấu hình chế độ hoạt động cho UART:
  - USART_Mode_Rx: Cấu hình truyền.
  - USART_Mode_Tx: Cấu hình nhận.
  - Có thể cấu hình cả 2 cùng lúc.
- USART_BaudRate: Cấu hình tốc độ baudrate cho uart.
- USART_HardwareFlowControl: Cấu hình chế độ bắt tay cho uart.
- USART_WordLength: Cấu hình số bit mỗi lần truyền.
- USART_StopBits: Cấu hình số lượng stopbits.
- USART_Parity: cấu hình bit kiểm tra chẵn, lẻ.
-

```
void UART_Config() {
    USART_InitTypeDef UART_InitStruct;
    UART_InitStruct.USART_BaudRate = 9600;
    UART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    UART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; // Cau hinh
USART thuc hien ca truyen nhan
    UART_InitStruct.USART_Parity = USART_Parity_No;
    UART_InitStruct.USART_StopBits = USART_StopBits_1;
    UART_InitStruct.USART_WordLength = USART_WordLength_8b;
    USART_Init(USART1, &UART_InitStruct); // Luu cau hinh vao bien
    USART_Cmd(USART1, ENABLE); // Lenh cho USART1 Hoat dong
```

USART_InitTypeDef

```
typedef struct
{
  uint32_t USART_BaudRate;            /*!< This member configures the USART communication baud rate.
                                           The baud rate is computed using the following formula:
                                            - IntegerDivider = ((PCLKx) / (16 * (USART_InitStruct->USART_BaudRate)))
                                            - FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5 */

  uint16_t USART_WordLength;          /*!< Specifies the number of data bits transmitted or received in a frame.
                                           This parameter can be a value of @ref USART_Word_Length */

  uint16_t USART_StopBits;            /*!< Specifies the number of stop bits transmitted.
                                           This parameter can be a value of @ref USART_Stop_Bits */

  uint16_t USART_Parity;              /*!< Specifies the parity mode.
                                           This parameter can be a value of @ref USART_Parity
                                           @note When parity is enabled, the computed parity is inserted
                                                 at the MSB position of the transmitted data (9th bit when
                                                 the word length is set to 9 data bits; 8th bit when the
                                                 word length is set to 8 data bits). */

  uint16_t USART_Mode;                /*!< Specifies wether the Receive or Transmit mode is enabled or disabled.
                                           This parameter can be a value of @ref USART_Mode */

  uint16_t USART_HardwareFlowControl; /*!< Specifies wether the hardware flow control mode is enabled
                                           or disabled.
                                           This parameter can be a value of @ref USART_Hardware_Flow_Control */
} USART_InitTypeDef;
```

```
/** @defgroup USART_Word_Length
  * @{
  */

#define USART_WordLength_8b                  ((uint16_t)0x0000)
#define USART_WordLength_9b                  ((uint16_t)0x1000)

#define IS_USART_WORD_LENGTH(LENGTH) (((LENGTH) == USART_WordLength_8b) || \
                                      ((LENGTH) == USART_WordLength_9b))
```

```
/** @defgroup USART_Stop_Bits
  * @{
  */

#define USART_StopBits_1                     ((uint16_t)0x0000)
#define USART_StopBits_0_5                   ((uint16_t)0x1000)
#define USART_StopBits_2                     ((uint16_t)0x2000)
#define USART_StopBits_1_5                   ((uint16_t)0x3000)
#define IS_USART_STOPBITS(STOPBITS) (((STOPBITS) == USART_StopBits_1) || \
                                     ((STOPBITS) == USART_StopBits_0_5) || \
                                     ((STOPBITS) == USART_StopBits_2) || \
                                     ((STOPBITS) == USART_StopBits_1_5))
```

```
/** @defgroup USART_Hardware_Flow_Control
  * @{
  */
#define USART_HardwareFlowControl_None       ((uint16_t)0x0000)
#define USART_HardwareFlowControl_RTS        ((uint16_t)0x0100)
#define USART_HardwareFlowControl_CTS        ((uint16_t)0x0200)
#define USART_HardwareFlowControl_RTS_CTS    ((uint16_t)0x0300)
#define IS_USART_HARDWARE_FLOW_CONTROL(CONTROL)\
                              (((CONTROL) == USART_HardwareFlowControl_None) || \
                               ((CONTROL) == USART_HardwareFlowControl_RTS) || \
                               ((CONTROL) == USART_HardwareFlowControl_CTS) || \
                               ((CONTROL) == USART_HardwareFlowControl_RTS_CTS))
/** @defgroup USART_Parity
  * @{
  */

#define USART_Parity_No                       ((uint16_t)0x0000)
#define USART_Parity_Even                     ((uint16_t)0x0400)
#define USART_Parity_Odd                      ((uint16_t)0x0600)
#define IS_USART_PARITY(PARITY)  (((PARITY) == USART_Parity_No) || \
                                  ((PARITY) == USART_Parity_Even) || \
                                  ((PARITY) == USART_Parity_Odd))
/** @defgroup USART_Mode
  * @{
  */

#define USART_Mode_Rx                         ((uint16_t)0x0004)
#define USART_Mode_Tx                         ((uint16_t)0x0008)
#define IS_USART_MODE(MODE) ((((MODE) & (uint16_t)0xFFF3) == 0x00) && ((MODE) != (uint16_t)0x00))
```

Hàm **USART_SendData(USART_TypeDef* USARTx, uint16_t Data)**, truyền data từ UARTx. Data này đã được thêm bit chẵn/lẻ tùy cấu hình.

Hàm **USART_ReceiveData(USART_TypeDef* USARTx)**, nhận data từ UARTx.

Hàm **USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG)** trả về trạng thái cờ USART_FLAG tương ứng:

- **USART_FLAG_TXE**: Cờ truyền, set lên 1 nếu quá trình truyền hoàn tất.
- **USART_FLAG_RXNE**: Cờ nhận, set lên 1 nếu quá trình nhận hoàn tất.
- **USART_FLAG_IDLE**: Cờ báo đường truyền đang ở chế độ Idle.
- **USART_FLAG_PE**: Cờ báo lỗi Parity.

Quá trình truyền/nhận có thể mô tả như sau:

- Bắt đầu truyền/nhận, UART xóa hết data trong thanh ghi DR để đảm bảo data đúng.
- Truyền: Gửi đi từng byte data. Sau đó đợi cờ TXE bật lên.
- Nhận: Đọc data từ bộ UART, chờ cờ RNXE bật lên.
- Đối với mảng dữ liệu, lặp lại quá trình cho từng byte.

```
void UART_SendDataArray(USART_TypeDef* USARTx, uint8_t* data, uint8_t size)
{ // Creat ham truyen 1 mang data
    for (int i = 0; i < size; i++) {
    USART_SendData(USARTx, data[i]);
    while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET) {} // wait
den khi co bao truyen thanh cong Flag_TXE = 1 thi thoat vong lap
    delay_us(1000);
    }
    }
    uint8_t UART_ReceiveByte(USART_TypeDef* USARTx) { // creat ham nhan
byte data
    while (USART_GetFlagStatus(USARTx, USART_FLAG_RXNE) == RESET) {}// wait
den khi co bao nhan thanh cong FLAG_RXNE =1, TH
    return USART_ReceiveData(USARTx);
    }
```

Hàm truyền bit data thực chất là ghi vào thanh ghi DR, nhận bit Data là đọc từ thanh ghi DR ra

```
591  void USART_SendData(USART_TypeDef* USARTx, uint16_t Data)
592  {
593    /* Check the parameters */
594    assert_param(IS_USART_ALL_PERIPH(USARTx));
595    assert_param(IS_USART_DATA(Data));
596
597    /* Transmit Data */
598    USARTx->DR = (Data & (uint16_t)0x01FF);
599  }
600
601  /**
602    * @brief  Returns the most recent received data by the USARTx peripheral.
603    * @param  USARTx: Select the USART or the UART peripheral.
604    *   This parameter can be one of the following values:
605    *   USART1, USART2, USART3, UART4 or UART5.
606    * @retval The received data.
607    */
608  uint16_t USART_ReceiveData(USART_TypeDef* USARTx)
609  {
610    /* Check the parameters */
611    assert_param(IS_USART_ALL_PERIPH(USARTx));
612
613    /* Receive Data */
614    return (uint16_t)(USARTx->DR & (uint16_t)0x01FF);
615  }
616
```