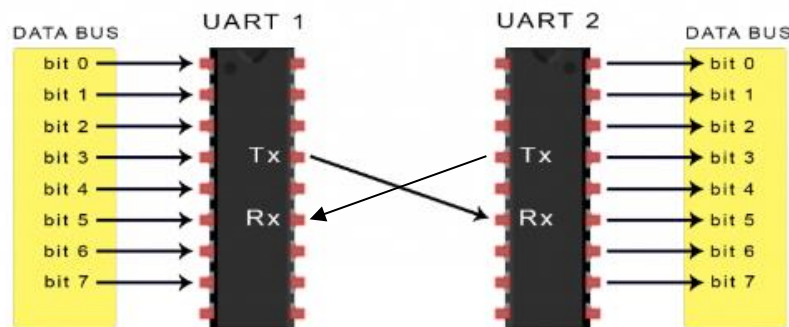
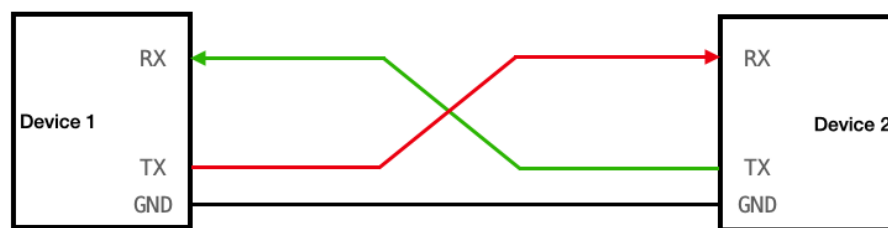


Chapter 5: Universal Synchronous Asynchronous Receiver Transmitter (USART)

5.1 Introduction to UART:

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



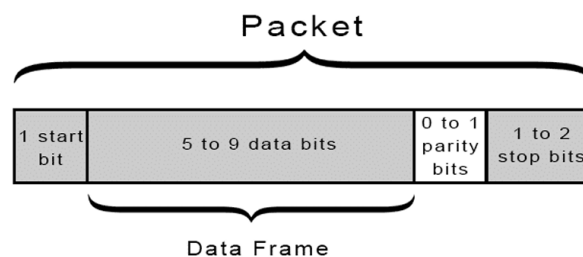
UARTs transmit data **asynchronously**, which means there is **no clock** signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred.

Wires Used	Two
Maximum Speed	Refer to the exact max Baud rate for the specific Baud Rate
Serial or Parallel	Serial
Max # of Slaves	1
Max # of Masters	1

- The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller.

- Data is transferred from the data bus to the transmitting UART in parallel form.
- After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet.
- Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin.
- The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end.

UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits.



The UART data transmission line is normally held at a **high voltage** level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high too **low for one clock cycle**. When the receiving UART detects the **high to low voltage transition**, it begins reading the bits in the data frame at the frequency of the baud rate.

Start Bit:

START BIT The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting **UART pulls the transmission line from high to low for one clock cycle**. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

Data Frame:

The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. **If no parity bit is used, the data frame can be 9 bits long.** In most cases, the data is sent with the least significant bit first.

PARITY Parity:

Why parity bit is required?

PARITY Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

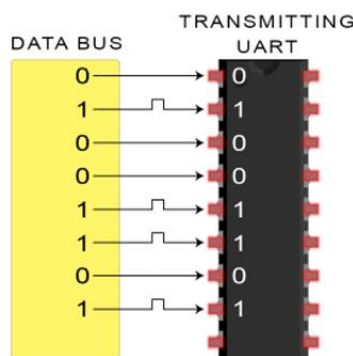
How it works?

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the **parity bit is a 0** (even parity), the 1 bit in the data frame should total to an **even number**. If the **parity bit is a 1** (odd parity), the 1 bit in the data frame should total to an **odd number**.

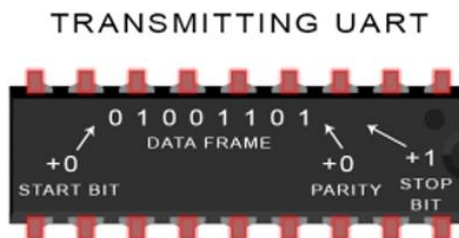
When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

5.1.1 Steps for UART:

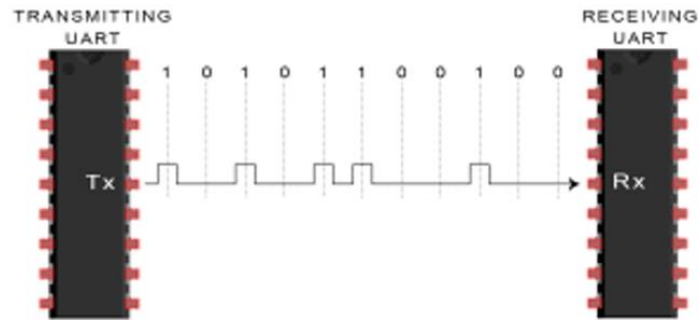
- 1) The transmitting UART receives data in parallel from the data bus.



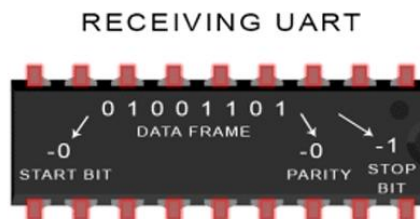
- 2) The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:



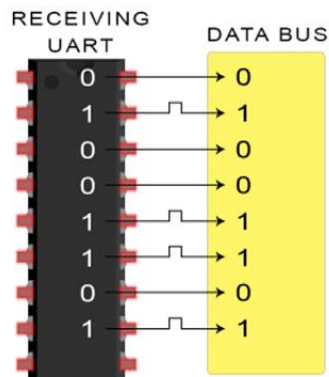
- 3) The entire packet is sent serially from the transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate.

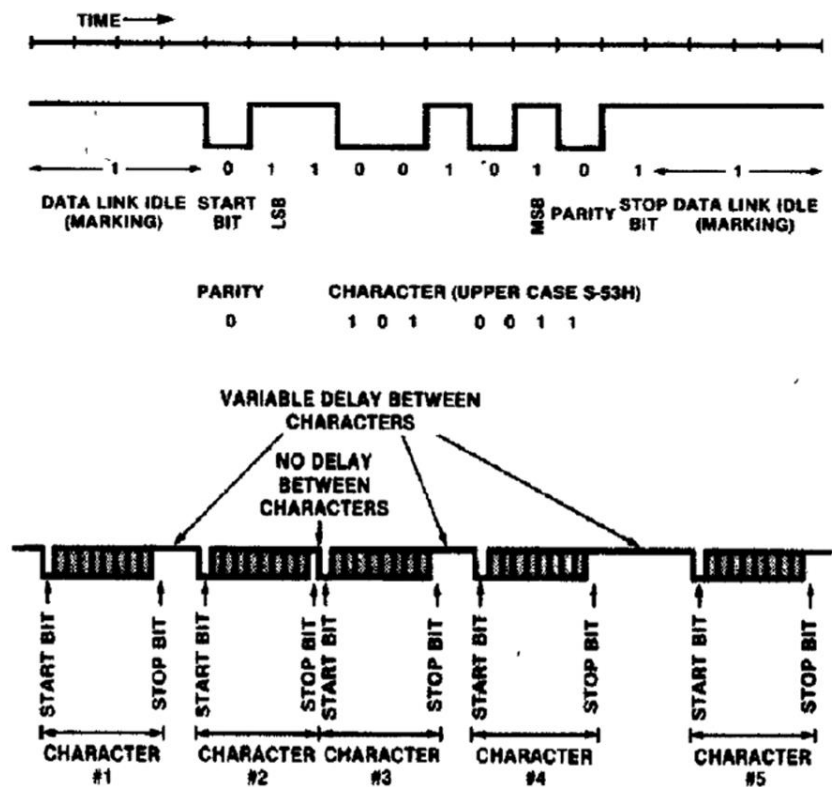


- 4) The receiving UART discards the start bit, parity bit, and stop bit from the data frame.



- 5) The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end.





5.1.2 Advantages and Disadvantages of UART:

Advantages	Disadvantages
<ul style="list-style-type: none"> Only uses two wires 	The size of the data frame is limited to a maximum of 9 bits
<ul style="list-style-type: none"> No clock signal is necessary 	Doesn't support multiple slave or multiple master systems
<ul style="list-style-type: none"> Has a parity bit to allow for error checking 	The baud rates of each UART must be within 10% of each other
<ul style="list-style-type: none"> The structure of the data packet can be changed as long as both sides are set up for it. 	
<ul style="list-style-type: none"> Well documented and widely used method 	

5.2 UART with CH32V003 MCU:

The module contains **one** Universal Synchronous Asynchronous Transceiver USART1.

5.2.1 Main Features

- Full-duplex or half-duplex synchronous or asynchronous communication
- NRZ data format
- Fractional baud rate generator, up to **3Mbps**
- Programmable data length
- Configurable stop bits
- Support LIN, IrDA encoders, smart cards (dnt knw what this is)
- DMA support
- Multiple interrupt sources

Let's first go to the datasheet of Ch32003 to observe the pin configuration of the MCU.

CH32V003F4P6

1	PD4/A7/UCK/T2CH1ETR/OPO/T1CH4ETR_	PD3/A4/T2CH2/AETR/UCTS/T1CH4_	20
2	PD5/A5/UTX/T2CH4_/URX_	PD2/A3/T1CH1/T2CH3_/T1CH2N_	19
3	PD6/A6/URX/T2CH3_/UTX_	PD1/SWIO/AETR2/T1CH3N/SCL_/URX_	18
4	PD7/NRST/T2CH4/OPP1/UCK_	PC7/MISO/T1CH2_/T2CH2_/URTS_	17
5	PA1/OSCI/A1/T1CH2/OPN0	PC6/MOSI/T1CH1CH3N_/UCTS_/SDA_	16
6	PA2/OSCO/A0/T1CH2N/OPP0/AETR2_	PC5/SCK/T1ETR/T2CH1ETR_/SCL_/UCK_/T1CH3_	15
7	VSS	PC4/A2/T1CH4/MCO/T1CH1CH2N_	14
8	PD0/T1CH1N/OPN1/SDA_/UTX_	PC3/T1CH3/T1CH1N_/UCTS_	13
9	VDD	PC2/SCL/URTS/T1BKIN/AETR_/T2CH2_/T1ETR_	12
10	PC0/T2CH3/UTX_/NSS_/T1CH3_	PC1/SDA/NSS/T2CH4_/T2CH1ETR_/T1BKIN_/URX_	11

Here we can see pin **PD5** and **PD6** (on CH32v003 MCU pins are already named as Tx and Rx) have default roles of **UTx** and **URx**. Alternate functions can be used to configure the other pins for UART. But here we will use default functions.

As you know we used WCH-LINK as the programmer/debugger module to download the code on MCU. Here we can use it for the serial communication purpose.

5.3 USART Example:

***This routine demonstrates that USART1 receives the data sent by CH341 and inverts it and sends it (baud rate 115200).**

***Hardware connection: (PD5 – Rx) & (PD6 – Tx)**

```
#include "debug.h"

/* Global define */

/* Global Variable */
vu8 val;

/*****
 * @fn      USARTx_CFG
 *
 * @brief   Initializes the USART2 & USART3 peripheral.
 *
 * @return  none
 */
void USARTx_CFG(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure = {0};
    USART_InitTypeDef USART_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_USART1,
ENABLE);

    /* USART1 TX-->D.5   RX-->D.6 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = 115200;
    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_Tx | USART_Mode_Rx;

    USART_Init(USART1, &USART_InitStructure);
    USART_Cmd(USART1, ENABLE);
}

/*****
 * @fn      main
 */
```

```

*
* @brief   Main program.
*
* @return  none
*/
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    SystemCoreClockUpdate();
    Delay_Init();
#ifdef SDI_PRINT == SDI_PR_OPEN
    SDI_Printf_Enable();
#else
    USART_Printf_Init(115200);
#endif
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    USARTx_CFG();

    while(1)
    {

        while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET)
        {
            /* waiting for receiving finish */
        }
        val = (USART_ReceiveData(USART1));
        USART_SendData(USART1, ~val);
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
        {
            /* waiting for sending finish */
        }
    }
}

```