第3章 S800的嵌入式实验

- **3.1 实验一 时钟选择与 GPIO 实验**

  (ref. B-chapter5, 10, C-chapter26,14)

- **3.2 实验二 I2C扩展及SYSTICK中断实验**

  (ref. B-chapter3,18, C-chapter16,17,28)

- **3.3 实验三 UART串行通讯口实验**

  (ref. B-chapter16, C-chapter30)

本章节参考资料:

A. 自编讲义《嵌入式系统实验教程》

B. Tiva™ TM4C1294NCPDT Microcontroller Data Sheet

C. TivaWare™ Peripheral Driver Library User's Guide

D. S800板介绍V0.65

实验三 UART串行通讯口实验

- 实验目的
  - 了解 UART 串行通讯的工作原理
  - 掌握在 PC 端通过串口调试工具与实验板进行 UART 通讯的方法
  - 掌握 UART 的堵塞式与非堵塞式数据传输编程方法
  - 理解和掌握中断优先级的设置及对程序运行的影响

实验三 UART串行通讯口实验

- 预备知识
  - 3.3.1 通用异步收发器 UART （ref. B-chapter16, C-chapter30）
  - 3.3.2 UART常用电路接口
  - 3.3.3 UART调试工具的使用

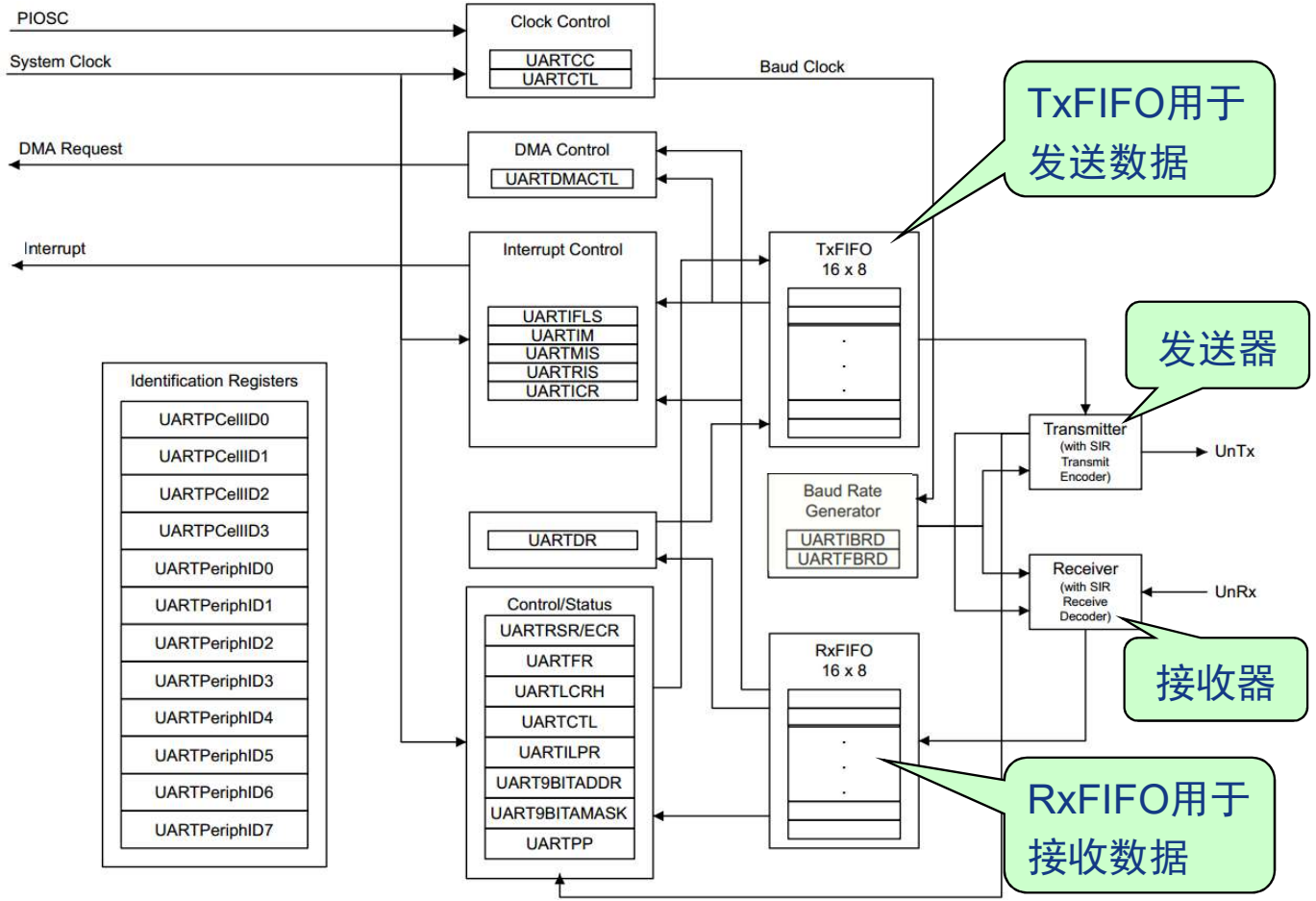
3.3.1 通用异步收发器 UART

- 计算机与外部设备连接的两类接口：
  - 并行接口是指数据的各个位同时进行传送，传输速度快，但当传输距离远、位数多时，通信线路变复杂且成本提高
  - 串行接口是指数据一位一位地顺序传送，通信线路简单，在远距离通信时可以大大降低通信成本
- 串行通信分为两类：
  - 异步串行通信ASYNC（Asynchronous Data Communication）
  - 同步串行通信SYNC（Synchronous Data Communication）
- UART（Universal Asynchronous Receiver/Transmitter，通用异步收发器）是设备间进行异步串行通信的关键模块

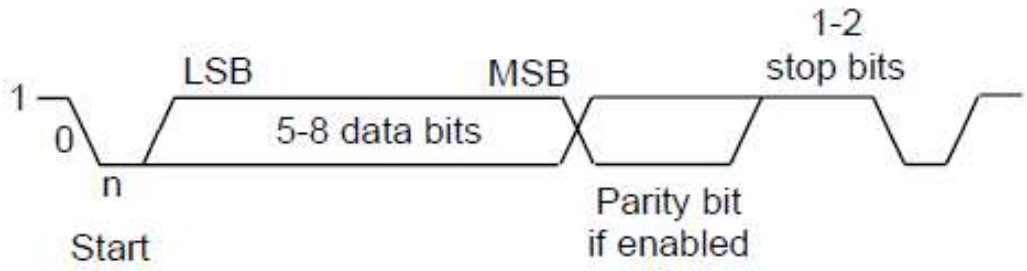
TM4C1294NCPDT的UART模块

- 可编程的波特率发生器，在常规模式（16 分频）下最高可达7.5Mbps，在高速模式（8分频）下最高可达15Mbps
- 相互独立的发送队列和接收队列
  - 发送队列TxFIFO：16×8位；接收队列RxFIFO：16×8位
- 标准的异步通讯位：起始位、停止位、奇偶校验位
- 完全可编程的串行接口特性
  - 5、6、7 或8 个数据位
  - 偶校验位、奇校验位、粘附(Stick)校验位 或 无校验位
  - 1 或2 个停止位

UART模块功能框图


PIOSC
System Clock
DMA Request
Interrupt
Clock Control
UARTCC
UARTCTL
Baud Clock
DMA Control
UARTDMACTL
Interrupt Control
UARTIFLS
UARTIM
UARTMIS
UARTRIS
UARTICR
UARTDR
Control/Status
UARTRSR/ECR
UARTFR
UARTLCRH
UARTCTL
UARTILPR
UART9BITADDR
UART9BITAMASK
UARTPP
Identification Registers
UARTPCellID0
UARTPCellID1
UARTPCellID2
UARTPCellID3
UARTPeriphID0
UARTPeriphID1
UARTPeriphID2
UARTPeriphID3
UARTPeriphID4
UARTPeriphID5
UARTPeriphID6
UARTPeriphID7
TxFIFO
16 x 8
RxFIFO
16 x 8
Baud Rate
Generator
UARTIBRD
UARTFBRD
Transmitter
(with SIR
Transmit
Encoder)
UnTx
Receiver
(with SIR
Receive
Decoder)
UnRx
TxFIFO用于
发送数据
发送器
接收器
RxFIFO用于
接收数据

- **UART功能描述**
- UART字符帧格式



- 发送器/接收器
  - 发送器对TxFIFO中的数据执行“并→串”转换，按字符帧格式输出串行比特流
  - 接收器在检测到一个有效的起始脉冲后，对接收到的比特流执行“串→并”转换和错误检测，并写入RxFIFO

- **UART功能描述**
- 波特率产生
  - 波特率分频系数BRD由16位整数 BRDI（**UARTIBRD**寄存器值）和6位小数 BRDF（**UARTFBRD**寄存器值）构成，即
  - BRD = BRDI.BRDF = UARTSysClk / (ClkDiv * Baud Rate)

    其中：
    - UARTSysClk 为系统时钟（UART模块的时钟直接来自系统时钟）
    - ClkDiv 为分频系数（取16 [常规模式] 或 8 [高速模式]）
    - Baud Rate为波特率（取9600，38400，115200等）

- **UART数据通讯原理**
- UART通过UnTx引脚发送数据，通过UnRx引脚接收数据。可以实现全双工收发
- 两种数据通讯方式
  - 阻塞式通讯：执行到发送或接收函数时，始终查询状态，直到发送或接收结束才退出。(没有空位或数据时等待)
  - 非阻塞式通讯：执行到发送或接收函数时，发送函数仅仅将数据推送给寄存器，并不保证传送成功，如果能传送则返回TRUE，否则返回FALSE；接收函数检查接收状态后即返回，如果没有收到数据则返回FALSE，如果收到则返回TRUE

- **带FIFO的UART数据通讯原理**
- 发送FIFO的基本操作
  - 只要有数据写到TxFIFO，就会立即启动发送。由于发送本身是个相对缓慢的过程，因此在发送的同时其它待发送的数据还可以继续填充到TxFIFO里，直到TxFIFO满
  - 按照“先进先出”的原则将TxFIFO中的数据一个个发送出去，直到TxFIFO全空为止。（已发送出去的数据会被自动清除，留出空位）
- 接收FIFO的基本操作
  - 当硬件逻辑接收到数据，就会往RxFIFO里填充接收到的数据，直到RxFIFO满
  - 程序应当及时取走这些数据，取走的数据会被自动从RxFIFO中清除，留出空位

UART数据通讯中断处理

- UART通信时，中断方式比轮询方式要简便且效率高
- 发送数据时，当TxFIFO里剩余的数据减到预设的深度时触发发送中断
- 接收数据时，当RxFIFO里累积的数据达到预设的深度时触发接收中断
- 收发FIFO的触发深度可配置为：1/8、1/4、1/2、3/4或7/8深度。
- 复位时，两个FIFO的中断触发深度均默认配置为1/2

※ 注意：在使能接收中断的同时一般还要使能接收超时中断，否则在接收FIFO未填充到预设深度而对方已经发送完毕的情况下，接收到的数据就得不到及时处理

- **UART的中断条件**
  - FIFO溢出错误
  - 线路中止错误
  - 奇偶校验错误
  - 帧错误
  - 接收超时（适用于RxFIFO已有数据但未达预设深度，而后续数据长时间不来的情况）
  - 数据发送
  - 数据接收

※ 注意：一个UART模块只有一个中断号，上述中断事件共用同一个UART一个中断请求。可以通过查询中断状态，在中断服务函数里处理多个中断事件

UART 中断参数（driverlib/uart.h）

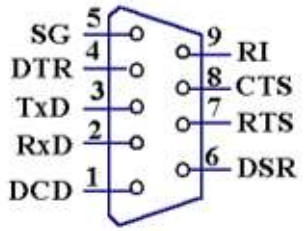
```
// Values that can be passed to UARTIntEnable, UARTIntDisable, UARTIntClear,
// and returned from UARTIntStatus.

……
#define UART_INT_OE             0x400       // Overrun Error Interrupt
#define UART_INT_BE             0x200       // Break Error Interrupt
#define UART_INT_PE             0x100       // Parity Error Interrupt
#define UART_INT_FE             0x080       // Framing Error Interrupt
#define UART_INT_RT             0x040       // Receive Timeout Interrupt
#define UART_INT_TX             0x020       // Transmit Interrupt
#define UART_INT_RX             0x010       // Receive Interrupt
……
```

3.3.2 UART常用电路接口

- UART只定义了串行通讯的逻辑层，即字符传送的格式与速率，但对于利用UART进行传输的物理电平接口需要另外定义，因此根据接口电平的不同与控制方式的不同，产生了 **RS-232** 与 **RS-485** 接口标准

UART常用电路接口说明

- **RS-232**
  - RS-232是 PC 机与设备通信里应用最广泛的一种串行接口。它被定义为一种在低速率串行通讯中增加通讯距离的单端标准。由于其最大通信距离的限制，因此它常用于本地设备之间的通信
  - RS232标准定义逻辑“1”信号相对于地为-3V~ -15V，而逻辑“0”相对于地为+3V~+15V。所以，当一个微控制器中的UART相连于PC时，它需要一个RS232驱动器来转换电平
  - 最高传输率20Kb/s
  - 最大传输距离约为15m
  - 单端传输（点对点），共模抑制比低



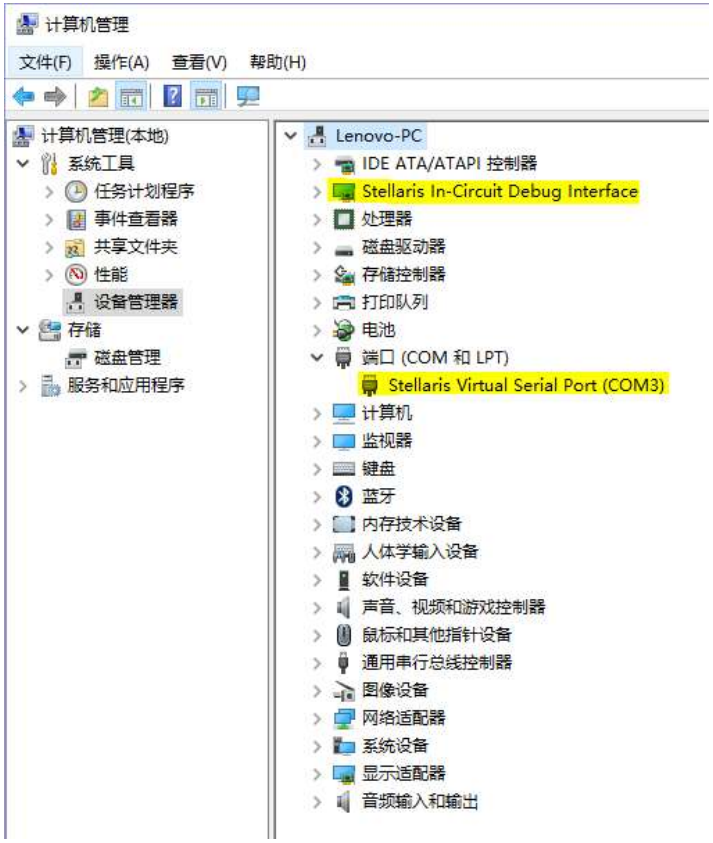
DB-9

UART常用电路接口说明

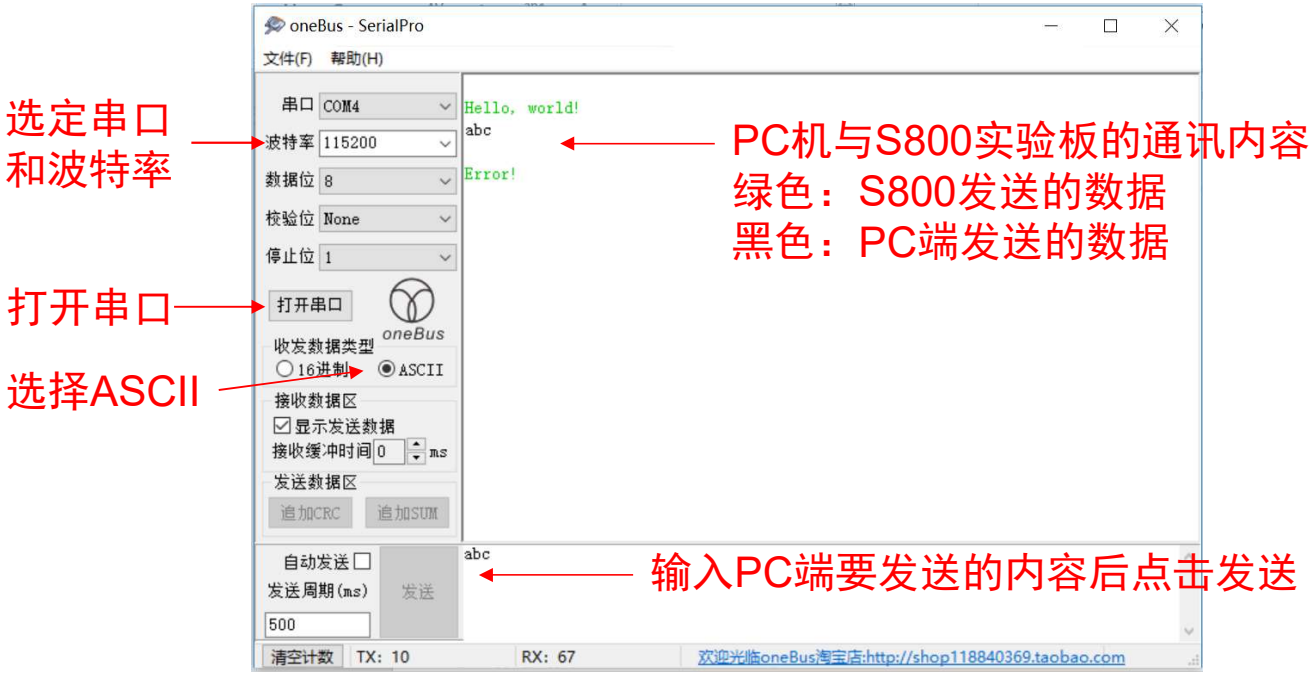
- **RS-485**
  - RS-485是一种常用于远距离和多机通信的串行接口。半双工工作方式，易于多点互连或联网成分布式系统
  - RS-485只是定义电压和阻抗
  - 与TTL电平兼容
  - 最高传输速率10Mb/s
  - 传输距离可达1200m
  - 双端传输，共模抑制比高
  - 需要 RS485 驱动器来将单端信号转换为双端信号

3.3.3 UART调试工具的使用

- **S800实验板的 UART 通讯口**
- 安装好驱动后，S800实验板通过MICRO-USB线与PC机连接，正常情况下打开电脑的设备管理器，会至少看到以下两个设备：
  - **ICDI在线调试工具**
  - **虚拟串口**（COM口）
- 这个虚拟串口可以用来作为MCU的UART通讯或是作为Bootload使用



- S800实验板与PC机端的串口调试工具
  - 任意选择一款PC端串行口通讯软件，在正确配置了控制板的UART口后即可实现PC与控制板的通讯
  - 推荐串口软件SerialProV1.04.exe



实验三 UART串行通讯口实验

- 实验内容
  - 例程exp3-1.c：对实验板UART0 进行初始化，在初始化完成后向主机发送“HELLO,WORLD！”。Task1完成UART ECHO，即通过PC端串口调试窗口向实验板发送字符串，实验板收到后原样返回；Task2为跑马灯控制。
  - 编程要点
    1. **UART初始化配置**
    2. UART数据发送
    3. UART数据接收

- UART模块初始化配置步骤（**S800_UART_Init()**）

1. 使能UART模块
2. 使能提供UART接口信号的GPIO端口

通过系统外设控制函数设置

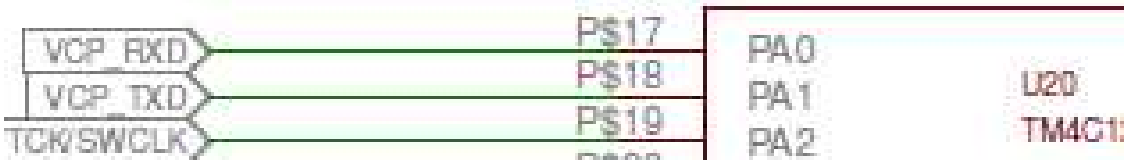
3. 将GPIO引脚设置为UART复用功能 —— 通过GPIO控制函数设置
4. 配置并使能UART通讯（包括通讯格式和FIFO深度等）

—— 通过UART控制函数设置

- UART接口信号
  - UART接口信号由GPIO引脚的复用功能提供
    - 默认跳线时，UART0模块的数据信号与GPIO A口 的PA0、PA1相连接

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
U0Rx	33	PA0 (1)	I	TTL	UART module 0 receive.
U0Tx	34	PA1 (1)	O	TTL	UART module 0 transmit.

    - **PA0：UART0_RX（用于虚拟串口VCP_RXD）**
    - **PA1：UART0_TX（用于虚拟串口VCP_TXD）**



- TivaWare的UART库函数
  - 驱动程序在driverlib/uart.c，API定义在driverlib/uart.h
- UART 的配置和控制函数：
  - **UARTConfigSetExpClk()**：配置UART模块的时钟速率、波特率和数据格式，并使能UART和 FIFO（复位时UART和FIFO是除能的）

```
void UARTConfigSetExpClk (uint32_t ui32Base,      //UART端口地址
                          uint32_t ui32UARTClk, //时钟频率
                          uint32_t ui32Baud,      //波特率
                          uint32_t ui32Config);     //数据格式
```

其中数据格式为数据位数、停止位数、校验位的“或”

例：UARTConfigSetExpClk(UART0_BASE, ui32SysClock, 115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE)); //115200-8-N-1

UART 配置参数（driverlib/uart.h）

```
// Values that can be passed to UARTConfigSetExpClk
// and returned by UARTConfigGetExpClk
#define UART_CONFIG_WLEN_8      0x00000060  // 8 bit data
#define UART_CONFIG_WLEN_7      0x00000040  // 7 bit data
#define UART_CONFIG_WLEN_6      0x00000020  // 6 bit data
#define UART_CONFIG_WLEN_5      0x00000000  // 5 bit data
#define UART_CONFIG_STOP_ONE    0x00000000  // One stop bit
#define UART_CONFIG_STOP_TWO    0x00000008  // Two stop bits
#define UART_CONFIG_PAR_NONE    0x00000000  // No parity
#define UART_CONFIG_PAR_EVEN    0x00000006  // Even parity
#define UART_CONFIG_PAR_ODD     0x00000002  // Odd parity
……
```

- UART收发FIFO深度设置函数：
  - **UARTFIFOLevelSet()**：设置触发中断的UART收发FIFO深度级别，默认都为1/2深度

```
void UARTFIFOLevelSet ( uint32_t ui32Base,        //UART端口地址
                        uint32_t ui32TxLevel,     //TxFIFO触发深度
                        uint32_t ui32RxLevel);    //RxFIFO触发深度
```

其中ui32TxLevel取值：UART_FIFO_TX1_8、UART_FIFO_TX2_8、UART_FIFO_TX4_8、UART_FIFO_TX6_8、UART_FIFO_TX7_8

ui32RxLevel取值：UART_FIFO_RX1_8、UART_FIFO_RX2_8、UART_FIFO_RX4_8、UART_FIFO_RX6_8、UART_FIFO_RX7_8

例：设置发送FIFO为1/2深度，接收FIFO为7/8深度

```
UARTFIFOLevelSet (UART0_BASE, UART_FIFO_TX4_8, UART_FIFO_RX7_8);
```

- UART模块初始化配置　详见S800_UART_Init()

1. 使能UART0模块

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
```

2. 使能提供UART接口信号的GPIOA端口

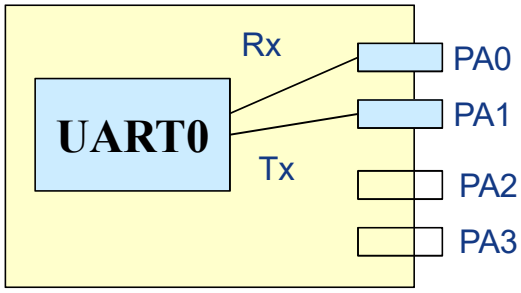
```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

3. 将GPIO引脚PA0, PA1设置为UART复用功能

```
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

4. 配置UART0的 时钟/波特率/帧格式 和 FIFO深度配置（可选）

```
UARTConfigSetExpClk(UART0_BASE, ui32SysClock,115200, …);
UARTFIFOLevelSet(UART0_BASE,UART_FIFO_TX2_8,UART_FIFO_RX4_8);
```



实验三 UART串行通讯口实验

- 实验内容
  - 例程exp3-1.c：对实验板UART0 进行初始化，在初始化完成后向主机发送“HELLO,WORLD！”。Task1完成UART ECHO，即通过PC端串口调试窗口向实验板发送字符串，实验板收到后原样返回；Task2为跑马灯控制。
  - 编程要点
    1. UART初始化配置
    2. **UART数据发送**
    3. UART数据接收

- UART数据发送函数
  - 阻塞发送：查询式数据发送，若TxFIFO没有空位，则等待直到发送成功

```
void UARTCharPut ( uint32_t ui32Base,          //UART端口地址
                   unsigned char ucData);      //要发送的字符
```

  - 非阻塞发送：若TxFIFO有空位则发送，否则不发送直接返回

```
void UARTCharPutNonBlocking (uint32_t ui32Base,        //UART端口地址
                             unsigned char ucData); //要发送的字符
```

  - 查询TxFIFO是否有空位：若有返回true，否则false

```
bool UARTSpaceAvail (uint32_t ui32Base);
```

- UART发送编程示例
  - 阻塞发送字符串

```
void UARTStringPut(const unsigned char *msg )
{
    while(*msg != '\0')
       UARTCharPut(UART0_BASE, *msg++);        //发送一个字符直到发送成功
}
```

调用方式：

UARTStringPut("\r\nHello, world!\r\n");

  - 非阻塞发送字符串

```
void UARTStringPutNonBlocking(const unsigned char *msg)
{
    while(*msg != '\0') {
        if (UARTSpaceAvail(UART0_BASE))          //发送FIFO有空位
          UARTCharPutNonBlocking(UART0_BASE, *msg++); //发送一个字符
    }
}
```

实验三 UART串行通讯口实验

- 实验内容
  - 例程exp3-1.c：对实验板UART0 进行初始化，在初始化完成后向主机发送“HELLO,WORLD！”。Task1完成UART ECHO，即通过PC端串口调试窗口向实验板发送字符串，实验板收到后原样返回；Task2为跑马灯控制。
  - 编程要点
    1. UART初始化配置
    2. UART数据发送
    3. **UART数据接收**

- UART数据接收函数
  - 阻塞接收：查询方式的数据接收， RxFIFO无数据时等待直到接收成功

```
int32_t UARTCharGet ( uint32_t ui32Base);
```

  - 非阻塞接收：若RxFIFO有数据则读取，否则直接返回false（常用于中断方式）

```
int32_t UARTCharGetNonBlocking ( uint32_t ui32Base);
```

  - 查询RxFIFO是否有数据

```
bool UARTCharsAvail (uint32_t ui32Base);
```

- UART编程：exp3-1.c 示例

```
int main(void)
{
    unsigned char uart_receive_char;
    ……
    while (1)
    {   //Task1 uart echo
        if (UARTCharsAvail(UART0_BASE)) {
            uart_receive_char = UARTCharGet(UART0_BASE);
            UARTCharPut(UART0_BASE, uart_receive_char);
        }
        ……
    }
}（试修改程序，将收到的字符改为大写后回显）
```

- UART数据发送和接收编程要点
  - UART数据发送：阻塞式发送
    - void UARTCharPut (uint32_t ui32Base, unsigned char ucData);
    - bool UARTSpaceAvail (uint32_t ui32Base); //多任务时可避免程序阻塞
  - UART数据接收：阻塞式或非阻塞式
    - 阻塞式接收：int32_t UARTCharGet (uint32_t ui32Base);
    - 非阻塞式接收：采用中断方式，使能接收和接收超时条件

实验三 UART串行通讯口实验

- 实验内容
  - 例程3-2.c：将实验3-1的Task1改为非堵塞式方式（即中断方式）接收字符串，并原样返回，以回车换行“\r\n”表示一次接收完成。当进行数据接收时，点亮PN1。全部接收完成后，熄灭PN1。
  - 编程要点
    1. 前后台程序结构：增加Uart接收状态标志 uart_receive_status
    2. **UART中断设置、使能、注册和编写中断服务程序等**
    3. UART数据接收和发送

- UART中断编程的基本步骤

1. UART相关外设使能，并进行基本的配置
2. 设置UART中断触发的FIFO深度（可选）
3. 设置中断分组和GPIO中断优先级（可选）
4. 使能中断 （UART源级、NVIC级、内核级）
5. 重写UART中断服务函数

1. 使能相关片上外设，并进行基本的配置（同UART初始化）
2. 设置UART中断触发的FIFO深度（可选）

   UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX4_8, UART_FIFO_RX7_8);
3. 设置中断优先级（可选）

   IntPrioritySet (INT_UART0, 0x00);
4. 使能中断
   - UART源级：指明中断触发条件（接收及接收超时）

     UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
   - NVIC级： IntEnable(INT_UART0);
   - 内核级： IntMasterEnable( ); （在main函数设置）
5. 重写UART中断服务函数：void UART0_Handler(void)

- UART中断编程示例（中断方式接收字符串）

```
void S800_UART_Init(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // Set GPIO A0 and A1 as UART pins.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    // Configure the UART for 115,200, 8-N-1 operation.
    UARTConfigSetExpClk(UART0_BASE, ui32SysClock,115200, (UART_CONFIG_WLEN_8 |
        UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
    // set RxFIFO to 7/8, TxFIFO to 1/4
    UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX2_8, UART_FIFO_RX7_8);
    // Enable UART0 RX,TX interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
    IntEnable(INT_UART0);
}
```

- UART非堵塞式接收的中断服务程序

```
volatile uint8_t uart_receive_status = 0;  //全局变量
void UART0_Handler(void) //重写已经注册的中断处理函数
{
    int32_t uart0_int_status;
    uart0_int_status= UARTIntStatus(UART0_BASE, true);          // 获取中断状态
    UARTIntClear(UART0_BASE, uart0_int_status);                 // 清中断请求信号
    //确认中断触发条件
    if (uart0_int_status & (UART_INT_RX | UART_INT_RT))         //接收或接收超时
          uart_receive_status = 1;
} // 在if语句处设断点，观察uart0_int_status的值
```

S800实验板UART串行通讯实验

- 实验内容
  - 实验3-3.c：编程实现一个虚拟AT指令集：

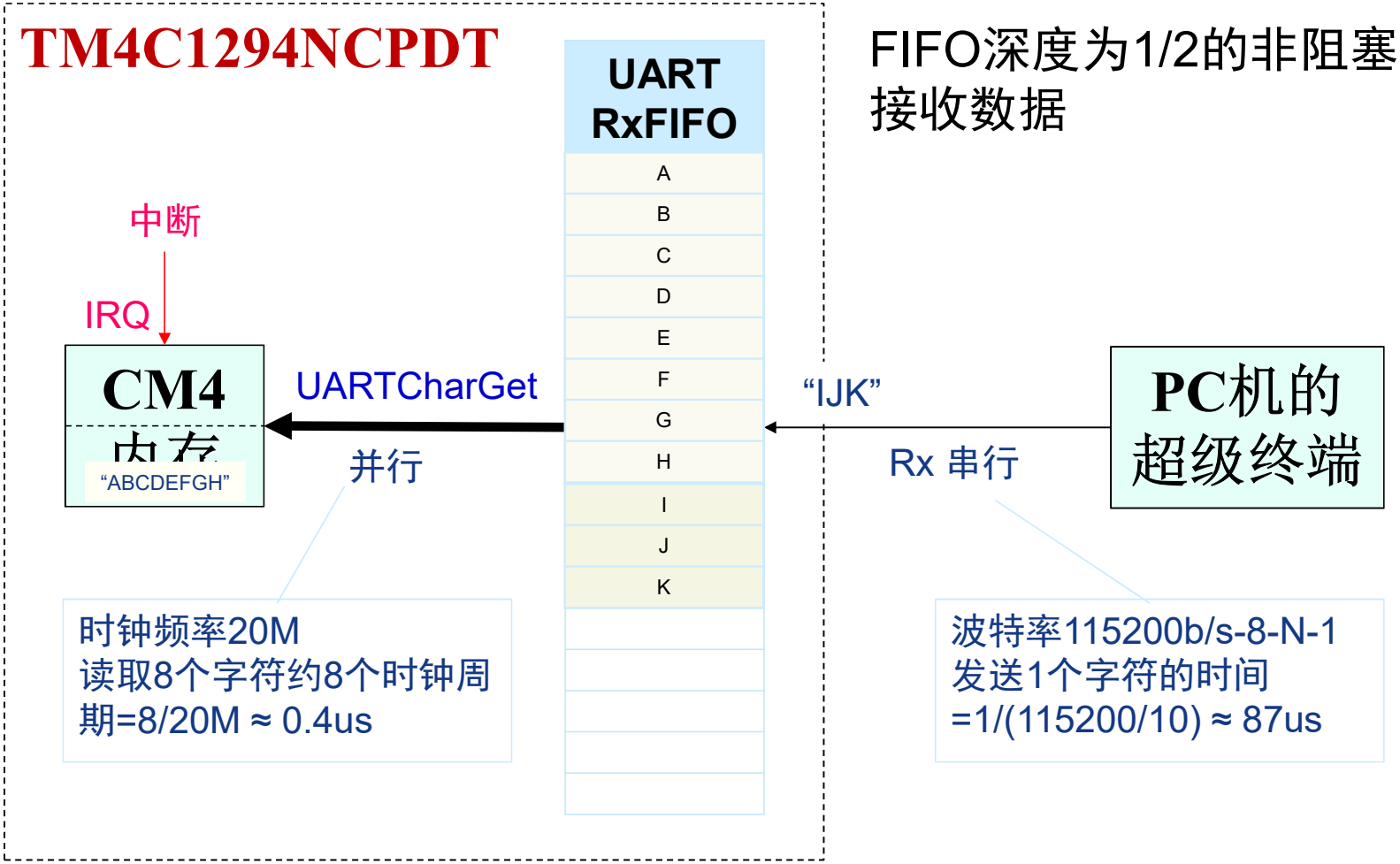
    当PC端发AT+CLASS，实验板回以CLASS###，###为班级号；

    当PC端发AT+STUDENTCODE，回以CODE###，###为学号；

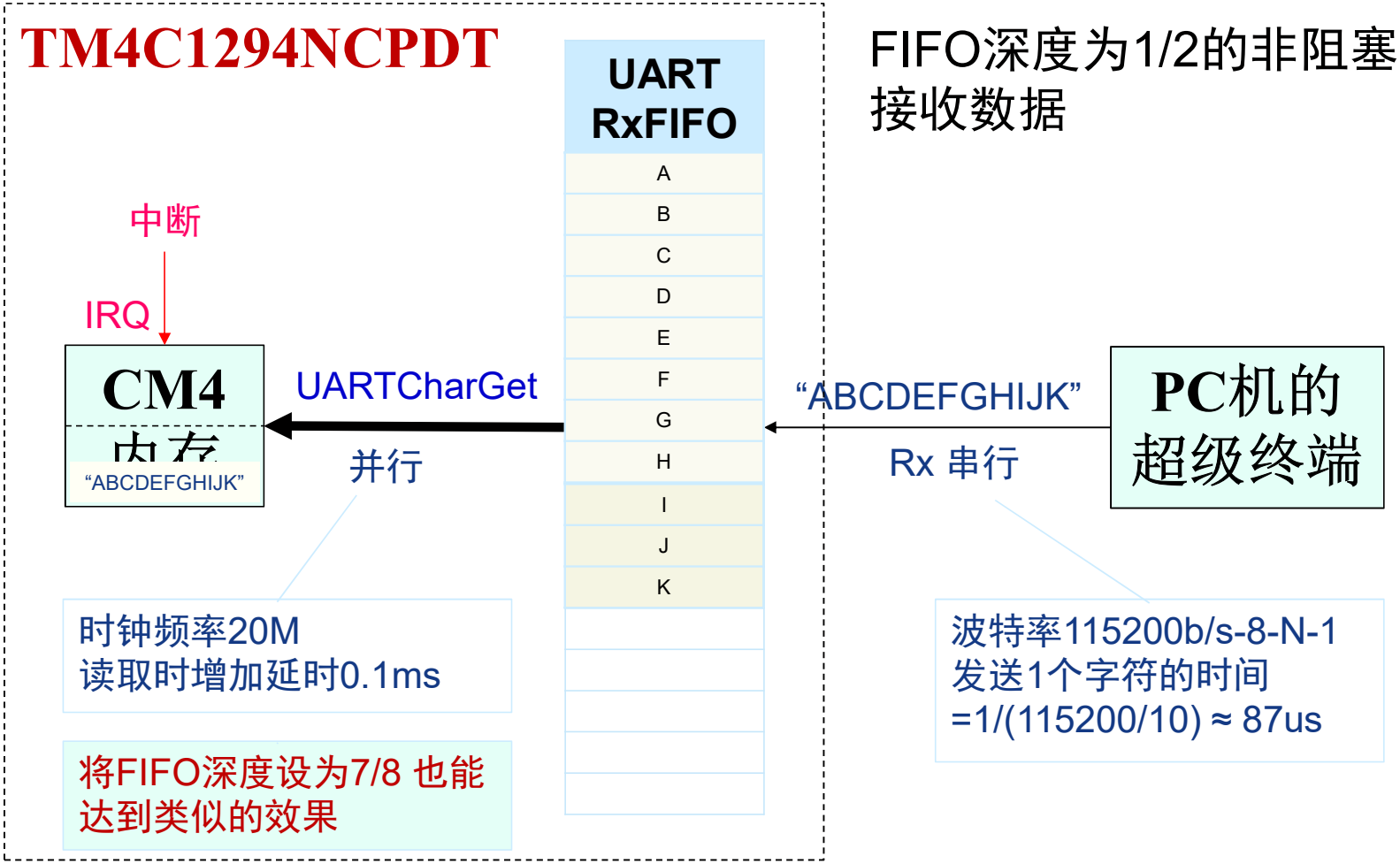
    当PC端发其它内容，回以Unknown
  - 编程要点
    1. **逐个接收字符，并拼接成字符串**

       ※ 注意：非阻塞接收方式下，FIFO深度与中断进入次数的关系
    2. 字符串的比较：使用C语言函数，strcmp()或strncmp()、toupper()/tolower()等

       ※ #include <string.h>，#include <ctype.h>



FIFO深度为1/2的非阻塞接收数据



FIFO深度为1/2的非阻塞接收数据

- RxFIFO深度与UART数据接收
  - 当接收字符数超出FIFO触发深度时，数据可能分多次送达，此时需考虑字符串的拼接
- 解决方法：
  1. 调整RxFIFO深度：RxFIFO深度为7/8时，可以一次性接收14个字符数

     在 S800_UART_Init 中设置：

     UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX4_8, UART_FIFO_RX7_8);
  2. 收发双方约定一个字符串结束标志
  3. 增加延时：接收数据时适当增加延时可保证在一次循环中接收所有字符
     - 延时时间计算：若传输协议为 115200-8-N-1，需要设置延时 ≥ 1/(115200/10)= 87us

     ※ 注意：增加延时可能会影响整个程序的运行效率，多任务情况下不推荐

实验三 UART串行通讯口实验

- 实验内容
  - 例程3-4.c：优先级调整实验。三个任务：
    - 任务1，主循环走马灯；
    - 任务2，Systick中断中点亮D1，长按USR_SW1点亮D2，释放熄灭D2；
    - 任务3，UART0中断中熄灭D1和D2，长按USR_SW2保持UART接收状态。
  - 编程要点
    1. **优先级设置**
    2. **中断驱动程序结构：任务处理在中断服务程序中完成**

- 中断优先级设置
  - IntPrioritySet() 和 IntPriorityGet()： 设置和检查中断的优先级
  - IntPriorityGroupingSet()：设置中断分组

```
void IntPrioritySet (uint32_t ui32Interrupt, uint8_t ui8Priority);
void IntPriorityGroupingSet (uint32_t ui32Bits);
```

其中：ui32Interrupt为中断号， ui8Priority为中断优先级，
ui32Bits指定抢占优先级的位数，默认为7（全抢占）

抢占级位数	Bit765二进制位	组优先级	子优先级	抢占式可选配置	子优先可选配置
0	b .yyy	无	3位: [7-5]	0	0~7
1	b x.yy	1位: [7]	2位: [6-5]	0~1	0~3
2	b xx.y	2位: [7-6]	1位: [5]	0~3	0~1
3-7	b xxx.	3位: [7-5]	无	0~7	0

- 设置优先级分组和中断优先级：
  - 调用优先级分组函数 IntPriorityGroupingSet()

    IntPriorityGroupingSet(7); //中断分组设为全抢占
  - 调用中断优先级设置函数 IntPrioritySet( )

    IntPrioritySet(INT_GPIOJ, 0x20); //设 PortJ 中断优先级

    IntPrioritySet(INT_UART0, 0x03); //设 UART0 中断优先级

    IntPrioritySet(FAULT_SYSTICK, 0x0e0); //设SYSTICK最低优先级
  - 问：Uart0的实际优先级为多少?

- 中断驱动的程序结构：
  - 调整UART0 的优先级，使之高于SYSTICK 的优先级，并处于抢占式优先。这样当按下USR_SW2 时，UART0 中断不退出，导致SYSTICK 中断不能进入，任务2无法执行，同时整个系统也都停滞（除了能继续uart接收外），任务1显示不再跳变。
  - 当Uart0优先级 > SysTick优先级时：
    - 长按USR_SW1， SysTick中断不退出，D1D2点亮，任务1停止走马灯。此时若超级终端有数据传送时，Uart0应能抢占SysTick的中断，进行数据收发。
    - 长按USR_SW2且超级终端有数据传送，UART0中断不退出。此时除了能继续uart数据收发外，系统其他任务都停滞，SYSTICK 中断不能进入，任务2无法执行，任务1停止走马灯。
  - 修改任务优先级，观察并解释当SysTick优先级大于或等于Uart0优先级时，长按user_Sw1和user_Sw2的现象。
  - 若将中断分组改为 IntPriorityGroupingSet(0); 又会如何?

- 完成实验三，提交实验报告和源程序 -