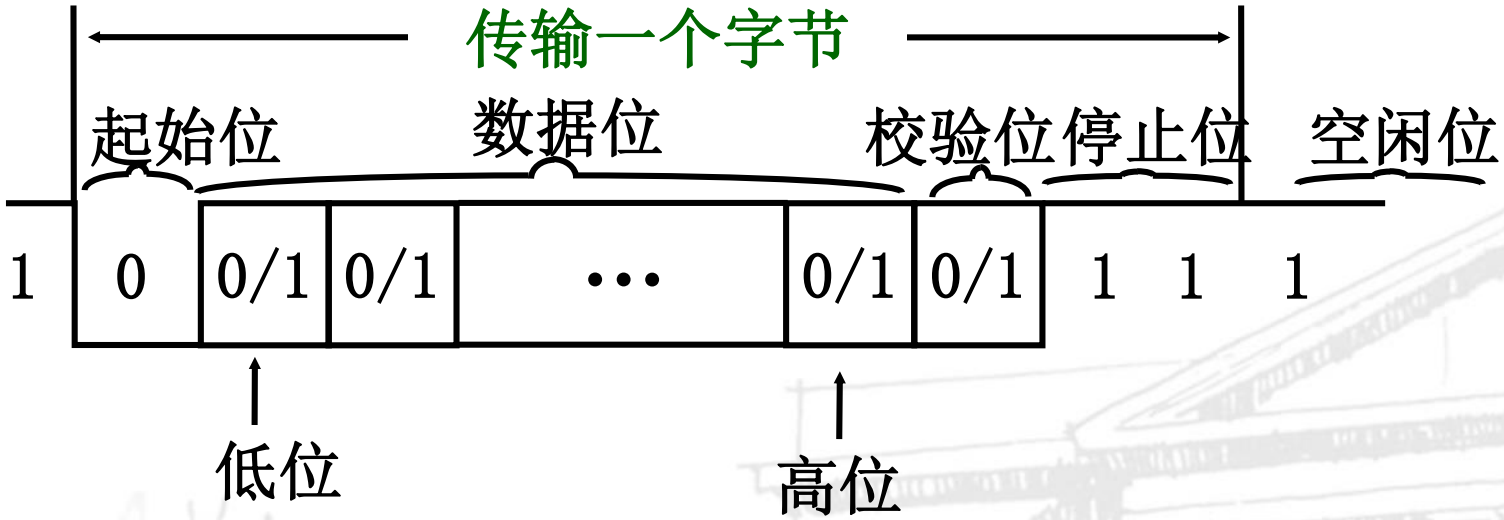
通用异步串行接口2

程晨闻

东南大学电气工程学院

➢ 异步串行通讯基础

- 分类（RS-232，RS485，RS-422，CAN，USB）
- 通讯协议
  - 起始位，奇偶校验位，停止位（1~2）
  - 数据位：先传输低位
  - 波特率



➢ 内容概要

–通用异步接收发送模块UART的工作原理

–通用异步接收发送模块UART的使用方法

–开发板UART功能的硬件连接与调试

➢ TM4C1294中的UART模块

– Universal Asynchronous Receiver/Transmitter (UART)

– 可编程波特率，最高可达7.5Mbps/15Mbps

– 发送和接收都有16*8 的先入先出寄存器（FIFO），深度可编程

– 可编程数据位、起始位、校验位

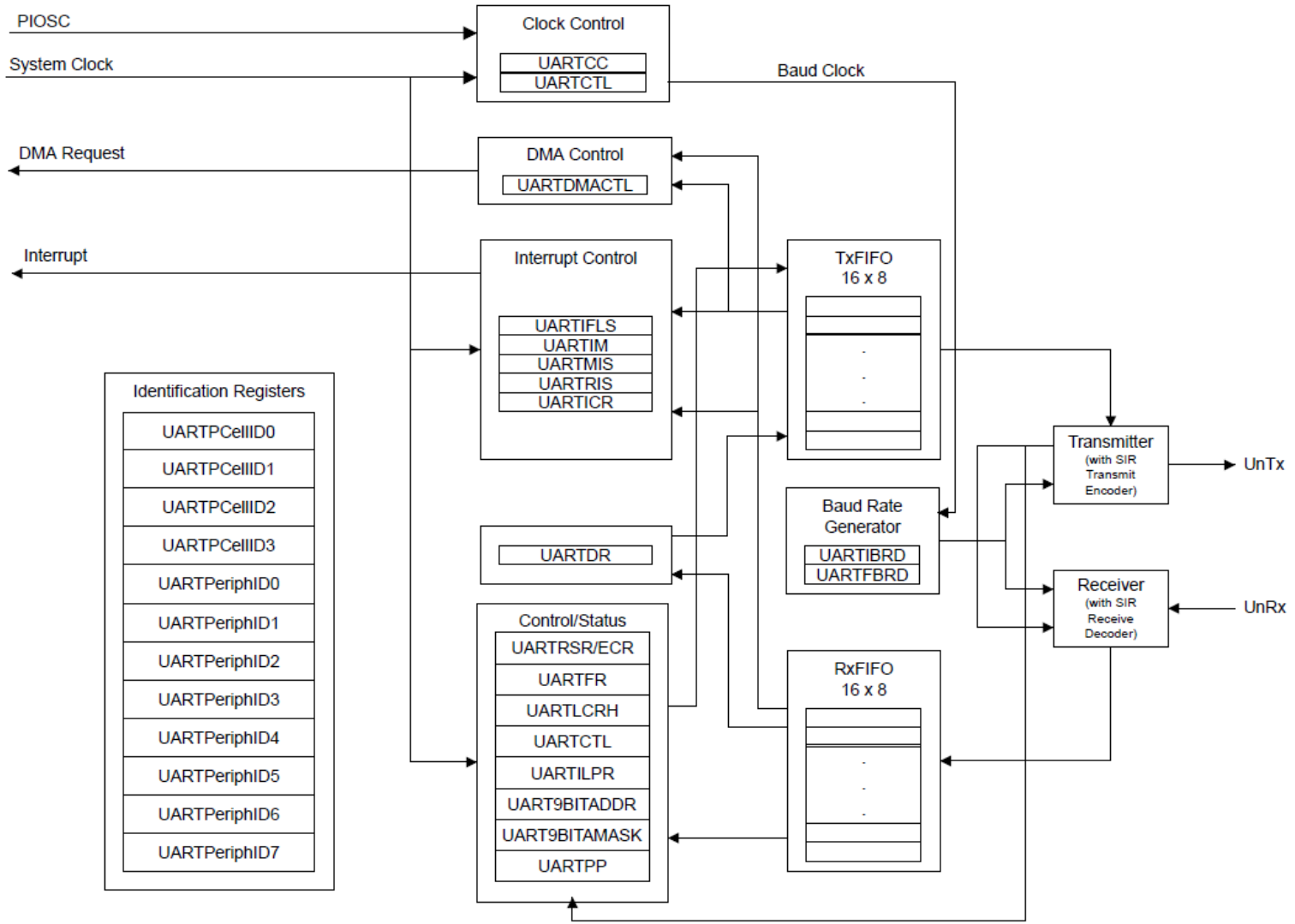
– 红外接口

– 多样化的中断系统

– 带DMA接口

– 流控制与状态信号

UART模块结构


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

➢ 功能描述

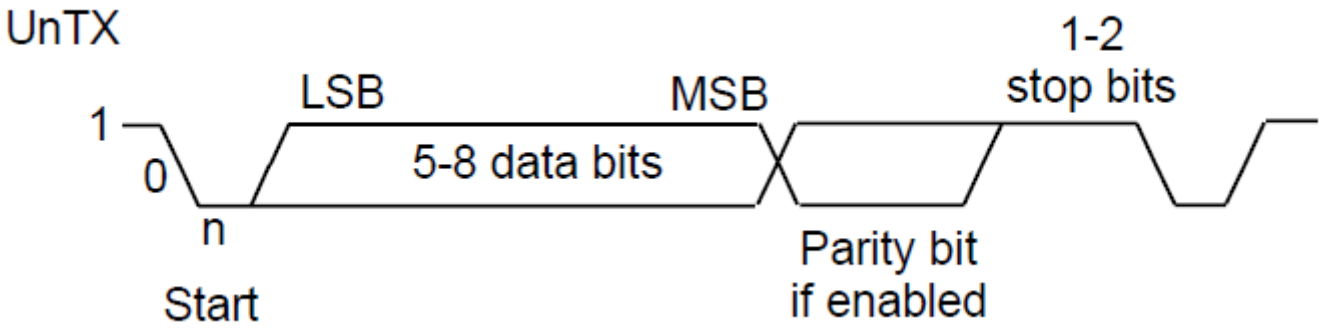
– UART通过**UARTCTL**寄存器的**TXE**和**RXE**位控制发送和接收功能

– 重启后UART发送和接收功能就有效

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CTSEN	RTSEN	reserved		RTS	DTR	RXE	TXE	LBE	reserved	HSE	EOT	SMART	SIRLP	SIREN	UARTEN
Type	RW	RW	RO	RO	RW	RW	RW	RW	RW	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

➢ 数据收发时序



数据发送

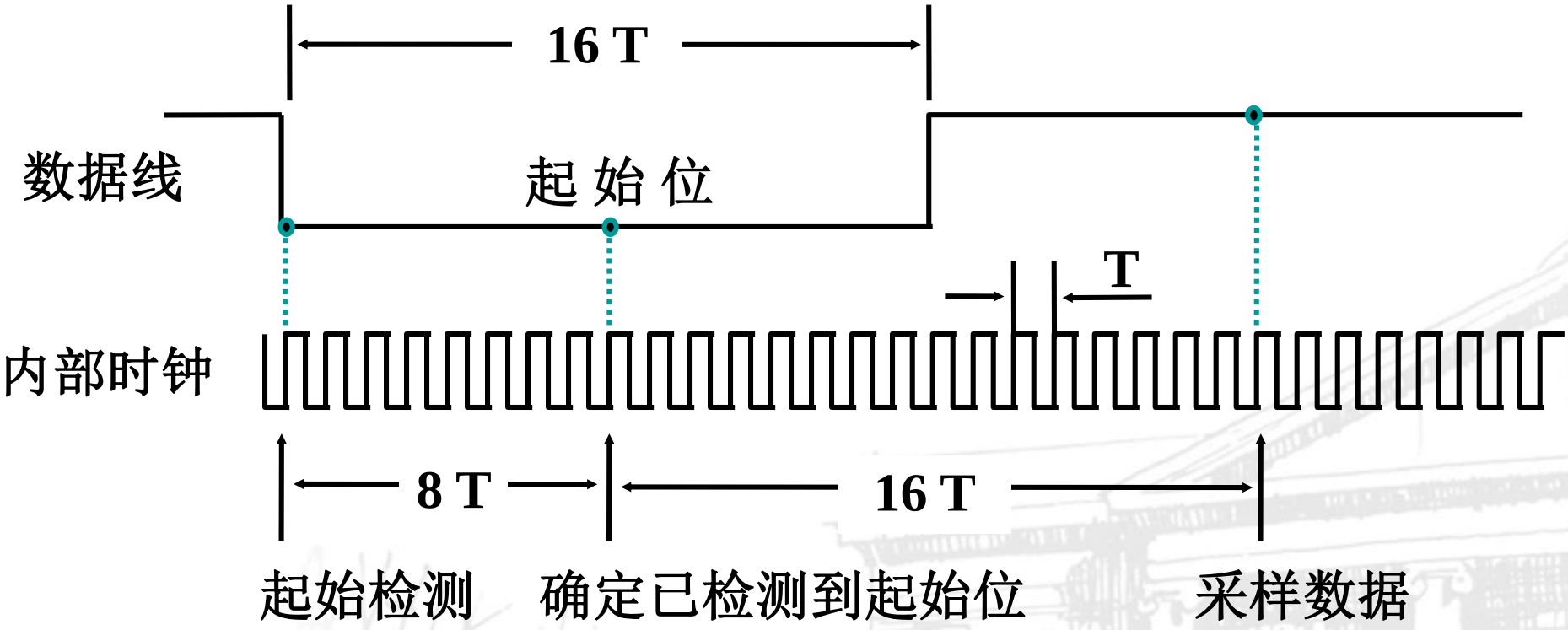
– 先把要发送的数据写入到发送FIFO（TxFIFO）中，向UARTDR寄存器中写入数据，这个数据就会自动填入到TxFIFO中。

– 只要TxFIFO中有数据，UART模块就会不停的把TxFIFO中的数据按照异步串行数据的通信格式发送出去，直到TxFIFO中的数据发送完为止

➢ 数据接收

– 收到起始信号后，在第**8**个（或者第**4**个）内部参考时钟周期后采样引脚上的电平，如果是低电平，则确认收到起始位

– 之后每**16**个（或者第**8**个）内部参考时钟周期(T)采样一次数据，存到RxFIFO中



➢ TM4C1294中的UART模块使用方法

- 1. 使能UART模块的时钟（操作**RCGCUART** 寄存器）
- 2. 使能UART模块引脚用到的GPIO模块（使用 **RCGCGPIO** 寄存器）
- 3. 配置GPIO的复用功能，使其与UART模块相连
- 4. 计算配置波特率分频器UARTIBRD和UARTFBRD
- 5. **禁用UART模块**，配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，**使能UART模块**
- 6. 使用UART模块发送和接收数据

➢ TM4C1294中的UART模块使用方法

– 第一步：使能UART模块的时钟（操作RCGCUART 寄存器）

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
```

Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART)

Base 0x400F.E000
Offset 0x618
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
#define SYSCTL_RCGCBASE        0x400fe600
#define SYSCTL_PERIPH_UART0    0xf0001800  // UART 0
```

```
void
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
{
    // Check the arguments.
    ASSERT(_SysCtlPeripheralValid(ui32Peripheral));
    // Enable this peripheral.
    HWREGBITW(SYSCTL_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),
              ui32Peripheral & 0xff) = 1;
}
```

➢ TM4C1294中的UART模块使用方法

– 第二步：使能UART模块引脚用到的GPIO模块（使用RCGCGPIO寄存器）

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

U0Rx	33	PA0 (1)	I	TTL	UART module 0 receive.
U0Tx	34	PA1 (1)	O	TTL	UART module 0 transmit.

➢ TM4C1294中的UART模块使用方法

– 第三步：配置GPIO的复用功能，使其与UART模块相连

设置GPIOAFSEL寄存器，使能引脚的复用功能。可以调用GPIOPinTypeUART函数实现

```
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

```
void
GPIOPinTypeUART(uint32_t ui32Port, uint8_t ui8Pins)
{
    //
    // Check the arguments.
    //
    ASSERT(_GPIOBaseValid(ui32Port));

    //
    // Make the pin(s) be peripheral controlled.
    //
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);

    //
    // Set the pad(s) for standard push-pull operation.
    //
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);
}
```

➢ TM4C1294中的UART模块使用方法

– 第三步：配置GPIO的复用功能，使其与UART模块相连

使用GPIOPCTL的PMCx位的配置PA0和PA1的复用功能，为UART0模块的Rx和Tx引脚

IO	Pin	Analog or Special Function[a]	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[b]											
			1	2	3	4	5	6	7	8	11	13	14	15
PA0	33	-	U0Rx	I2C9SCL	T0CCP0	-	-	-	CAN0Rx	-	-	-	-	-
PA1	34	-	U0Tx	I2C9SDA	T0CCP1	-	-	-	CAN0Tx	-	-	-	-	-

可以调用GPIOPinConfigure函数实现：

```
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
```

➢ TM4C1294中的UART模块使用方法

– 第三步：配置GPIO的复用功能，使其与UART模块相连

```
void
GPIOPinConfigure(uint32_t ui32PinConfig)
{
    uint32_t ui32Base, ui32Shift;
    …
    // Extract the base address index from the input value.
    ui32Base = (ui32PinConfig >> 16) & 0xff;
    // Get the base address of the GPIO module, selecting either the APB or the
    // AHB aperture as appropriate.
    if(HWREG(SYSCTL_GPIOHBCTL) & (1 << ui32Base))
    {
        ui32Base = g_pui32GPIOBaseAddrs[(ui32Base << 1) + 1];
    }
    else
    {
        ui32Base = g_pui32GPIOBaseAddrs[ui32Base << 1];
    }
    // Extract the shift from the input value.
    ui32Shift = (ui32PinConfig >> 8) & 0xff;
    // Write the requested pin muxing value for this GPIO pin.
    HWREG(ui32Base + GPIO_O_PCTL) = ((HWREG(ui32Base + GPIO_O_PCTL) &
                                      ~(0xf << ui32Shift)) |
                                     ((ui32PinConfig & 0xf) << ui32Shift));
}
```

```
#define GPIO_PA0_U0RX          0x00000001
#define GPIO_PA1_U0TX          0x00000401
```

➢ TM4C1294中的UART模块使用方法

– **第四步：计算配置波特率分频器UARTIBRD和UARTFBRD**

- UART模块的波特率发生器由一个**22**位的分频器，将系统时钟分频为所需要的时钟
- UARTIBRD的**低16位**组成分频器的**整数位**，UARTFBRD的**低6位**组成分频器的**小数位**

分频器与波特率的关系为

$$BRD = BRDI + BRDF = UARTSysClk / (ClkDiv * Baud\ Rate)$$

其中，

- Baud Rate 为波特率
- ClkDiv为发送/接收每个位所用的内部时钟的个数一般为16，高速模式下，可以配置为8
- BRD为波特率分频系数
- BRDI为波特率分频系数的整数部分，直接填入到UARTIBRD寄存器中
- BRDF为波特率分频系数的小数部分，计算方式为：

  UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)

➢ TM4C1294中的UART模块使用方法

– **第五步：禁用UART模块（ UARTCTL），配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，使能UART模块**

数据格式等参数，由UART线路控制寄存器**UARTLCRH**控制。

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x02C
Type RW, reset 0x0000.0000

SPS：强制校验位
如果SPS、EPS、PEN都为1，则校验位强制为0;
如果SPS、PEN都为1，EPS为0，则校验位强制为1;
如果SPS为0，则禁用强制校验位功能，校验位由EPS、PEN和数据决定。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								SPS	WLEN		FEN	STP2	EPS	PEN	BRK
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

WLEN：0x0: 5bits; 0x1: 6 bits; 0x2: 7 bits; 0x3: 8 bits

FEN：0，禁用FIFO，即FIFO的深度为1；1：启用FIFO

STP2：停止位选择，0：一个停止位，1：2个停止位

EPS：奇偶校验选择：0：奇校验，1：偶校验，需要PEN位为1才有效

PEN：校验位使能：0：关闭奇偶校验功能，无校验位；1：开启奇偶校验功能，有校验位

➢ TM4C1294中的UART模块使用方法

– 第五步：禁用**UART**模块，配置数据格式等参数（**UARTLCRH**），配置时钟源**UARTCC**，使能**UART**模块

UART状态标志寄存器UARTFR：

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x018
Type RO, reset 0x0000.0090

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved							RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

BUSY：0：UART模块空闲，1：UART模块正忙

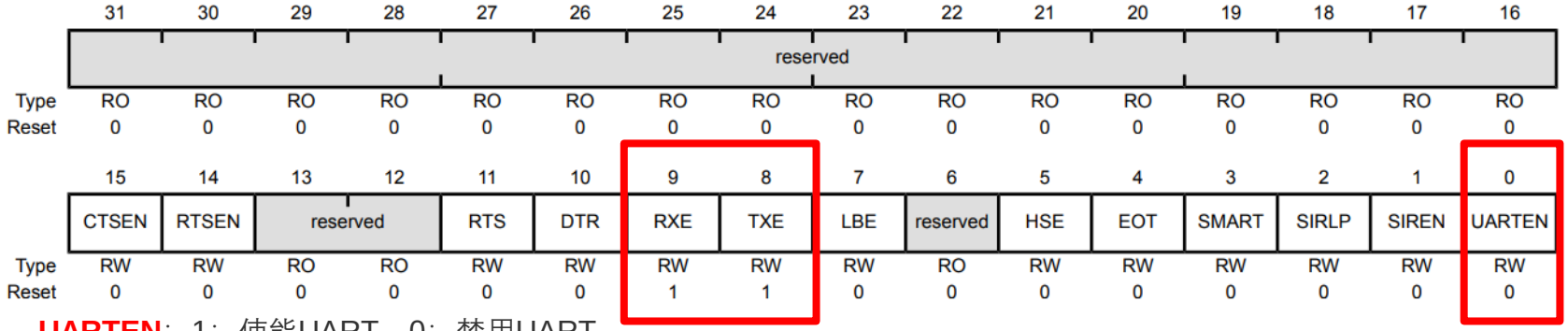
➢ TM4C1294中的UART模块使用方法

– **第五步：禁用UART模块，配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，使能UART模块**

使能和禁止UART模块，由UART控制寄存器**UARTCTL**控制。

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x030
Type RW, reset 0x0000.0300



UARTEN：1：使能UART，0：禁用UART

RXE：1：接收使能，同时UARTEN 也要设置为1，0：禁用接收功能

TXE：1：发送使能，同时UARTEN 也要设置为1，0：禁用发送功能

HSE，1：高速模式，即发送或接收一个位需要8个内部时钟，最高波特率位15Mbsp。0：常规模式，即发送或接收一个位需要16个内部时钟，最高波特率位7.5Mbps。

➢ TM4C1294中的UART模块使用方法

– **第五步：禁用UART模块，配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，使能UART模块**

使能和禁止UART模块，由UART控制寄存器**UARTCTL**控制。

禁用UART模块，调用函数**UARTDisable**，其具体实现方式为：

```
void
UARTDisable(uint32_t ui32Base)
{
    // Check the arguments.
    ASSERT(_UARTBaseValid(ui32Base));
    // Wait for end of TX.
    //
    while(HWREG(ui32Base + UART_O_FR) & UART_FR_BUSY)
    {
    }
    // Disable the FIFO.
    //
    HWREG(ui32Base + UART_O_LCRH) &= ~(UART_LCRH_FEN);
    // Disable the UART.
    //
    HWREG(ui32Base + UART_O_CTL) &= ~(UART_CTL_UARTEN | UART_CTL_TXE |
                                      UART_CTL_RXE);
}
```

➢ TM4C1294中的UART模块使用方法

– **第五步：禁用UART模块，配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，使能UART模块**

使能和禁止UART模块，由UART控制寄存器**UARTCTL**控制。

使能UART模块，调用函数**UARTEnable**，其具体实现方式为：

```
void
UARTEnable(uint32_t ui32Base)
{
    //
    // Check the arguments.
    //
    ASSERT(_UARTBaseValid(ui32Base));
    //
    // Enable the FIFO.
    //
    HWREG(ui32Base + UART_O_LCRH) |= UART_LCRH_FEN;

    //
    // Enable RX, TX, and the UART.
    //
    HWREG(ui32Base + UART_O_CTL) |= (UART_CTL_UARTEN | UART_CTL_TXE |
                                     UART_CTL_RXE);
}
```

➢ TM4C1294中的UART模块使用方法

– 第五步：禁用UART模块，配置数据格式等参数（UARTLCRH），配置时钟源UARTCC，使能UART模块

TivaWare提供了UARTConfigSetExpClk函数，一次性初始化UART模块：

```
UARTConfigSetExpClk(UART0_BASE, g_ui32SysClock, 115200,
                    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                     UART_CONFIG_PAR_NONE));
```

```
void
UARTConfigSetExpClk(uint32_t ui32Base, uint32_t ui32UARTClk,
                    uint32_t ui32Baud, uint32_t ui32Config)
{
    uint32_t ui32Div;
    // …
    // Stop the UART.
    UARTDisable(ui32Base);
    // Is the required baud rate greater than the maximum rate supported
    // without the use of high speed mode?
    if((ui32Baud * 16) > ui32UARTClk)
    {   // Enable high speed mode.
        HWREG(ui32Base + UART_O_CTL) |= UART_CTL_HSE;
        // Half the supplied baud rate to compensate for enabling high speed
        // mode.  This allows the following code to be common to both cases.
        ui32Baud /= 2;
    }
    else
    {   // Disable high speed mode.
        HWREG(ui32Base + UART_O_CTL) &= ~(UART_CTL_HSE);
    }
    // Compute the fractional baud rate divider.
    ui32Div = (((ui32UARTClk * 8) / ui32Baud) + 1) / 2;
    // Set the baud rate.
    HWREG(ui32Base + UART_O_IBRD) = ui32Div / 64;
    HWREG(ui32Base + UART_O_FBRD) = ui32Div % 64;
    // Set parity, data length, and number of stop bits.
    HWREG(ui32Base + UART_O_LCRH) = ui32Config;
    // Clear the flags register.
    HWREG(ui32Base + UART_O_FR) = 0;
    // Start the UART.
    UARTEnable(ui32Base);
}
```

```
#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
#define UART_CONFIG_PAR_ONE     0x00000082  // Parity bit is one
#define UART_CONFIG_PAR_ZERO    0x00000086  // Parity bit is zero
```

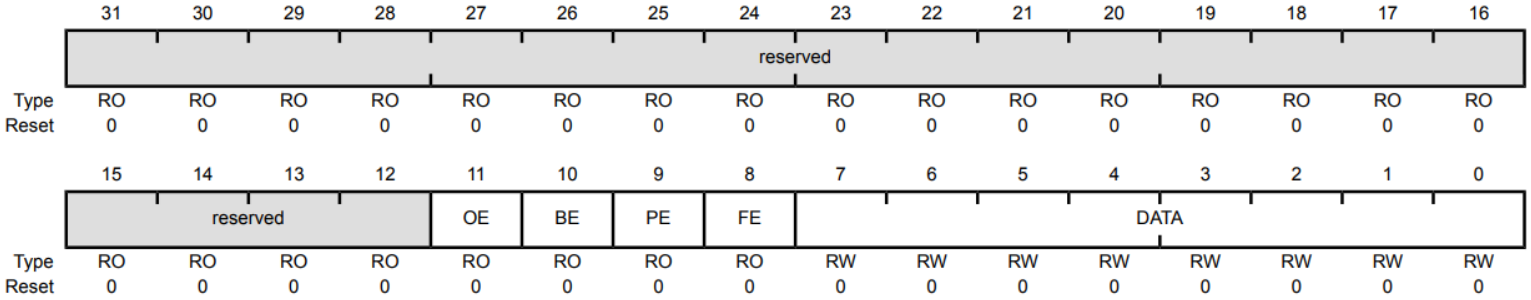
➢ TM4C1294中的UART模块使用方法

– 第六步：使用**UART模块发送和接收数据**

UART数据寄存器 UARTDR

UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x000
Type RW, reset 0x0000.0000



DATA：数据位

如果要**发送数据**，就把数据**写入UARTDR 寄存器的DATA**域中，写入DATA的数据会被转移到发送FIFO（TxFIFO）中，然后UART模块开始发送数据，直到TxFIFO中的数据全部发送完成为止。

如果要**读取数据**，读DATA域就会把收到的数据从接收FIFO（RxFIFO）中读出来，读数据的时候，不仅有DATA域，还有OE、BE、PE、FE四个状态位，用来指示本次接收的数据是否有错误。

OE：溢出错误。

BE: Break error。

PE: 奇偶校验错误。

FE：帧错误。

➢ TM4C1294中的UART模块使用方法

– 第六步：使用**UART模块发送和接收数据**

UART状态标志寄存器UARTFR

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x018
Type RO, reset 0x0000.0090

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved							RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

BUSY忙碌标志：0：UART模块空闲，1：UART模块正忙

TXFF TxFIFO满标志：0：TxFIFO未满，可以继续填入数据。1：TxFIFO满。

RXFE RxFIFO空标志：0：RxFIFO中有数据，需要把数据取出来。1：RxFIFO为空，还没有收到数据

➢ TM4C1294中的UART模块使用方法

– 第六步：使用UART模块发送和接收数据

UART状态标志寄存器UARTFR

可用UARTCharsAvail函数判断接收FIFO是否有数据

```
bool
UARTCharsAvail(uint32_t ui32Base)
{
    //
    // Check the arguments.
    //
    ASSERT(_UARTBaseValid(ui32Base));

    //
    // Return the availability of characters.
    //
    return((HWREG(ui32Base + UART_O_FR) & UART_FR_RXFE) ? false : true);
}
```

➢ TM4C1294中的UART模块使用方法

– 第六步：使用UART模块发送和接收数据

TivaWare提供了**UARTCharPut**函数和**UARTCharPutNonBlocking**函数发送一个字节的数据。

UARTCharPut函数的实现方式如下：

阻塞模式

```
void
UARTCharPut(uint32_t ui32Base, unsigned char ucData)
{
    //
    // Check the arguments.
    ASSERT(_UARTBaseValid(ui32Base));
    //
    // Wait until space is available.
    //
    while(HWREG(ui32Base + UART_O_FR) & UART_FR_TXFF)
    {
    }
    //
    // Send the char.
    //
    HWREG(ui32Base + UART_O_DR) = ucData;
}
```

➢ TM4C1294中的UART模块使用方法

– 第六步：使用UART模块发送和接收数据

TivaWare提供了**UARTCharPut**函数和**UARTCharPutNonBlocking**函数发送一个字节的数据。

UARTCharPutNonBlocking函数的实现方式如下：

非阻塞模式

```
bool
UARTCharPutNonBlocking(uint32_t ui32Base, unsigned char ucData)
{
    // Check the arguments.
    ASSERT(_UARTBaseValid(ui32Base));
    // See if there is space in the transmit FIFO.
    if(!(HWREG(ui32Base + UART_O_FR) & UART_FR_TXFF))
    {
        // Write this character to the transmit FIFO.
        HWREG(ui32Base + UART_O_DR) = ucData;
        // Success.
        return(true);
    }
    else
    {   // There is no space in the transmit FIFO, so return a failure.
        return(false);
    }
}
```

➢ TM4C1294中的UART模块使用方法

– 第六步：使用UART模块发送和接收数据

TivaWare提供了UARTCharGet函数和UARTCharGetNonBlocking函数接收一个字节的数据。

UARTCharGet函数的实现方式如下：

阻塞模式

```
int32_t
UARTCharGet(uint32_t ui32Base)
{
    //
    // Check the arguments.
    ASSERT(_UARTBaseValid(ui32Base));
    //
    // Wait until a char is available.
    while(HWREG(ui32Base + UART_O_FR) & UART_FR_RXFE)
    {
    }
    //
    // Now get the char.
    return(HWREG(ui32Base + UART_O_DR));
}
```

➢ TM4C1294中的UART模块使用方法

– 第六步：使用UART模块发送和接收数据

TivaWare提供了**UARTCharGet**函数和**UARTCharGetNonBlocking**函数接收一个字节的数据。

UARTCharGetNonBlocking函数的实现方式如下：

非阻塞模式

```
int32_t
UARTCharGetNonBlocking(uint32_t ui32Base)
{
    // Check the arguments.
    ASSERT(_UARTBaseValid(ui32Base));
    // See if there are any characters in the receive FIFO.
    if(!(HWREG(ui32Base + UART_O_FR) & UART_FR_RXFE))
    {
        // Read and return the next character.
        return(HWREG(ui32Base + UART_O_DR));
    }
    else
    {
        // There are no characters, so return a failure.
        return(-1);
    }
}
```

➢ TM4C1294中的UART模块使用方法

– UART模块的初始化及发送程序示例：

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

UARTConfigSetExpClk(UART0_BASE, g_ui32SysClock, 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                                 UART_CONFIG_PAR_NONE));
    uint32_t cThisChar;
    while(1)
    {
        cThisChar = UARTCharGet(UART0_BASE);

        UARTCharPut(UART0_BASE, cThisChar);

   }
```

这段程序的作用是什么？

谢谢！