FLEXIO FOR HALF DUPLEX UART

FRED FU

DEC 2019

SECURE CONNECTIONS
FOR A SMARTER WORLD

起因

- MKE1X仅仅只有3个串口
- MKE1X的FLEXIO为第一代的FLEXIO, 仅仅只有4个shifter和4个timer
- 如果使用全双工配置方式， 4个shifter和4个timer只能配置出2个串口
- 很多情况下，客户并不需要全双工的串口，那个是否可以只使用1个shifter和1个timer来模拟半双工的串口呢?

*vs Cortex-M33

起因

- **MKE1X仅仅只有3个串口**
- **MKE1X的FLEXIO为第一代的FLEXIO, 仅仅只有4个shifter和4个timer**
- **如果使用全双工配置方式， 4个shifter和4个timer只能配置出2个串口**
- **很多情况下，客户并不需要全双工的串口，那个是否可以只使用1个shifter和1个timer来模拟半双工的串口呢?**

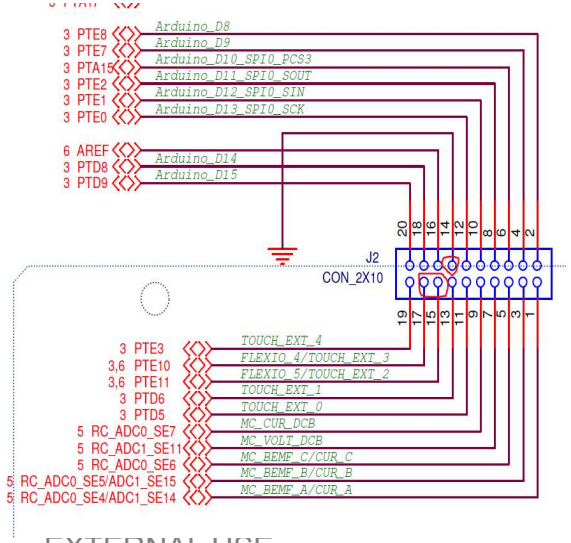
*vs Cortex-M33

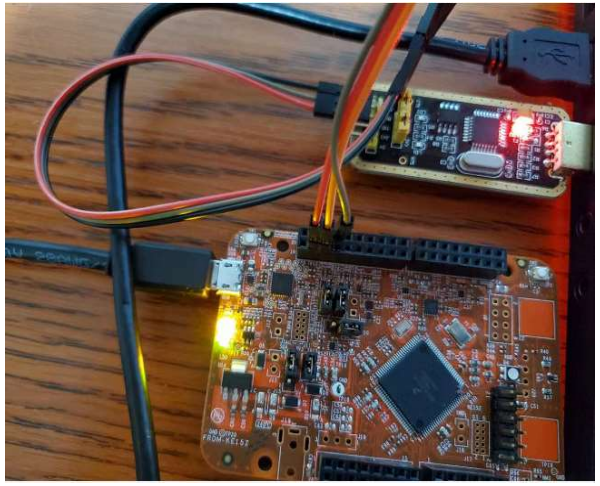
工具

- **FRDEM-KE15Z**
- **USB转TTL UART**
- **IAR 8.20.2**
- **SDK_2.6.0_MKE15Z256xxx7**

环境搭建

- 将USB转TTL UART插入电脑
- 将FRDM-KE15Z的调试口和PC用Micro USB连接
- 代开目录\boards\frdmke15z\driver_examples\flexio\uart\interrupt_transfer\iar下的flexio_uart_interrupt_transfer.eww工程





设计思路

- **TX和RX使用相同的shifter和timer编号**
- **平时处于接收状态，当收到设定字节数后，转为发送，发送完成再次切换为接收，如此反复**

FLEXIO_UART_TYPE结构体

- **SDK使用FLEXIO_UART_Type来定义串口配置结构体，如下图**
- **特别注意里面的两个数组**

  shifterIndex[2]: shifterIndex[0]定义TX的shifter编号，shifterIndex[1]定义RX的shifter编号

  timerIndex[2]: timerIndex[0]定义TX的timer编号，timerIndex[1]定义RX的timer编号

```
typedef struct _flexio_uart_type
{
    FLEXIO_Type *flexioBase; /*!< FlexIO base pointer. */
    uint8_t TxPinIndex;      /*!< Pin select for UART Tx. */
    uint8_t RxPinIndex;      /*!< Pin select for UART Rx. */
    uint8_t shifterIndex[2]; /*!< Shifter index used in FlexIO UART. */
    uint8_t timerIndex[2];   /*!< Timer index used in FlexIO UART. */
} FLEXIO_UART_Type;
```

```
uartDev.flexioBase      = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex      = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex      = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;                 //TX
uartDev.shifterIndex[1] = 0U;       //1U;
uartDev.timerIndex[0]   = 0U;                 //RX
uartDev.timerIndex[1]   = 0U;       //1U;
```

初始化函数修改

- **原来SDK中初始化函数FLEXIO_UART_Init同时操作了控制寄存器设置，TX和RX的shifter和timer的设置，收发之间有很高的耦合度，现在需要将其拆成5个函数**
- **FLEXIO_UART_CTRL_Init(&uartDev, &config) 配置控制寄存器**
- **FLEXIO_UART_ShifterTX_Init(&uartDev)配置TX使用的shifter**
- **FLEXIO_UART_TimerTX_Init(&uartDev,&config,FLEXIO_CLOCK_FREQUENCY)配置TX使用的timer**
- **FLEXIO_UART_ShifterRX_Init(&uartDev)配置RX使用的shifter**
- **FLEXIO_UART_TimerRX_Init(&uartDev,&config,FLEXIO_CLOCK_FREQUENCY)配置RX使用的timer**

UART HANDLE函数修改

- **原来SDK中UART HANDLE初始化函数FLEXIO_UART_TransferCreateHandle里面调用了函数FLEXIO_RegisterHandleIRQ去建立中断服务函数，这个函数同时操作了收发配置，需要将其拆分，故将FLEXIO_UART_TransferCreateHandle函数拆分成FLEXIO_UART_TransferCreateTXHandle和FLEXIO_UART_TransferCreateRXHandle**
- **FLEXIO_UART_TransferCreateTXHandle函数调用FLEXIO_RegisterTXHandleIRQ(base, handle, FLEXIO_UART_TransferTXHandleIRQ)**
- **FLEXIO_UART_TransferCreateRXHandle函数调用FLEXIO_RegisterRXHandleIRQ(base, handle, FLEXIO_UART_TransferRXHandleIRQ)**
- **这样收发就有不同的中断服务函数了**

FLEXIO_RegisterHandleIRQ

- 原来SDK中FLEXIO_RegisterHandleIRQ建立的中断程序，收发都在一个程序里，现在需要将其拆分为两个函数FLEXIO_RegisterTXHandleIRQ(base, handle, FLEXIO_UART_TransferTXHandleIRQ)和FLEXIO_RegisterRXHandleIRQ(base, handle, FLEXIO_UART_TransferRXHandleIRQ)

```
status_t FLEXIO_RegisterTXHandleIRQ(void *base, void *handle, flexio_isr_t isr)
{
    assert(base);
    assert(handle);
    assert(isr);
        s_flexioType[0]   = base;
        s_flexioHandle[0] = handle;
        s_flexioIsr[0]    = isr;
            s_flexioType[1]   = NULL;
        s_flexioHandle[1] = NULL;
        s_flexioIsr[1]    = NULL;
    return kStatus_Success;
```

```
status_t FLEXIO_RegisterRXHandleIRQ(void *base, void *handle, flexio_isr_t isr)
{
    assert(base);
    assert(handle);
    assert(isr);
            s_flexioType[0]   = NULL;
        s_flexioHandle[0] = NULL;
        s_flexioIsr[0]    = NULL;
        s_flexioType[1]   = base;
        s_flexioHandle[1] = handle;
        s_flexioIsr[1]    = isr;
    return kStatus_Success;
```

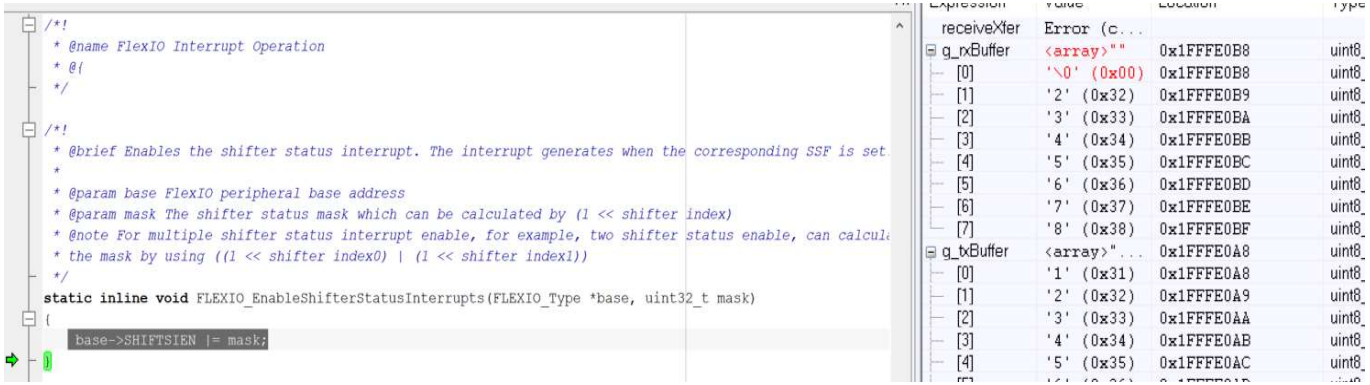
注意事项

- 在进行收发切换时需要一定的延时时间，否则收发数据会有异常

```
}

if((FlexIORunFlag == 1) && (txBufferFull == false))
{
//      FLEXIO_UART_Deinit(&uartDev);
  rxBufferEmpty = true;
  FlexIORunFlag = 0;
  FLEXIO_UART_CTRL_Init(&uartDev, &config);    //初始化控制寄存器
}

if(OverRunErrFlag == true)
{
  FLEXIO_UART_Deinit(&uartDev);
  rxOnGoing = false;
  rxBufferEmpty = true;
  FLEXIO_UART_CTRL_Init(&uartDev, &config);    //初始化控制寄存器
  OverRunErrFlag = false;
}
/*
EnableInterrupts;
*/
for(i=0;i<1000;i++)
  asm("nop");
```

有待解决的问题

- **在使能接收时，调用FLEXIO_EnableShifterStatusInterrupts(base->flexioBase, 1U << base->shifterIndex[1])函数，执行完base->SHIFTSIEN |= mask语句口，接收缓冲区的第一个字节会被清零**



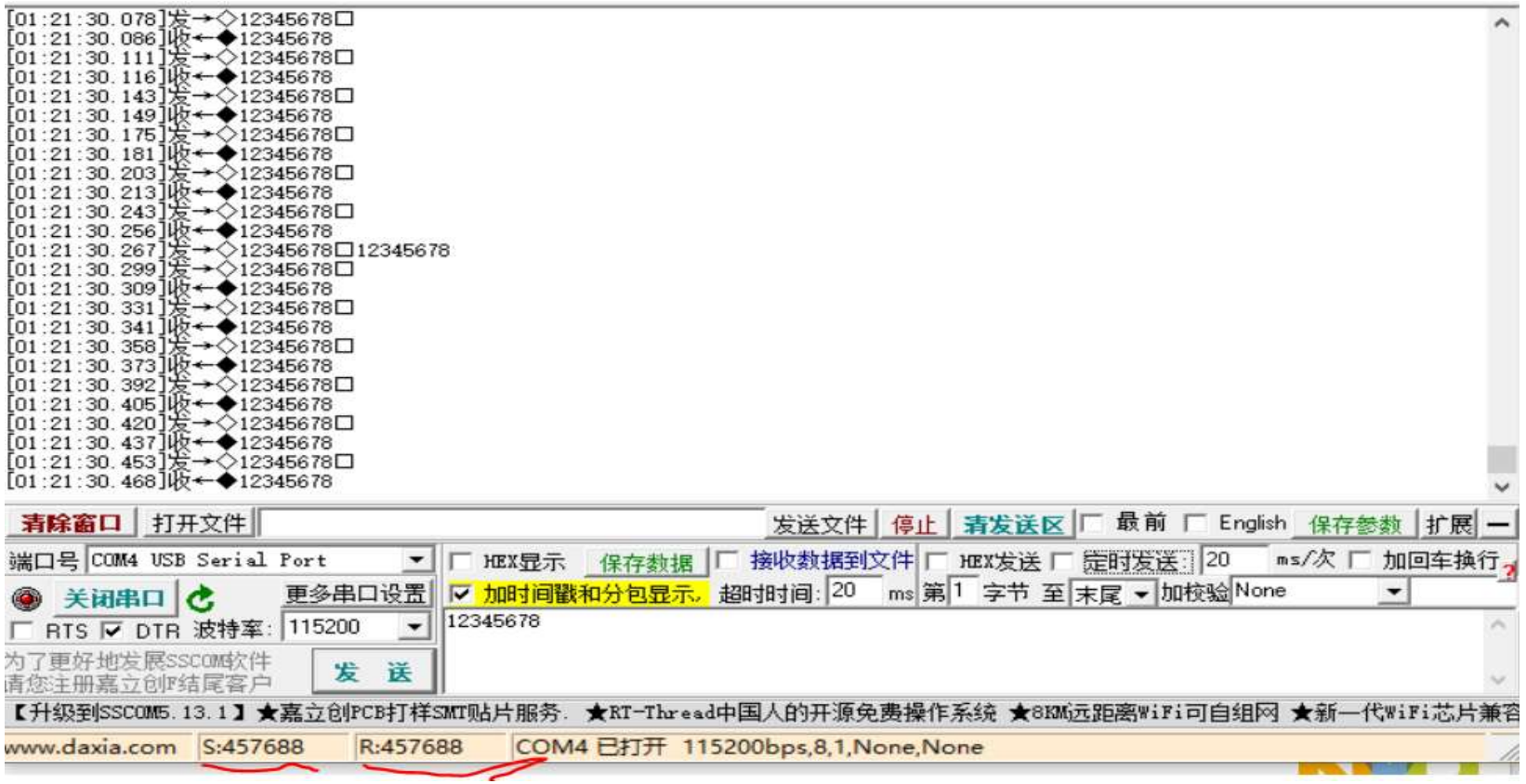
```
/*!
 * @name FlexIO Interrupt Operation
 * @{
 */

/*!
 * @brief Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set
 *
 * @param base FlexIO peripheral base address
 * @param mask The shifter status mask which can be calculated by (1 << shifter index)
 * @note For multiple shifter status interrupt enable, for example, two shifter status enable, can calcul
 * the mask by using ((1 << shifter index0) | (1 << shifter index1))
 */
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
{
    base->SHIFTSIEN |= mask;
}
```

Expression	Value	Location	Type
receiveXfer	Error (c...		
g_rxBuffer	<array>""	0x1FFFE0B8	uint8_
[0]	'\0' (0x00)	0x1FFFE0B8	uint8_
[1]	'2' (0x32)	0x1FFFE0B9	uint8_
[2]	'3' (0x33)	0x1FFFE0BA	uint8_
[3]	'4' (0x34)	0x1FFFE0BB	uint8_
[4]	'5' (0x35)	0x1FFFE0BC	uint8_
[5]	'6' (0x36)	0x1FFFE0BD	uint8_
[6]	'7' (0x37)	0x1FFFE0BE	uint8_
[7]	'8' (0x38)	0x1FFFE0BF	uint8_
g_txBuffer	<array>"...	0x1FFFE0A8	uint8_
[0]	'1' (0x31)	0x1FFFE0A8	uint8_
[1]	'2' (0x32)	0x1FFFE0A9	uint8_
[2]	'3' (0x33)	0x1FFFE0AA	uint8_
[3]	'4' (0x34)	0x1FFFE0AB	uint8_
[4]	'5' (0x35)	0x1FFFE0AC	uint8_

- **每次接收设定字节数的字符串，都会发生溢出**

执行结果

- **按20ms发送8字节来做压力测试，测试收发正常**



FF1

相关文件

- **将fsl_flexion_uart.c和fsl_flexion_uart.h替换SDK->driver中相应的文件，其它文件复制到项目目录下**

FlexioForMultiUart_HalfDuplex.zip

Slide 14

FF1 Fred Fu, 2020/8/6

SECURE CONNECTIONS
FOR A SMARTER WORLD