13 - Olá Mundo (UART)

Rafael Corsi - rafael.corsi@insper.edu.br

Abril - 2017

Figure 1: Camelos

Introdução

O código exemplo *13 - UART* configura e demonstra a utilização do periférico USART1 operando em modo de comunicação serial assíncrona. Os pinos TX e RX da comunicação serial são conectados ao EDGB, chip responsável pela gravação e debug do kit de desenvolvimento SAME70-XPLD.

A comunicação USART é um dos protocolos de comunicação mais simples e utilizados entre dois dispositivos, é utilizado por exemplo no arduino para a gravação e debug (print), é utilizado por diversos equipamentos para comunicação e configuração (impressoras térmicas, máquinas de cartão de crédito, sensores de biometria, ...).

Periféricos utilizados :

- Power Management Controller (PMC)
- Universal Synchronous Asynchronous Receiver Transceiver (USART)
- PIO (PIOA, PIOB)

Código Visão geral

A seguir analisa-se as algumas partes do código.

USART1_init()

Essa função o periférico USART1 para operar em modo de comunicação serial, também é responsável por configurar os PIOs (A e B) para conectarem os pinos os pinos PB4 e PA21 no periférico USART1.

```
static void USART1_init(void){

  /* Configura USART1 Pinos */
 sysclk_enable_peripheral_clock(ID_PIOB);
 sysclk_enable_peripheral_clock(ID_PIOA);
 pio_set_peripheral(PIOB, PIO_PERIPH_D, PIO_PB4);  // RX
 pio_set_peripheral(PIOA, PIO_PERIPH_A, PIO_PA21); // TX
 MATRIX->CCFG_SYSIO |= CCFG_SYSIO_SYSIO4;

  /* Configura opcoes USART */
  const sam_usart_opt_t usart_settings = {
    .baudrate     = 115200,
    .char_length  = US_MR_CHRL_8_BIT,
    .parity_type  = US_MR_PAR_NO,
    .stop_bits    = US_MR_NBSTOP_1_BIT    ,
    .channel_mode = US_MR_CHMODE_NORMAL
  };

  /* Ativa Clock periferico USART0 */
  sysclk_enable_peripheral_clock(USART_COM_ID);

  /* Configura USART para operar em modo RS232 */
  usart_init_rs232(USART_COM, &usart_settings, sysclk_get_peripheral_hz());

  /* Enable the receiver and transmitter. */
    usart_enable_tx(USART_COM);
    usart_enable_rx(USART_COM);
 }
```

uint32_t usart_puts(uint8_t *pstring){}

Prototipação da função a ser implementada, a função usart_puts() recebe como parâmetro um vetor de char (*pstring) e deve enviar um a um os valores desse vetor pela porta serial. O término do envio é feito quando encontrado no vetor o carácter NULL (0x00).

O envio pela serial ocorre normalmente a uma taxa de bits por segundo previamente definida, é necessário verificarmos se a transmissão já foi concluída antes do envio do próxima byte.

Essa função possui grande analogia com a função puts do C.

```
The C library function int puts(const char *str) writes a string to stdout
up to but not including the null character.
A newline character is appended to the output.
```

- return : Retorna a quantidade de char escrito.

uint32_t usart_gets(uint8_t *pstring){}

Prototipação da função a ser implementada, a função usart_gets() recebe como parâmetro um ponteiro de um vetor (*pstring) de char onde deve salvar a string a ser recebida pela porta serial. A recepção deve ser finalizada quando na porta serial ser encontrado o carácter **‘ n’**.

Essa função possui grande analogia com a função gets.

```
The C library function char *gets(char *str) reads a line from stdin and
stores it into the string pointed to by str. It stops when either the newline
character is read or when the end-of-file is reached, whichever comes first.
```

- return : Retorna a quantidade de char lido.

USART funções

Funções úteis :

- inline int usart_serial_putchar(usart_if p_usart, const uint8_t c)
  - Envia pela serial um único carácter
- uint32_t uart_is_tx_empty(Uart *p_uart)
  - verifica se a transmissão já foi finalizada, retorna:
  - 1 : não finalizada
  - 0 : finalizada
- inline void usart_serial_getchar(usart_if p_usart, uint8_t *data)
  - recebe da serial um único carácter
  - função blocante.

Função blocante ?

Funções blocantes são aquelas que travam a execução do código se uma condição não for aceita, no caso da função usart_serial_getchar() a mesma bloqueará a execução do código caso não chegue nenhum dado pela interface serial. Veja a implementação da mesma :

```
static inline void usart_serial_getchar(usart_if p_usart, uint8_t *data){
        while (uart_read((Uart*)p_usart, data));
}
```

A implementação verifica se existe um dado de recepção na interface USART, caso contrário fica travado no while(1) enquanto nenhum dado estiver disponível.

```
/**
 * \brief Read from USART Receive Holding Register.
 *
 * \note Before reading user should check if rx is ready.
 *
 * \param p_usart Pointer to a USART instance.
 * \param c Pointer where the one-byte received data will be stored.
 *
 * \retval 0 on success.
 * \retval 1 if no data is available or errors.
 */
uint32_t usart_read(Usart *p_usart, uint32_t *c)
{
    if (!(p_usart->US_CSR & US_CSR_RXRDY)) {
        return 1;
    }

    /* Read character */
    *c = p_usart->US_RHR & US_RHR_RXCHR_Msk;

    return 0;
}
```

inline ?

inline é um pragama do GCC para indicar que a função deve ser tratada como um macro e não como uma função. A diferença entre o macro e uma função é que não existe jumpers, o macro é substituto em tempo de compilação e executado em sequência. O inline é utilizado para chamadas que devem ser executadas rápidas porém possuem o contra efeito de aumentarem o tamanho ocupado pelo código, já que para cada chamada de função inline o compilador irá criar uma nova cópia dessa função.

Para maiores detalhes : (Inline GCC)[https://gcc.gnu.org/onlinedocs/gcc/Inline.html]

static ?

Funções que só são visíveis para outras funções no mesmo arquivo. Static força a função a não ser do tipo **extern**, ou seja, visível para todo o projeto.

hPara maiores detalhes : (Static) [ttp://codingfreak.blogspot.com/2010/06/static-functions-in-c.html]

Interrupção

Podemos ativar no periférico USART para que o mesmo gere interrupções para diversos eventos, sendo alguns deles :

- RXRDY: RXRDY Interrupt Enable
- TXRDY: TXRDY Interrupt Enable
- OVRE: Overrun Error Interrupt Enable
- FRAME: Framing Error Interrupt Enable
- PARE: Parity Error Interrupt Enable
- TIMEOUT: Time-out Interrupt Enable
- TXEMPTY: TXEMPTY Interrupt Enable

Dentre os eventos de interesse, podemos destacar dois : **RXRDY** e **TXRDY** pois indicam respectivamente que um dado está disponível para a leitura e que o transmissor está disponível para o envio de um novo dado.

Para ativarmos a interrupção de recepção basta fazermos a seguinte chamada de função :

```
  usart_enable_interrupt(USART_COM, US_IER_RXRDY);
  NVIC_EnableIRQ(USART_COM_ID);
```

Onde confiramos o USART para gerar uma interrupção e o NVIC para aceitar as interrupções geradas por esse periférico.

A interrupção deve ser tratada no handler do USART :

```
void USART1_Handler(void){
  uint32_t ret = usart_get_status(USART_COM);

  // Verifica por qual motivo entrou na interrupçcao
  if(ret & US_IER_RXRDY){                    // Dado disponível para leitura

  } else if(ret & US_IER_TXRDY){             // Transmissão finalizada

  }
}
```

Desafio

O desafio da aula é implementar as duas funções usart_puts() e usart_gets() já declaradas no main.c. Uma vez implementado as duas funções deve-se utilizar a interrupção de recebimento de dado para tornar a função usart_gets() não blocante.

Passos de execução

1. Entender o código exemplo
2. Entender o que está sendo pedido
3. Implementar a função usart_puts()
4. Implementar a função usart_gets()
5. Implementar a recepção de dados (usart_gets(())) via interrupção
6. Criar uma flag para indicar que o buffer (5.) está montado e pode ser tratado