Insper

12 - Handout - Olá Mundo (UART)

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

Abril - 2018

Figure 1: Camelos

Introdução

O código exemplo *12 - RTOS-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 utilizados entre dois para a comunicação entre dispositivos (Machine-to-Machine m2m), 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 nesse exemplo:

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

Objetivo do firmware

A versão final desse firmware permitirá o controle de um LED do uC (poderia ser um motor, uma bomba, ...) pelo o envio e recebimento de dados via o protocolo UART. O usuário poderá ainda controlar o LED utilizando o botão da placa.

Nesse código devemos fazer uso de um RTOS (freeRtos) tanto para o controle do LED quanto para gerenciar a comunicação com o PC (envio e recebimento de dados). Para a troca de dados entre as tarefas de comunicação e a de controle usaremos de diversas *queues*.

Código Exemplo

A seguir analisa-se as algumas partes do código exemplo que já será fornecido, nesse código, demonstra-se o uso de uma queue para o controle do LED. Duas tarefas são usadas para isso : task_but que recebe o sinal da interrupção e cria uma mensagem com a frequência para a task_led que gerencia o piscar do LED.

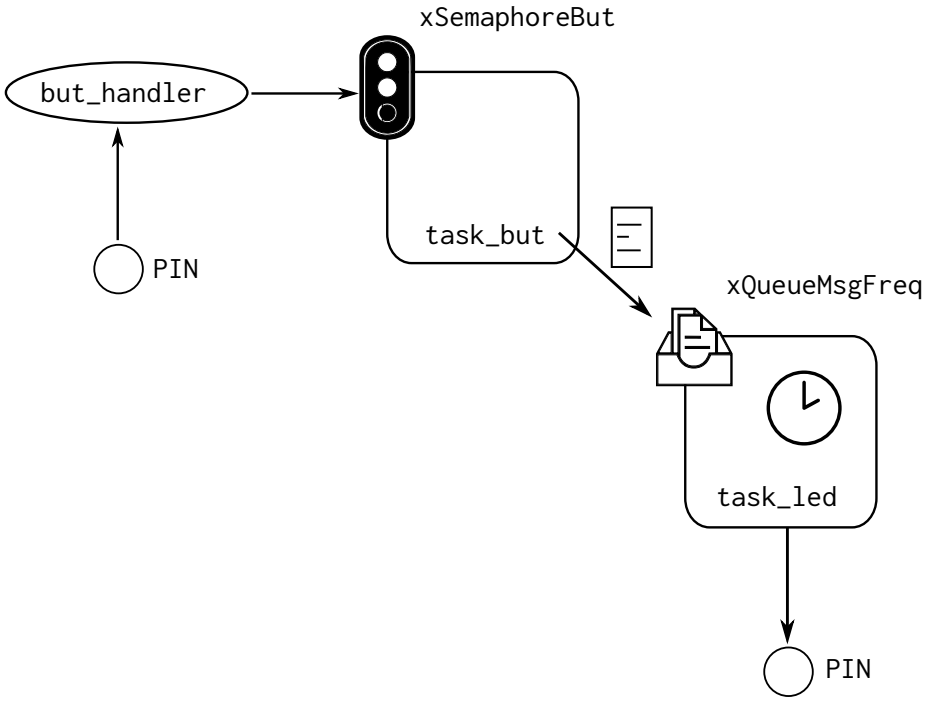

Figure 2: Diagrama

USART1_init()

Essa função configura periférico USART1 para operar em modo de comunicação serial, também é responsável por configurar os pinos TX e RX do periférico USART para acessar os pinos do microcontrolador (pinos PB4 e PA21)

but_task

Tarefa do RTOS responsável por receber os sinais de interrupção do botão e criar uma mensagem para a tarefa task_led. Essa tarefa faz uso de um semáforo para recebimento da informação que o botão foi pressionado, a informação é enviada via um queue da seguinte maneira :

```
/* envia nova frequencia para a task_led */
xQueueSend(xQueueLedFreq, (void *) &delayTicks, 10);
```

Essa funçao do RTOS adiciona na fila *xQueueLedFreq* (fim) o valor de delayTicks com um timeout de 10 ticks.

> **!** De uma olhada nas possíveis funções de manipulação de fila do FreeRtos : - https://www.freertos.org/a00018.html

led_task

Essa tarefa é responsável por piscar o LED da placa, o led inicializa com uma frequência pré determinada e é alterada quando uma nova mensagem chega pela queue *xQueueLedFreq*, quando chega um novo dado a variável responsável por determinar a frequência do LED é alterada :

```
if( xQueueReceive(xQueueLedFreq, &msg, ( TickType_t ) 0 )){
        /* chegou novo valor, atualiza delay ! */
        /* aqui eu poderia verificar se msg faz sentido (se esta no range certo) */
        /* converte ms -> ticks */
        delayMs = msg / portTICK_PERIOD_MS;
        printf("delay = %d \n", delayMs );
}
```

A função *xQueueReceive()* recebe como primeiro parâmetro a fila (queue) a ser gerenciada, o segundo é o endereço de retorno da mensagem (que será copiado) e o último é o TimeOut que nesse caso é 0 (se não tem mensagem retorna da função).

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 (baudrate), a cada novo envio 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, porém agora escrevendo a string na USART e não no terminal.

```
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.

> Essa função deve ser implementada.

Funções úteis do periférico USART:

Envia pela serial um único char :

```
inline int usart_serial_putchar(usart_if p_usart, const uint8_t c)
```

Verifica se a transmissão já foi finalizada :

```
uint32_t uart_is_tx_empty(Uart *p_uart)
```

Retorna :

- 1 : não finalizada
- 0 : finalizada

Recebe um único char da serial :

```
inline void usart_serial_getchar(usart_if p_usart, uint8_t *data)
```

- recebe da serial um único carácter
- função blocante.

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.

Para 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);
NVIC_SetPriority(USART_COM_ID, 5);
```

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
```

```
  }
}
```

Aqui devemos criar uma queue para envio dos dados que chegam da USART para alguma task que irá criar uma string.