Prof.: Paulo D. G. da Luz

ATIVIDADE PRÁTICA 6 – SETUP, UTILIZAÇÃO DO FREERTOS E PROTOCOLOS

Contexto:

O *setup* do **FreeRTOS** para uma determinada plataforma é dependente de ajustes na IDE e no próprio sistema operacional. Muitas vezes a IDE disponibiliza algumas versões pré-configuradas e totalmente funcionais para suas as arquiteturas suportadas. Porém, é importante saber fazer um *setup* para uma determinada arquitetura, assim entendemos um pouco mais como é o funcionamento do RTOS. Neste laboratório, portanto, iremos executar um passo a passo para configurar um novo projeto utilizando o Kit TIVA **EK-TM4C1294XL** e o **FreeRTOS** no IDE **Keil**.

Após a configuração do kit e do sistema operacional, teremos como objetivo criar no mínimo algumas tarefas: 1) *GateKeeper* da UART, 2) Leitor do Sensor1, 3) Leitor do Sensor2 e 4) Atuador dos "Relés". Além destas tarefas, precisamos criar 3 filas de mensagens: 1) *GateKeeper* ↔ Sensor1, 2) *GateKeeper* ↔ Sensor2, 3) *GateKeeper* ↔ Atuador dos "Relés".

Também neste laboratório, iremos expandir o conceito de utilização de Protocolos, adicionando ao protocolo de comunicação o conceito de CHECKSUM. Este item é de extrema importância para validar a integridade de um pacote. Entender um recurso muito importante da linguagem "C" o *UNION*, que possui pouca referência aliado à sua utilização com protocolos. Lembrando que ainda em protocolos avançadas podemos encontrar os conceitos de C.R.C e criptografia.

Obs: Este laboratório deve ser feito após o minicurso de FreeRTOS.

1. A tarefa 1: *GateKeeper* será responsável por gerenciar o periférico da UART e por decodificar o protocolo de comunicação entre o Kit e o PC. Após receber um pacote válido e decodificá-lo, o *GateKeeper* irá se comunicar com a tarefa responsável por tratar os dados, aguardando uma mensagem de retorno de conclusão da execução do último comando. Quando este retorno ocorrer o *GateKeeper* monta o protocolo de retorno e devolve o pacote ao PC.
2. A tarefa 2: Nesta tarefa iremos simular uma leitura de algum sensor analógico, como exemplo: um sensor de temperatura. Nesta simulação iremos receber um valor *float* do PC e executaremos a operação:

$$Temperatura = (float)\left(Recebido \times \frac{7}{3} \times \sqrt{5}\right)$$

O resultado Temperatura será obviamente do tipo ***float*** e será devolvido ao *GateKeeper*, e o *GateKeeper* devolve o valor ao PC.

3 A tarefa 3: Nesta tarefa iremos controlar o brilho do LED através do controle por **PWM**, como foi executado no Laboratório 5. A diferença aqui é que iremos receber do *GateKeeper* o valor inteiro do tipo (***unsigned short int***) de 0 a 100 ... onde 0 = 0% de brilho e 100 = 100% de brilho. Após receber o valor do brilho, atuar no periférico do PWM e devolver o *status* do "PWM" para o *GeteKeeper*, e o *GateKeeper* devolve o status ao PC.

4 A tarefa 4: Utilizar os outros 3 LEDs do kit para simular os "Relés" como foi feito no Laboratório 5. Assim como nas duas tarefas anteriores, atualizar os acionamentos dos "Relés" e devolver o status ao *GateKeeper* e este devolver o status ao PC.

5 A utilização dos *Shields* é opcional para este laboratório, mas ajuda no desenvolvimento tanto para *debug* visual e interação com o código.

6 ... podem ser utilizadas estruturas adicionais ou até mais tarefas para montar a solução.

Dicas:

Executar o tutorial: http://www.elf74.daeln.com.br/Pdfs/Tutorial_Keil_FreeRTOS_TivaWare_Projeto.pdf

O novo protocolo desenvolvido para esta atividade está descrito no arquivo: **Protocolo_Lab6.xlsx** - http://www.elf74.daeln.com.br/Labs/Protocolo_Lab6.xlsx. Também está disponível para *download* o programa **sscom3.2.exe**, que é um utilitário para comunicação serial e que não precisa ser instalado no Windows. Também está disponível para *download* o programa **Lab6_Serial.exe**, que é um programa para comunicação serial que implementa o protocolo descrito no **PDF** e que não precisa ser instalado no Windows.

Com relação ao uso do **UNION**, segue alguns trechos de código para facilitar o entendimento e seu uso aplicado a protocolos. O *union* em linguagem C serve para acessar / compartilhar uma mesma região de memória RAM por diferentes tipos de dados. Este recurso se torna extremamente útil para ser utilizado em protocolos, pois permite a conversão instantânea entre tipos sem necessidade de códigos matemáticos ou algoritmos para conversão entre tipos ...

```
union usword2halfword2byte {
uint32_t word;                //word = 32bits
uint16_t hword[2];            //half word = 16bits
uint8_t byte[4];              //byte = 8bits
} w_dh_b;                     //compartilhamento de memória RAM
// 315774220 = 0x12D2550C

w_dh_b.word = 315774220;      //acessando via 32bits a RAM compartilhada

w_dh_b.hword[0]=0x550C;       //acessando via 16bits a RAM compartilhada
w_dh_b.hword[1]=0x12D2;       //high half word

w_dh_b.byte[0]=0x0C;          //acessando via 8bits a RAM compartilhada
w_dh_b.byte[1]=0x55;          //b[1]
w_dh_b.byte[2]=0xD2;          //b[2]
w_dh_b.byte[3]=0x12;          //b[3] = high byte
```

O trecho acima ilustra o acesso a uma região de memória RAM de 32bits, acessando via uma variável de 32bits ou via duas variáveis de 16bits ou por 4 variáveis de 8bits. Portanto quando se armazena um valor via **.word(32bits)** da *union*, já existe a conversão direta para 16bits e 8bits sem a necessidade de nenhum algoritmo de conversão de tipos. Resumidamente, caso se deseje enviar via comunicação serial o valor de 32bits (315774220), basta utilizar o *union* exemplo acima, utilizando w_dh_b.word=315774220; e enviar na serial os 4 bytes: w_dh_b.byte[0] a w_dh_b.byte[3]. Já no receptor basta colocar os 4 bytes na *union* em e depois acessar o valor no **w_dh_b.word**.

CheckSum

Checksum (*unsigned short int*) =soma do byte[0] ao byte[10] do protocolo.

$$CheckSum = \sum_{i=0}^{10} bufferp[i];$$

```
union usint2byte {
unsigned short int I;
unsigned char B[2];
} checksum;

for (j=0,checksum.I=0;j<10;j++) checksum.I+=bufferp[j];
bufferp[11]=checksum.B[0];
bufferp[12]=checksum.B[1];
```

Além do ***union*** *CheckSum*, por analogia, criar mais duas definições de ***union***: 1) **usfloat2byte** e 2) **usuint16_t2byte**. Uma para ser utilizada com o protocolo do sensor de temperatura e a outra para ser utilizada com o protocolo de PWM.

Conteúdo ao arquivo (readme.txt):

Disciplina: ELF74
Laboratório: 6
Equipe: Nome do aluno 1, RA
Nome do aluno 2, RA
Data: XX/YY/ZZZZ

Configuração de Diretórios:

Manter os projetos de todos os laboratórios no mesmo nível de diretório que o diretório da "*TivaWare*", também manter o nome do diretório original: **"TivaWare_C_Series-2.2.0.295"**. Isto serve para que consiga compilar os exemplos do professor e vice-versa, quando entregar os laboratórios para avaliação.

Para entregar o Laboratório enviar um *email* para o professor com o diretório do projeto compactado em um arquivo .zip.

Email: garcez@professores.utfpr.edu.br

Título: **ELF74 – Laboratório 6**

Corpo do email:

Equipe: Aluno1, RA
Aluno2, RA

Anexo: **lab6.zip**

***** Este laboratório deverá ser mostrado em sala de aula em funcionamento junto ao professor para ser validada a nota. *****