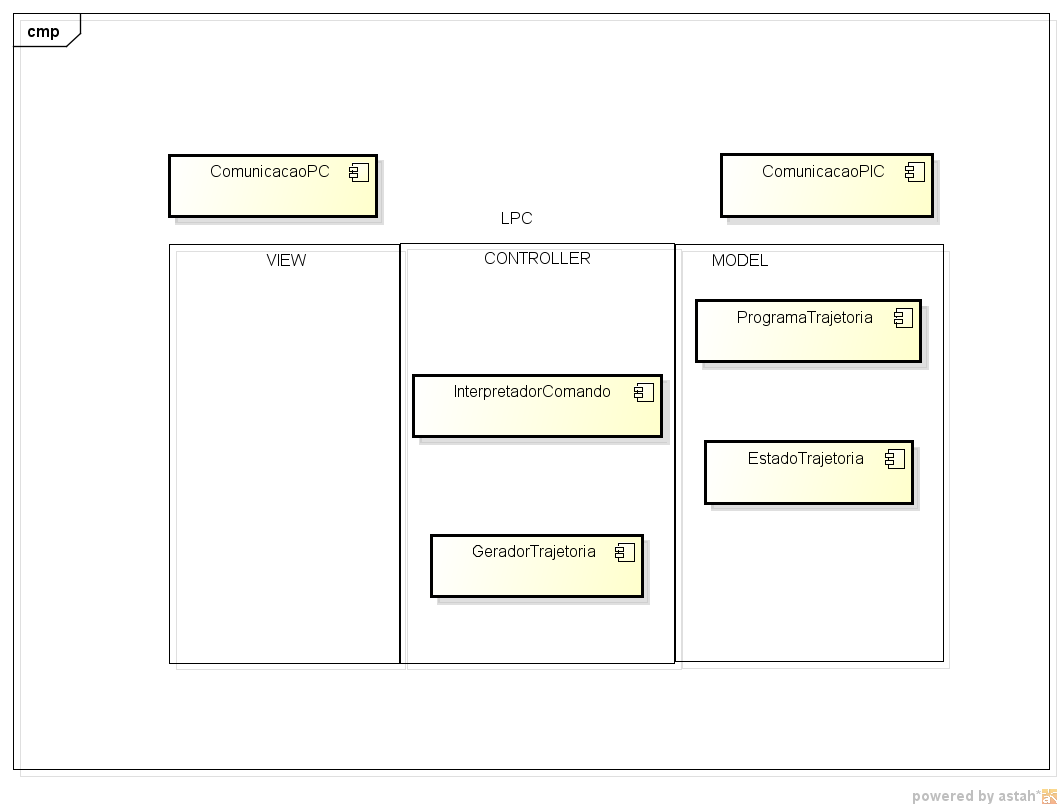
**Controlador de Trajetória**

Notas de implementação

# 1.Design

A fig.1 mostra o diagrama de componentes determinado para o software do LPCxpress.



*Fig.1:Diagrama de componentes para LPCxpresso*

Pode-se perceber a divisão funcional em:

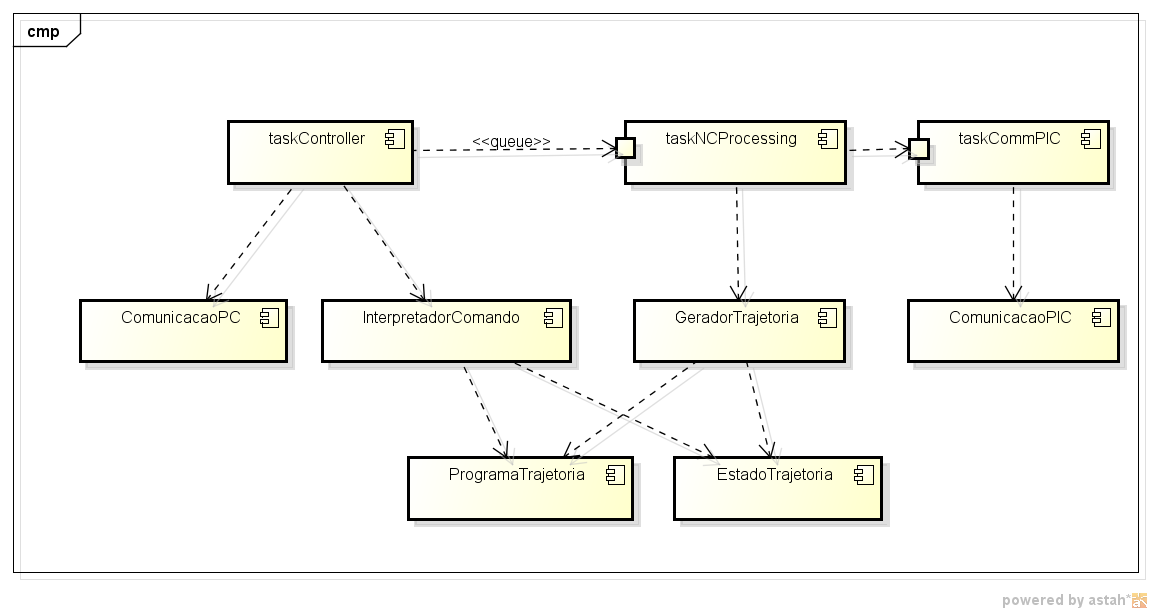
* ComunicacaoPC: responsável pela implementação do protocolo MODBUS, entre PC e LPCxpresso;
* InterpretadorComando: responsável pela execução de comandos recebidos do PC;
* GeradorTrajetoria: responsável pela geração da trajetória propriamente dita, incluindo a conversão de coordenadas “de garra”, recebidas do PC, para coordenadas de junta, a serem enviadas ao PIC. Inclui também a eventual interpolação entre pontos.
* ProgramaTrajetoria: armazena o programa a ser executado;
* EstadoTrajetoria: armazena o estado da trajetória, como as coordenadas atuais da máquina.
* ComunicacaoPIC: responsável pelo envio de *setpoints* para os PICs, de acordo com o protoloco LPCxpresso-PIC (ver notas de implementação do PIC, aonde este protocolo está descrito).

O detalhamento das chamadas entre estes módulos já foi apresentado em sala, quando foi discutida a arquitetura MVC (ver slides da aula sobre MVC).

Note-se, entretanto, que várias atividades devem ser executadas “em paralelo”, como por exemplo:

* A interpolação da trajetória (GeradorTrajetoria) não deve impedir que novos comandos sejam recebidos e processados (ComunicacaoPC, InterpretadorComando);
* O envio de *setpoints* aos PICs (ComunicacaoPIC) não pode impedir que novos comandos sejam recebidos e processados (ComunicacaoPC, InterpretadorComando) nem que sejam gerados os próximos *setpoints* (GeradorTrajetoria).

Assim, decidiu-se por uma implementação em que 3 Tasks (como os define o FreeRTOS) serão encarregados do processamento, comunicando-se através de Queues (como definidas no FreeRTOS). A fig.2 mostra esta decomposição. Note-se o uso do estereótipo <<task>>, que identifica o componente como uma Task.

**

*Fig2: Arquitetura de execução*

Comumente, chama-se este mapeamento de funções a tasks/processos de “arquitetura de execução”. Os Tasks são:

* taskController: realiza a comunicação MODBUS com o PC (ComunicacaoPC) e invoca o InterpretadorComando. Para certos comandos (como o que inicia a execução do Programa NC), envia informações através da fila *qControlCommands* ao *taskNCProcessing*.
* taskNCProcessing: responsável pelo processamento do Programa NC (GeradorTrajetoria).
* taskCommPIC: responsável pela comunicação com o PIC (ComunicacaoPIC).

# 2.Implementação

A implementação fornecida é parcial e precisará ser completada. Os principais pontos a serem completados estão indicados a seguir.

|  |  |
| --- | --- |
| Módulo | Implementar |
| modbus.c | processWriteFile: implementar |
|  | processReadRegister: o registrador possui apenas 1 byte; alterar a implementação ser for necessário um outro tipo de dado. |
|  | processWriteRegister: o registrador possui apenas 1 byte; alterar a implementação se for necessário outro tipo de dado. |
| Interpretador\_comando.c | ctl\_WriteRegister: apenas o comando CMD\_START está implementado mas pode necessitar de alterações. |
|  | ctl\_ReadRegister: retorna um inteiro; alterar a implementação se for necessário outro tipo de dado |
|  | ctl\_WriteFile: implementar |
| comunicacao\_pic.c | pic\_sendToPIC: implementar. O protocolo de comunicação LPCxpresso-PIC foi discutido no material relativo à implementação do PIC. |
| controlador\_trajetoria.c | trj\_generateSetpoint: rever a implementação; a que foi implementada até pode servir. |
| programa\_trajetoria.c | ptj\_storeProgram: implementar |

Note também que os drivers dos dispositivos (UART, console, PWM, timer) foi movido para o diretório “drivers”, na árvore de projeto.

# 3.Configuração

Os principais aspectos de configuração que podem precisar ser alterados são indicados a seguir.

|  |  |
| --- | --- |
| Parâmetro | Configuração |
| BAUD | Define a taxa de comunicação entre o PC e o LCPxpresso. Encontra-se no arquivo modbus.c. |
| MODE | Modo de operação da comunicação MODBUS, podendo ser DEVELOPMENT\_MODE ou REAL\_MODE. Pode ser alterado na função com\_init().  \_mode = DEVELOPMENT\_MODE;  Em DEVELOPMENT\_MODE, os caracteres são lidos da fila(queue) qCommDev, visando simular a comunicação com o PC. E os caracteres são escritos na console, ao invés de serem enviados à UART0. Ver, no programa principal (main.c) como inserir informações nesta fila, para simular a comunicação sem o uso do PC.  Em REAL\_MODE, a comunicação é através da UART0. |
| MAX\_RX\_SIZE | Tamanho do buffer de recepção. Pode ser necessário alterar seu tamanho para permitir armazenar o programa em código G. Encontra-se no arquivo modbus.c |

# 4.Teste sem conexão ao PC

Para testar o seu programa, pode ser útil usar MODE=DEVELOPMENT\_MODE (ver item 3.Configuração). Neste modo, os caracteres são lidos da fila (queue) qCommDev, ao invés de o serem da UART0. Desta forma, você pode testar o programa sem precisar estar conectado ao PC. Para fazer isso, crie a mensagem que seria enviada pelo PC como, por exemplo:

// test data for ReadRegister: read reg 1 (CoordY)

uint8\_t msgReadRegister[] = {0x3a, 0x00, 0x00, 0x30, 0x33,   
 0x00, 0x00, 0x30, 0x31, 0x33, 0x3c, 0x0d, 0x0a};

Este exemplo é de uma mensagem de ReadRegister (FunctionCode 03) para leitura do registrador 01. Notar que, em DEVELOPMENT\_MODE, não há verificação de LRC.

No programa principal, mande os caracteres para a fila qCommDev, como no exemplo:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEBUG FOR ReadRegister

// insert ReadRegister msg on qCommDev for debug

for (i=0; i<sizeof(msgReadRegister); i++) {

ch = msgReadRegister[i];

xQueueSend(qCommDev, &ch, portMAX\_DELAY);

}

Notar que esta fila recebe um **caracter por vez**. Assim, para testar o envio do programa em código G, pode ser necessário aumentar o tamanho desta fila, alterando o valor de DEV\_Q\_SIZE, no arquivo main.c.