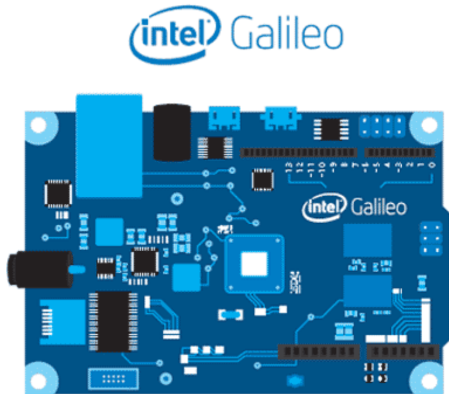
Revisão para programação no Galilleo (linguagem C)



Objetivos

- Resumo sobre os comandos para programação no Galileo
  - Utilização de pinos digitais, analógicos e PWM
  - Utilização da biblioteca PahoMQTT
  - Configurações gerais para Galileo
- A explicação dos códigos será feita utilizando-se linguagem C

Leitura e escrita em GPIO(*General purpose Input Output*)

- Para utilizar-se um pino de entrada/saída é necessário alocar uma instância, referenciando o número do pino desejado:
  - **mraa_gpio_context** pino_1; - Inicializa uma variável do tipo contexto, que será utilizado como referência para um pino digital. (pino_1 é o nome da variável exemplo)
  - **mraa_gpio_init(NUMERO_PINO);** - Retorna uma referência para uma variável do tipo gpio_context, referenciando o numero do pino desejado.
  - **mraa_gpio_dir(REF_PINO, STATE);** Onde:
    - REF_PINO é a referência para uma variável do tipo gpio_context(já deve estar inicializada);
    - STATE é um 'enum' que define o propósito do pino. Para *input* – MRAA_GPIO_IN, para *output* MRAA_GPIO_OUT
- Com os comandos anteriores já é possível efetuar leitura ou escrita nos pinos alocados.
  - **mraa_gpio_write(CONTEXT_PIN, VALUE);**Onde:
    - CONTEXT_PIN é a variável referente ao pino alocado; VALUE refere-se aos valores 1 (+5V) e 0 (GND).
  - **mraa_gpio_read(CONTEXT_PIN)**
    - CONTEXT_PIN é a variável referente ao pino alocado;
- Referência
  - http://iotdk.intel.com/docs/master/mraa/gpio_8h.html#a9d0e44ca57ac07619f237ffc50d6021c

Exemplo de código para leitura e escrita em GPIO

```
#define HIGH 1  //+5V
#define LOW 0   //GND


// Inicializa referencias para GPIO
mraa_gpio_context sensor = mraa_gpio_init(10); //(Pino 10)
mraa_gpio_context atuador = mraa_gpio_init(11);//(Pino 10)

// Atribui a funcao do GPIO
mraa_gpio_dir(sensor,MRAA_GPIO_IN);  //Input pin
mraa_gpio_dir(atuador,MRAA_GPIO_OUT);//Output pin

// Efetua a leitura do pino digital 'sensor'
int leitura_sensor = mraa_gpio_read(sensor);

// verificacao do estado do pino digital
if(leitura_sensor){
    // Ativa o atuador (dependendo da topologia)
    mraa_gpio_write(atuador,HIGH);
}
else
    // Desativa o atuador (dependendo da topologia)
    mraa_gpio_write(atuador,LOW);
```

Leitura e escrita em AIO (*Analog Input Output*)

- A utilização das entradas analógicas é semelhante aos pinos de propósito geral. A grande diferença é que em portas analógicas são exclusivas para leitura de dados:
- O ADC (Analog to Digital Converter) do Galileo possui uma resolução de 10bits, com tensão de referência padrão de +5V. Portanto, ao receber um sinal de +5V o valor lido será de 1023.
- Os comandos mais utilizados são:
  - **mraa_aio_context** pino_1; - Inicializa uma variável do tipo contexto, que será utilizado como referência para um pino analógico.
  - **mraa_aio_init(NUMERO_PINO)**; - Retorna uma referência para uma variável do tipo gpio_context, referenciando o numero do pino analógico desejado.
  - **mraa_aio_read(CONTEXT_REF);** - Retorna a leitura obtida na porta analógica alocada.

Exemplo de código para leitura de dados analógicos

O código efetua a alocação de um pino analógico, obtém sua leitura e converte o valor digital para o valor da tensão lida, considerando-se a resolução do ADC e a tensão de referência.

```
// Precisão obtida por um ADC de 10 bits, com Tensão de referência em 5V
#define precisao 0.005 // Refere-se a uma precisão de 5mV

// O numero do pino alocado e referente as pinos analógicos. (0 - 5)
mraa_aio_context analogico = mraa_aio_init(0);

// O resultado obtido do ADC é um valor inteiro representando a
// amostragem de um sinal analógico com precisão de 10 bits.
// O resultado será dado entre 0 - 1023
int leitura = mraa_aio_read(analogico);

float valor_de_tensao = (float)leitura*precisao;
```

Utilizando a Biblioteca PahoMQTT

- Como visto em sala, o protocolo MQTT é um protocolo de aplicação muito leve, utilizado para transmissão de mensagens, muito aplicado para comunicações M2M *(machine to machine).*
- A estrutura básica para se conectar com um *broker* MQTT segue:
  - Configuração de *client*()
  - Credenciamento e Autenticação
  - Configuração da função para recebimento de mensagens (via *subscribe*)
  - Efetuar conexão com o *broker*
  - Estruturação da mensagem a ser enviada (padronização para json)
  - Inscrição e publicação de dados em tópicos

Criando e configurando o *client MQTT*

- Primeiramente iniciamos com a configuração do Cliente, credenciamento e autenticação.
  - **MQTTClient cliente**; - Cria uma variável referente a uma estrutura de dados para um *client.*
- Após a criação da instância para um cliente é necessário efetuar sua inicialização.
  - **int MQTTClient_create(MQTTClient* handle, char * serverURI, char * clientId, int persistence_type, void * persistence_context)**
    handle – é o endereço da variável referente ao MQTTClient anteriormente criada.
  - serverURI – é o host do servidor que disponibiliza o serviço de *broker*, e segue o padrão: tcp://DNS_ou_IP_Servidor:1883 (A porta padrão para o MQTT é 1883, mas pode variar de acordo com o servidor).
  - clientID – é o nome utilizado para referenciar o dispositivo que esta acessando o *broker*. Cada dispositivo, dentro de uma rede corporativa, deve ter um ID único.
  - persistence_type; - é um ENUM que define o tipo de persistência de dados que será utilizado. E pode ter os seguintes valores:
    - MQTT_PERSISTENCE_NONE
    - MQTT_PERSISTENCE_DEFAULT
    - MQTT_PERSISTENCE_USER

Criando e configurando o *client MQTT*

  - Persistence_context; -é uma estrutura para persistência de dados, que pode ser instanciada pelo código **MQTTClient_persistence**. Tal variável só é necessária se o tipo de persistência for configurado em modo DEFAULT ou USER, caso contrário basta configurar como parâmetro nulo (NULL).
- OBS: O retorno da função **MQTTClient_create()** é um valor inteiro que indica se a operação ocorreu com sucesso ou falha. Para verificação utiliza-se o ENUM:
  - MQTT_SUCCESS equivalente ao inteiro 0
  - MQTT_FAILURE equivalente ao inteiro -1
- A partir da estrutura de *client* devidamente instanciada é necessário criar o pacote de dados para autenticação. Tais informações serão vinculadas ao *client* a partir da conexão com o *broker*.
  - **MQTTClient_connectOptions data***;* - Tipo de dados que armazena informações sobre o criente. (Usuário e senha para o *broker*).
  - **MQTTClient_connectOptions_initializer**; - Retorna a instância para a estrutura anterior.
- A partir da nova variável basta configurar os campos da variável data.
  - data.username = “usuário”
  - data.password = “senha”
  - OBS: As strings devem ser do tipo constante.

Configurando a função *callbacks*

- Antes de efetuar a conexão com o *broker* é necessário configurar a função callbacks. Sua principal funcionalidade é instanciar uma thread para gerenciamento de mensagens de resposta vindas do *broker*, inclusive atualização dos tópicos em que o cliente está inscrito.
  - **Int MQTTClient_setCallbacks(MQTTClient handle,void* context,MQTTClient connectionLost *cl,MQTTClient_messageArrived* ma,MQTTClient_deliveryComplete *dc)**
  - handle; é a instância do cliente que está sendo configurado
  - context; é a referencia para um contexto específico da aplicação. (nos casos mais genéricos configura-se como NULL)
  - cl; é a referência para uma funçãoque habilita notificações assincronas quando há a perda de conexão com o servidor. É necessário implementa-la em código.
    - **void MQTTClient_connectioLost(void *context, char *cause){}**
  - ma; é a referência para uma função chamada quando há notificações referentes a atualizações nos tópicos em que o cliente está inscrito. Deve-se implementar a seguinte função:
    - **Int MQTTClient_messageArrived(void* context, char* topicName,int topicLen, MQTTClient_message *message)**
  - **dc;** é uma referência para uma função que permite avaliar se a mensagem foi enviada com sucesso. Caso deseja-se obter o resultado do envio das mensagens é necessário implementar a seguinte função:
    - **void MQTTClient_deliveryComplete(void *context, MQTTClient_deliveryToken dt){}**

Conexão, inscrição e publicação

- Agora que já possuímos tudo configurado basta efetuar a conexão:
  - **int MQTTClient_connect**(**MQTTClient** handle,**MQTTClient_connectOptions** * options)
  - handler; é o objeto referente ao cliente a se conectar
  - options; é o conjunto de opções para autenticação no *broker* MQTT.
  - O retorno da função é um valor inteiro que representa se a conexão foi feita com sucesso ou falha, e segue os mesmos critérios dos ENUMs anteriormente citados
    - MQTT_SUCCESS equivalente ao inteiro 0
    - MQTT_FAILURE equivalente ao inteiro -1
- Para efetuar a inscrição em um tópico MQTT, é necessário ter o nome do tópico e uma conexão previamente estabelecida com o *broker*.
  - **int MQTTClient_subscribe**(**MQTTClient** handle,char * topic,int qos)
  - handler; é o objeto referente ao cliente que receberá as atualizações do *broker*
  - topic; é o nome do tópico em que será feita a inscrição
  - Qos; Corresponde a qualidade do serviço que será utilizado. Pode ser QoS1,QoS2 e QoS3. (http://www.eclipse.org/paho/files/mqttdoc/Cclient/qos.html)
  -

Conexão, inscrição e publicação

- Para efetuar a publicação dos dados basta escrever o seguinte comando;
  - **Int MQTTClient_publish(MQTTClient handler, char* topicName, int payloadlen, void* payload, int qos, int retained, MQTTClient_deliveryToken* dt);**
  - Handler; é o objeto referente ao cliente a publicar os dados;
  - Topicname; é o nome do tópico onde os dados serão publicados;
  - payloadLen; é o tamanho máximo da mensagem que será enviada;
  - Payload; é a mensagem estruturada que será enviada. Como o tipo do dado da mensagem não é especificado, tem-se como parâmetro da função um ponteiro para void. Portanto a mensagem pode ser de qualquer tipo de dados.
  - Qos; inteiro que representa a qualidade de serviço. Segue os mesmos parâmetros do sistema *subscribe.*
  - Retained; flag que representa se os dados da mensagem serão retidos (1) no *broker* MQTT, ou serão descartados(0).
  - Dt; é a referência para a estrutura do tipo MQTTClient_deliveryToken, que atribui um valor de token para a mensagem caso ela seja publicada com sucesso. Tal valor não é utilizado na maior parte dos casos, portanto tal ferramenta é opcional, podendo ser configurada como NULL.

Exemplo de estruturação para envio de strings

- As mensagens para publicação podem ser de qualquer tipo de dados. No entanto, é comum co envio de mensagens do tipo *String* (vetor de char) para isso é interessante utilizar uma padronização, como XML ou JSON. Neste caso é necessário construir a estrutura de acordo com o padrão.
- Por exemplo, na estruturação de uma mensagem em JSON é necessário utilizar a seguinte estrutura;
  - {“chave1”:”valor1”, “chave2”:”valor2” ...}
- Em linguagem C é necessário construir tal estrutura como um vetor de caracteres estático.
  - “{\”chave1\”:\”valor1\”,\”chave2\”:\”valor2\”...}”
  - É necessário utilizar a contra barra para representação das aspas duplas.
  - A estrutura pode ter N atributos, separados por virgula.
  - Uma chave pode ter como valor uma outra estrutura do tipo JSON.
  - {“chave”:{“sub_chave”:”sub_valor”}}

Referências

- PahoMQTT:
  - http://www.eclipse.org/paho/files/mqttdoc/Cclient/_m_q_t_t_client_8h.html#abef83794d8252551ed248cde6eb845a6
- Libmraa (c/c++)
  - http://iotdk.intel.com/docs/master/mraa/index.html