Como foi visto no [artigo anterior](http://www.embarcados.com.br/configurando-o-microcontrolador-kl05z/), para ter acesso aos periféricos, configurar os registros da CPU, ou executar algumas atividades, é necessário alterar algumas posições específicas de memória através de ponteiros. Por questão de praticidade, é comum reunir um conjunto de operações ou registros em comum em uma biblioteca. Isto simplifica a utilização de um periférico ou funcionalidade do microcontrolador.

Neste artigo implementaremos a biblioteca "io.h", que será responsável por fazer o acesso aos terminais de IOs da KL05, levando em conta a numeração utilizada pelo barramento do Arduino. Isto vai simplificar o desenvolvimento do software quando os terminais físicos forem utilizados. Nesta biblioteca serão criadas basicamente três funções: **digitalRead()**, **digitalWrite()** e **PinMode()**.

Configuração dos GPIOs e as informações do Datasheet

Antes de desenvolver a biblioteca para um periférico, é importante conhecer os detalhes de funcionamento do mesmo. O datasheet é a fonte principal de informações sobre os GPIOs, funções, configurações e registros do microcontrolador. Por esse motivo esta parte do artigo será baseada nas informações fornecidas pela NXP para o KL05z.

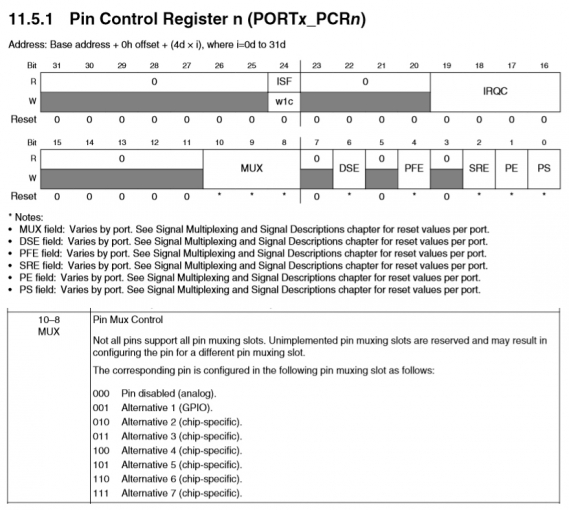
Por questões de custo, e para aumentar as possibilidades de uso de um microcontrolador, é comum uma fabricante disponibilizar mais de um periférico de modo multiplexado num mesmo terminal (pág. 145 do datasheet).

O capítulo 10 do datasheet oferece ainda uma tabela com todas as possibilidades de uso dos terminais. Por efeito de espaço e simplicidade, a tabela foi reduzida apenas para os terminais disponíveis na placa. De acordo com a alternativa/opção escolhida, o terminal exerce uma função diferente.



O módulo de**port control and interrupt (PORT)**oferece suporte para o controle das portas e funções de interrupções externas. A maioria das funções podem ser configuradas separadamente para cada terminal utilizando uma variável de 32 bits. Deve-se tomar cuidado pois nem todos os terminais estão implementados em todas as versões de microcontroladores. [PAG159].

Para escolhermos qual alternativa queremos para um determinado GPIO **n** de uma porta **x**, devemos ir no registro responsável por sua configuração, o **PORTx\_PCRn**. A tabela abaixo apresenta seus detalhes. Note que cada terminal de GPIO possui seu próprio registro de controle.



Como queremos operar o terminal como um GPIO, devemos escolher, segundo a tabela acima, a opção 1 (utilizaremos o terminal 2 da porta B como exemplo: PortB2). Para isso é necesário colocar o valor 0b100 nos bits 10, 9 e 8. Como as demais configurações do registro não são importantes neste ponto, podemos escrever o seguinte código:

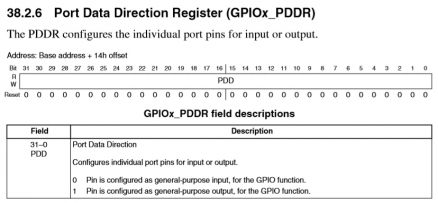
PORTB\_PCR2 = 0x100;             //PORTB2

Para evitar a utilização de [números mágicos](https://en.wikipedia.org/wiki/Magic_number_(programming)) podemos utilizar as máscaras e structs fornecidas pelo fabricante, através do MKL05Z4.h. Deste modo podemos reescrever o código acima como:  
  
PORTB\_BASE\_PTR-> PCR[2]  = PORT\_PCR\_MUX(1) ;

OBS: É necessário realizar essa configuração para todos os pinos das portas que se desejam utilizar.

O módulo do **general-purpose input and output (GPIO)** se comunica quase que diretamente com o processador. Os registros de **GPIO data direction (PDDR)** e **GPIO output data (PDOR)** controlam o sentido, entrada ou saída, e o valor de cada pino, quando configurado como saída. O registro **GPIO input data register (PDIR)** mostra o valor lógico de cada pino configurado como entrada (pág. 661).

Para definirmos então o sentido do pino, podemos manipular o registro GPIOx\_PDDR:



Cada bit nesse registro representa um pino de uma determinada porta, por exemplo: O pino 2 da porta A se encontra no registro **GPIOA\_PDDR**, no bit **2**.

Como escrito no datasheet, para configurarmos o pino 2 da porta A como output ou input devemos manipular o bit 2. Logo, os códigos responsáveis por essas operações são:

GPIOB\_PDDR |= 0x4u; // pino definido como output

GPIOB\_PDDR &=~ 0x4u;      // pino definido como input  
  
 Utilizando as máscaras e structs fornecidas, o código acima se transforma em:

PTB\_BASE\_PTR ->PDDR |= 0x4u;  
PTB\_BASE\_PTR ->PDDR &=~ 0x4u;

Havendo a possibilidade de utilização de operações de manipulação de bits, podemos reescrever o código como:

bitSet(PTB\_BASE\_PTR ->PDDR, 2); // pino definido como output

bitClr(PTB\_BASE\_PTR ->PDDR, 2); // pino definido como input

Se um pino é definido como output podemos configurar se seu valor de saída é alto (1) ou baixo (0). Para isso é necessário manipular os bits do registro **GPIOx\_PDOR.** Este registro funciona de modo similar ao GPIOx\_PDDR, onde cada bit representa um terminal e, de acordo com seu valor, o pino pode apresentar nível alto ou baixo de tensão.

Adotando novamente como exemplo o pino 2 da porta A, os códigos responsáveis por definir a lógica do pino são:

GPIOB\_PDOR |= 0x4u; // Saída 1 - nível alto

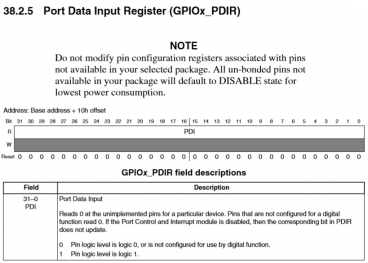
GPIOB\_PDOR &=~ 0x4u;       // Saída 0 - nível baixo

Utilizando as máscaras e structs fornecidas, o código pode ser reescrito como:

PTB\_BASE\_PTR ->PDOR |= 0x4u;

PTB\_BASE\_PTR ->PDOR &=~ 0x4u;

Se o terminal foi configurado como entrada, podemos ler os bits do registro GPIOx\_PDIR para conhecer seu estado. Nesse registro, assim como nos outros dois, cada bit representa um terminal distinto.



Para ler o bit, e utilizar corretamente sua informação, é necessário primeiro limpar todos os demais bits, para que eles não influenciem na tomada de decisão. O código responsável pela leitura os bits do registro é (adotando novamente o bit 2 da porta A) :

if (GPIOB\_PDIR & 0x4u){  
   //o terminal 2 da porta B está com nível alto  
}else{  
   //o terminal está com nível baixo  
}

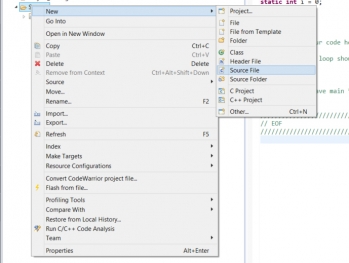
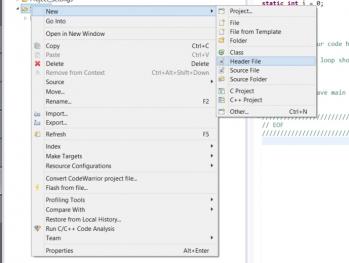
**Criando uma biblioteca em linguagem C**

Na criação de uma biblioteca em linguagem C é comum utilizamos dois arquivos com funções distintas. Toda implementação dos códigos é feita no arquivo com extensão **.c (code)**. É nele que criamos as funções, definimos as variáveis e realizamos a programação do código. Se existem dois arquivos .c no projeto e queremos que um deles possa usar as funções do outro, é necessário realizar um #include.

Os arquivos **.h (header)** têm como função ser um espelho dos arquivos .c disponibilizando as funções para serem utilizadas em outros arquivos. Neles colocamos todos os protótipos das funções que queremos que os outros arquivos usem.

Se quisermos que uma determinada função só possa ser utilizada dentro do próprio arquivo, por motivo de segurança ou organização, basta declará-la APENAS no arquivo .c.

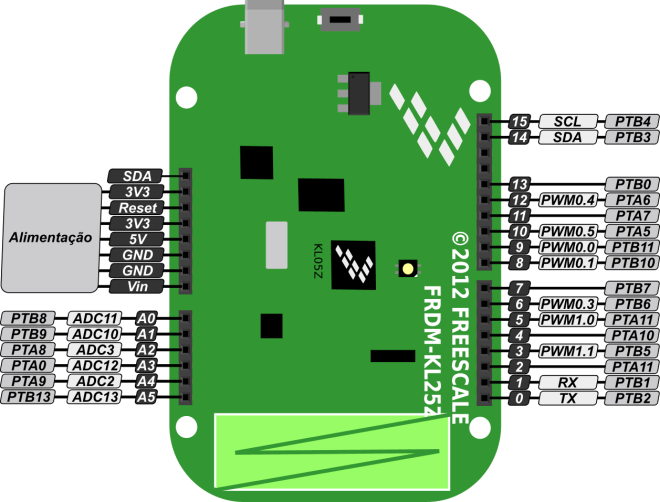
Para a criação do arquivo .c e .h no KDS, clique com o botão direito do mouse na pasta Sources e selecione **New>Source File** e depois **New>Header File**. É comum que este par de arquivos possuam o mesmo nome, apenas sendo diferentes em sua extensão.

É importante que se explicite no final do nome a extensão do arquivo, **.c** (Code) ou **.h** (Header), caso contrário será criado um arquivo **.txt** (bloco de notas).

**Criando a biblioteca**

Com base nas informações anteriores e com o mapa dos pinos da placa, é possível criar uma biblioteca de acionamento dos terminais de IO. Por efeito de simplicidade, vamos utilizar o padrão estabelecido pelo framework Wiring, utilizado pelo Arduino, que é composto de três funções: **digitalRead()**, **digitalWrite()** e**PinMode()**.



Na placa o pino 0 (zero) está conectado na porta PTB 2, como podemos ver na figura acima. Na inicialização deste terminal precisamos então primeiramente configurá-lo como um GPIO. Num segundo momento deve-se especificar qual é o seu sentido: output ou input. Essas configurações podem ser codificadas através da função **PinMode()** conforme o código abaixo:

void PinMode(int pin, int mode)

{

   switch (pin)

   {  
 case 0 :

   PORTB\_PCR2 = PORT\_PCR\_MUX(1); // GPIO Confing. (DATASHEET)

//O Pino 0 pertence a PORTB2, logo a configuração PDDR deve ser realizada no bit 2, em Hex : 0x4u

   if(mode == OUTPUT)

   {

   GPIOB\_PDDR |= 0x4u;               // Configura para OUTPUT

   }

   else if(mode == INPUT)

   {

   GPIOB\_PDDR &=~ 0x4u;                          // Configura para INPUT (DATASHEET)

   }

   break;

//[...] Outros pinos

}

Para acionar terminais que foram configurados como saída, podemos criar a função **DigitalWrite(),** que receberá o terminal desejado bem como o valor de saída esperado:

void DigitalWrite(int pin, int value)

{

   switch (pin)

   {

//O Pino 0 pertence a PORTB2, logo a configuração PDOR deve ser realizada no bit 2, em Hex : 0x4u,

//De acordo o datasheet PDOR(Port Data Output Register) altera a lógica no pino, apenas para pinos OUTPUT

   case 0 :

   if(value == HIGH)

   {

   GPIOB\_PDOR |= 0x4u;     // Pode utilizar o SHIFT, logo :GPIOB\_PDOR |=(1<<2); Facilita o código

   }

   else if(value == LOW)

   {

   GPIOB\_PDOR &=~ 0x4u;                          // Configura para LOW (DATASHEET)

   }

   break;

//[...] Outros pinos

}

Quando o terminal escolhido estiver configurado como input, basta que a função **DigitalRead()** faça a leitura deste terminal e retorne se ele está ligado ou não. Deve-se tomar cuidado pois a implementação a seguir utiliza a lógica de verdadeiro/falso da linguagem C, onde falso é um valor igual a zero e verdadeiro é um valor diferente de zero.

int DigitalRead(int pin)

{

   switch (pin)

   {

//De acordo o datasheet PDIR(Port Data Input Register) é registro apenas de leitura, quando a lógica do pino é 1, o Bit do registro passa a ser 1

   case 0 :

   return GPIOB\_PDIR & 0x4u;

   break;

}

Já o arquivo header da biblioteca é dado pelo código abaixo:

#ifndef IO\_H

   #define IO\_H

   //definições básicas

   #define OUTPUT 1

   #define INPUT 0

   #define HIGH 1

   #define LOW 0

   //protótipos das funções

   int DigitalRead(int pin);

   void DigitalWrite(int pin, int value);

   void PinMode(int pin, int mode);

#endif

Agora o código mínimo apresentado no artigo anterior para piscar um led pode ser reescrito como:

//Autor: Gabriel Silva Marcatto

//    Rodrigo Almeida

#include "MKL05Z4.h"

#include "io.h"

void main(void){

   float i; //apenas para contagem de tempo

//-------------------------Config para boot minimo da placa -------------------------------------------------------

   MCG\_BASE\_PTR ->C4 |= MCG\_C4\_DMX32\_MASK;

   SIM\_BASE\_PTR ->SCGC5 |= (SIM\_SCGC5\_PORTA\_MASK | SIM\_SCGC5\_PORTB\_MASK);

   PORTB\_BASE\_PTR-> PCR[5] = PORT\_PCR\_MUX(1);

   SIM\_BASE\_PTR -> COPC &=~ SIM\_COPC\_COPT\_MASK;

//-----------------------------------------------------------------------------------------------------------

   pinMode(8,OUTPUT);

   for(;;){

       digitalWrite(8,HIGH);

       for(i = 0; i<100000; i++);

       digitalWrite(8,LOW);

       for(i = 0; i<100000; i++);

   }

}