μC Journal – Texas BoosterPack MKII using TivaWare HAL library
Encoder de Quadratura

Pedro Henrique B. Bonifácio
Faculdade de Tecnologia SENAI Anchieta
São Paulo, São Paulo, Brasil
pedrobonifa@gmail.com

***Abstract*— Este trabalho tem como finalidade caracterizar o periférico Encoder de Quadratura presente no microcontrolador *TM4C123G6PM*, explicando seu funcionamento, configuração e fornecendo um código de exemplo.**

***Index Terms*— Encoder, Quadratura, TM4C123G6PM, ARM, Texas Instruments.**

I. INTRODUÇÃO

Encoders de quadratura são sensores utilizados para medição de velocidade e direção de um eixo, seja ele linear ou rotativo, de modo que é possível ter conhecimento da quantidade de movimento realizada pelo equipamento. Este sensor é baseado na leitura de dois sinais, normalmente chamados de Canal A e Canal B; estes canais geram pulsos que podem ser utilizados para medição do movimento e velocidade, através destes pulsos é possível determinar, a partir de uma sequência conhecida, a direção do movimento e a velocidade é determinada pelo intervalo entre pulsos.

Na Figura 1 abaixo tem-se o exemplo de pulsos gerados por um encoder:

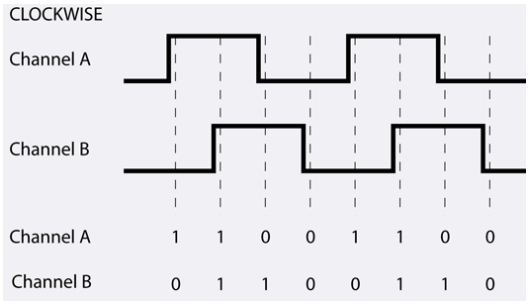

Fig. 1: Exemplo de sinal gerado por encoder.

II. ENCODER DE QUADRATURA NO *TM4C123G6PM*

O microcontrolador *TM4C123G6PM* possui dois módulos encoder de quadratura, cada um deles funciona isoladamente para determinar a direção, velocidade e posição de um eixo rotativo ou linear. Algumas das funcionalidades dos módulos encoder:

- Integrador de posição que monitora a posição do encoder.
- Filtro de ruído programável nas entradas.
- Medição de velocidade utilizando timer interno.
- A frequência do sinal de entrada pode ser de até 1/4 da frequência de funcionamento do microcontrolador.
- Geração de interrupção nos seguintes eventos:
  - Pulso de reset.
  - Estouro do timer da velocidade.
  - Mudança de direção.
  - Erro de detecção.

Neste microcontrolador os canais do enconder são chamados de *PhA* e *PhB*, além de possuir um sinal de reset representado por *ID*. O diagrama de blocos que mostra a lógica de funcionamento de um dos módulos de encoder de quadratura está ilustrada na Figura 2 abaixo.

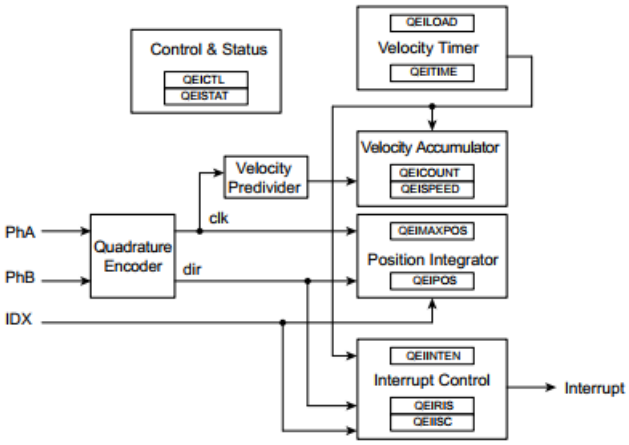

Fig. 2: Diagrama de funcionamento do módulo encoder.

A. *Pinagem*

Para utilizar os módulos encoders, os seguintes pinos estão disponíveis:

Pino Encoder	GPIO
IDX0	PF4 ǀ PD3
PhA0	PF0 ǀ PD6
PhB0	PD7 ǀ PF1
IDX1	PC4
PhA1	PC5
PhB1	PC6

B. *Descrição de Funcionamento*

O módulo QEI interpreta o código *grey* de dois bits produzido por uma roda de encoder de quadratura para integrar a posição ao longo do tempo e determinar o sentido de rotação. Além disso, pode capturar uma estimativa de

funcionamento da velocidade da roda do encoder. O integrador de posição e a captura de velocidade podem ser ativados independentemente, embora o integrador de posição deva ser ativado antes que a captura de velocidade possa ser ativada. Os sinais de dois canais, PhAn e PhBn, podem ser invertidos antes de serem interpretados pelo módulo QEI para alterar o significado de avanço e retrocesso e para corrigir erros de ligação do sistema. Alternativamente, os sinais de fase podem ser interpretados como um sinal de clock e direção como saída por alguns encoders.

O módulo QEI suporta dois modos de operação: modo de fase de quadratura e modo de clock / direção. No modo de fase em quadratura, o encoder produz dois clocks que estão defasados em 90 graus; a relação de bordas é usada para determinar a direção de rotação. No modo de clock / direção, o encoder produz um sinal de clock para indicar os passos e um sinal de direção para indicar a direção da rotação. Este modo é determinado pelo bit SIGMODE do registrador QEICTL. Quando a borda PhA antecede a borda de PhB, o contador de posição é incrementado. Quando a borda PhB antecede a borda de PhA, o contador de posição é diminuído. Quando um par de borda de subida e descida é visto em uma das canais sem bordas no outro, a direção da rotação mudou.

C. Configuração e Inicialização

Para configurar e utilizar o módulo QEI devem ser seguidos os seguintes passos:

1) Habilitar o clock do QEI, utilizar o registrador **RCGCQEI**.
2) Habilitar o clock da GPIO onde o QEI está ligado, utilizar o registrador **RCGCGPIO**.
3) Configurar os pinos do GPIO para função alternativa, utilizar o registrador **GPIOAFSEL**.
4) Configurar os sinais do QEI para os pinos corretos do GPIO, utilizar o registrador **GPIOPCTL**.
5) Configurar o QEI para capturar apenas um ou os dois canais de sinal.
6) Ativar o módulo QEI, utilizar o registrador **QEICTL**.
7) Ler a posição do encoder, utilizar o registrador **QEIPOS**.

Para realizar a configuração utilizando a biblioteca *TivaWare for C Series* deve-se incluir as seguintes bibliotecas de periféricos:

- GPIO
- Quadrature Encoder (QEI)
- System Control

Em anexo tem-se um código exemplo de configuração e uso do Quadrature Encoder (QEI).

```
#ifndef TICK
#define TICK            1000
#endif

#ifndef rvmdk
#define rvmdk
#endif

#ifndef PART_TM4C123GH6PM
#define PART_TM4C123GH6PM
#endif

#ifndef TARGET_IS_TM4C123_RB1
#define TARGET_IS_TM4C123_RB1
#endif

#include "TM4C123GH6PM.h"
//TIVAWARE
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "driverlib/sysctl.h"
#include "driverlib/sysctl.h" //Prototypes for the
   system control driver
#include "driverlib/pin_map.h" //Mapping of
   peripherals to pins for all parts
#include "driverlib/uart.h" //Defines and Macros for
   the UART
#include "driverlib/gpio.h" //Defines and Macros for
   GPIO API
#include "driverlib/qei.h"

static uint32_t systick_ms_count = 1;

uint16_t UART_Write_String(uint32_t uart, uint8_t* txt
   )
{
    uint16_t length = 0;

    while('\0' != *txt)
    {
        UARTCharPut(uart, *(txt++));
        length++;
    }

    return length;
}

void Delay_ms(uint32_t time)
```

```
{
    uint32_t start_time = systick_ms_count;

    while (time > (systick_ms_count - start_time))
    {
        __NOP();
    }
}

uint8_t Hardware_Init(void)
{
    /*  Habilita PLL e configura clock para 80MHz */
    SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL
       | SYSCTL_OSC_MAIN | \
        SYSCTL_XTAL_16MHZ);

    /*  Configura UART0 para debug. UART0 esta nos
       pinos PA0(RX) e PA1(TX) */
    /*  Habilita clock do GPIOA */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    /*  Habilita clock da UART0 */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    /*  Configura as funcoes alternativas para os
       pinos */
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    /*  Configura os pinos para usarem a UART0 */
    GPIOPinTypeUART(GPIOA_BASE, (GPIO_PIN_0 |
       GPIO_PIN_1));
    /*  Desabilita a UART0 antes de realizar a
       configuracao */
    UARTDisable(UART0_BASE);
    /*  Configuracao da UART0 com baudrate de 115200
       */
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(),
       115200, \
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
           UART_CONFIG_PAR_NONE));
    /*  Envia mensagem de teste */
    UART_Write_String(UART0_BASE, "Quadrature Encoder
       Interface \n");

    /*  Configura Quadrature Encoder Interface */
    /*  SERAO UTILIZADOS OS PINOS PD7 E PD6 */
    /*  Possiveis pinagens:
        IDX0 - PF4 | PD3 - Nivel TTL - QEI module 0
           index
        PhA0 - PF0 | PD6 - Nivel TTL - QEI module 0
           phase A
        PhB0 - PD7 | PF1 - Nivel TTL - QEI module 0
```

```
            phase B
        IDX1 -    PC4     - Nivel TTL - QEI module 1
            index
        PhA1 -    PC5     - Nivel TTL - QEI module 1
            phase A
        PhB1 -    PC6     - Nivel TTL - QEI module 1
            phase B
    */
    /*  Habilita o clock da GPIOD */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    /*  Habilita clock do QEI0 */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI0);
    /*  Configura as funcoes alternativas nos pinos */
    GPIOPinConfigure(GPIO_PD6_PHA0);
    GPIOPinConfigure(GPIO_PD7_PHB0);
    /*  Habilita o QEI0 nos pinos */
    GPIOPinTypeQEI(GPIOD_BASE, (GPIO_PIN_6 |
        GPIO_PIN_7));
    /*  Desabilita QEI0 antes de configurar */
    QEIDisable(QEI0_BASE);
    /* Configura o QEI0 */
    QEIFilterConfigure(QEI0_BASE, QEI_FILTCNT_5);
    //QEIFilterEnable(QEI_FILTCNT_17);
    QEIConfigure(QEI0_BASE, (QEI_CONFIG_CAPTURE_A  |
        QEI_CONFIG_NO_RESET    | QEI_CONFIG_QUADRATURE
        | QEI_CONFIG_NO_SWAP), 100);
    QEIEnable(QEI0_BASE);
    QEIPositionSet(QEI0_BASE, 50);
    return 0;
}

int main()
{
    uint32_t last_read = 0;
    uint32_t new_read = 0;

    Hardware_Init();

    SysTick_Config(SysCtlClockGet() / TICK);
    new_read = QEIPositionGet(QEI0_BASE);

    while(1)
    {

        new_read = QEIPositionGet(QEI0_BASE);
        if (last_read != new_read)
        {
            char qeiPosition[10];
            sprintf(qeiPosition, "%d", new_read);
            UART_Write_String(UART0_BASE, "QEI0: ");
```

```
            UART_Write_String(UART0_BASE, (uint8_t*)
                qeiPosition);
            UART_Write_String(UART0_BASE, "\r\n");

            last_read = new_read;

            Delay_ms(500);
        }
    }
}

void SysTick_Handler(void)
{
    systick_ms_count++;
}
```