Configure o PWM do MSP430

Por **Luiz Fernando Palarmini** - 05/05/2016

ÍNDICE DE CONTEÚDO [MOSTRAR]

Antes de iniciarmos esse tópico vamos abordar brevemente algumas características do famoso microcontrolador da Texas Instruments, o MSP430.

O significado da sigla MSP vem do inglês Mixed Signal Processor (Processador de Sinais Mistos). É um microcontrolador de arquitetura RISC de 16 bits e que ganhou fama devido ao seu baixo custo e principalmente seu baixo consumo de energia.

Hoje mostrarei como configurar o PWM em um MSP430 através de dois exemplos. O primeiro é bem simples e o resultado é um duty cycle fixo, onde conseguimos mudar apenas no código. Já o segundo aumentamos o duty cycle através de uma tecla. O hardware utilizado é uma placa da Texas Instruments, a Launchpad EXP430G2 (Figura 1) com o microcontrolador MSP430G2231. O compilador utilizado foi o CCS (Code Composer Studio), que pode ser baixado no próprio site da Texas Instruments. Para implementar o PWM no MSP430 vamos utilizar o periférico Timer0_A.

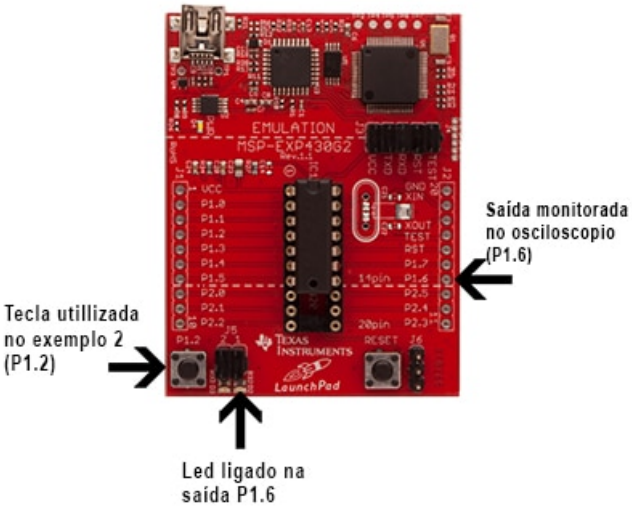

Figura 1 - Placa de Desenvolvimento Launchpad EXP430G2

Exemplo de PWM

Vamos ao primeiro exemplo:

```
/*
 * MSP430 PWM I
 * AUTOR: LUIZ FERNANDO PALARMINI
 * EXEMPLO DE PWM: CONFIGURANDO O DUTY CYCLE
 */

#include <msp430.h>

int main(void) {

 //############################################//
 //####  CONFIGURAÇÃO DO WATCHDOG TIMER  ####//
 //############################################//

 WDTCTL = WDTPW | WDTHOLD; //WATCHDOG TIMER PARADO

 //############################################//
 //#######  CONFIGURAÇÃO DO CLOCK  ##########//
 //############################################//

 DCOCTL = 0;              //
 BCSCTL1 = CALBC1_1MHZ;   //CONFIGURA CLOCK EM 1 MHZ
 DCOCTL = CALDCO_1MHZ;    //

 //############################################//
 //##########  DECLARAÇÃO DE I/Os  ##########//
 //############################################//

 P1DIR |= 0x40; //P1.6 IMPLEMENTADO COMO SAÍDA
 P1SEL |= 0x40; //CONFIGURANDO P1.6 COMO PERIFÉRICO TA0.1

 //############################################//
 //#######  CONFIGURAÇÃO DO TIMER0_A  #######//
 //############################################//

 TACCR0 = 1000; //PERÍODO DO PWM
 TACCTL1 = OUTMOD_7; //MODO DE SAÍDA DO TIMER0_A: RESET/SET
 TACCR1 = 100; //DUTY CYCLE DO PWM EM 10%
 TACTL = TASSEL_2 + MC_1; //TASSEL_2 -> CLOCK SOURCE: MCLK  MC_1 ->                  //TIME

 _BIS_SR(CPUOFF); //DESLIGA A CPU PARA ECONOMIZAR CONSUMO (LPM0)

 return 0;

}
```

Entendendo o código

```
WDTCTL = WDTPW | WDTHOLD; //WATCHDOG TIMER PARADO
```

Nessa linha desabilito o Watchdog Timer.

```
DCOCTL = 0;              //
BCSCTL1 = CALBC1_1MHZ;   //CONFIGURA CLOCK EM 1 MHZ
DCOCTL = CALDCO_1MHZ;    //
```

Aqui configuramos o clock interno do microcontrolador para 1 MHz através do gerador de clock DCO (Digitally Controlled Oscilator - Oscilador Controlado Digitalmente).

```
P1DIR |= 0x40; //P1.6 IMPLEMENTADO COMO SAÍDA
P1SEL |= 0x40; //CONFIGURANDO P1.6 COMO PERIFÉRICO TA0.1
```

Registradores de I/Os, P1DIR selecionamos se o pino será entrada (nível lógico baixo) ou saída (nível lógico alto). P1SEL, nesse registrador informamos se o pino terá função alternativa (no nosso caso, periférico Timer A) ou será I/O.

```
TACCR0 = 1000; //PERÍODO DO PWM
TACCTL1 = OUTMOD_7; //MODO DE SAÍDA DO TIMER0_A: RESET/SET
TACCR1 = 100; //DUTY CYCLE DO PWM EM 10%
TACTL = TASSEL_2 + MC_1; //TASSEL_2 -> CLOCK SOURCE: MCLK MC_1 ->
                         //TIMER COM CONTAGEM PROGRESSIVA DE 0 ATÉ TACCR1
```

Nesse trecho estamos configurando o período do PWM através do registrador TACCR0. Com o valor 1000 estamos informando ao Timer que conte até esse valor. Uma das funções do registrador TACCTL1 é controlar o comportamento das saídas físicas conectadas ao timer, existindo no total 8 modalidades diferentes. Para o PWM utilizaremos somente os modos 3 e 7 (OUTMOD_3 - PWM ativo em nível lógico baixo, OUTMOD_7 - PWM ativo em nível lógico alto). O registrador TACCR1 será o nosso duty cycle do PWM. Como configuramos o clock do MSP430 em 1MHz teremos um período de 1KHz devido ao valor 1000 armazenado em TACCR0 e o duty cycle será 10% devido ao valor 100 armazenado em TACCR1. No registrador TACTL informamos que o clock source do microcontrolador será MCLK (Tassel_2) e o modo de operação do timer será progressivo de 0 até o valor armazenado em TACCR0.

```
_BIS_SR(CPUOFF); //DESLIGA A CPU PARA ECONOMIZAR CONSUMO (LPM0)
```

Nessa linha de código, entramos em modo de economia de energia LPM0, desligando a CPU.

Agora o segundo exemplo:

```
/*
 * MSP430 PWM II
 * AUTOR: LUIZ FERNANDO PALARMINI
 * EXEMPLO DE PWM: CONTROLANDO O DUTY CYCLE ATRAVÉS DE BOTÃO
 */

#include <msp430.h>

int main(void) {

 //#############################################//
 //####  CONFIGURAÇÃO DO WATCHDOG TIMER  ####//
 //#############################################//

 WDTCTL = WDTPW | WDTHOLD; //WATCHDOG TIMER PARADO

 //#############################################//
 //#######  CONFIGURAÇÃO DO CLOCK  ###########//
 //#############################################//

 DCOCTL = 0;              //
 BCSCTL1 = CALBC1_1MHZ;   //CONFIGURA CLOCK EM 1 MHZ
 DCOCTL = CALDCO_1MHZ;    //

 //#############################################//
 //###########  DECLARAÇÃO DE I/Os  ###########//
 //#############################################//

```

```
 P1DIR |= 0x40; //P1.6 IMPLEMENTADO COMO SAÍDA
 P1SEL |= 0x40; //CONFIGURANDO P1.6 COMO PERIFÉRICO TA0.1
 P1DIR &= ~0x08; //P1.3 IMPLEMENTADO COMO ENTRADA
 P1IE  |= 0X08; //HABILITA INTERRUPÇÃO EM P1.3
 P1IFG &= ~0X08; //LIMPA FLAG DE INTERRUPÇÃO DO PINO P1.3
 P1REN |= 0X08; //HABILITA RESISTOR INTERNO EM P1.3
 P1OUT |= 0X08; //RESISTOR DE PULL-UP EM P1.3
 P1IES ^= 0x08;                  //BORDA DE SUBIDA/DESCIDA DO PINO P1.3


 //##########################################//
 //#######  CONFIGURAÇÃO DO TIMER0_A  #######//
 //##########################################//

 TACCR0 = 1000; //PERÍODO DO PWM
 TACCTL1 = OUTMOD_7; //MODO DE SAÍDA DO TIMER0_A: RESET/SET
 TACCR1 = 100; //DUTY CYCLE DO PWM EM 10%
 TACTL = TASSEL_2 + MC_1;        //TASSEL_2 -> CLOCK SOURCE: MCLK   MC_1 ->

 //_BIS_SR(CPUOFF);         //DESLIGA A CPU PARA ECONOMIZAR CONSUMO (LPM0)


 __enable_interrupt();        //HABILITA INTERRUPÇÕES

 while(1);
}


//##########################################//
//#############  INTERRUPÇÕES  #############//
//##########################################//

#pragma vector = TIMER0_A0_VECTOR
__interrupt void timer_int(void){

 TACCTL1 &= ~CCIFG;

}

#pragma vector = PORT1_VECTOR
__interrupt void port_int(void){

 if (P1IN & 0x08) {
 TACCR1 = (TACCR1 + 10)%100; //AUMENTA EM 10% O DUTY CYCLE DO PWM
 }

 P1IFG &= ~0x08; //LIMPA FLAG DE INTERRUPÇÃO DO PINO P1.3

}
```

Vamos agora analisar o que há de novo no código

```
P1IE  |= 0X08; //HABILITA INTERRUPÇÃO EM P1.3
P1IFG &= ~0X08; //LIMPA FLAG DE INTERRUPÇÃO DO PINO P1.3
P1REN |= 0X08; //HABILITA RESISTOR INTERNO EM P1.3
P1OUT |= 0X08; //RESISTOR DE PULL-UP EM P1.3
P1IES ^= 0x08;                  //BORDA DE SUBIDA/DESCIDA DO PINO P1.3
```

Aqui temos Registradores de I/O. P1IE é responsável por habilitar interrupção nos ports, P1IFG é a flag de interrupção dos ports, P1REN habilita resistor interno dos ports e em P1OUT estamos configurando resistor de pull-up.

```
__enable_interrupt(); //HABILITA INTERRUPÇÕES
```

Esta é uma função intrínseca que habilita as interrupções do MSP430.

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void timer_int(void){

 TACCTL1 &= ~CCIFG;
}
```

Aqui limpamos a flag de estouro do timer.

```
#pragma vector = PORT1_VECTOR
__interrupt void port_int(void){

    if (P1IN & 0x08) {
        TACCR1 = (TACCR1 + 10)%100; //AUMENTA EM 10% O DUTY CYCLE DO PWM
```

```
6       }
7
8       P1IFG &= ~0x08; //LIMPA FLAG DE INTERRUPÇÃO DO PINO P1.3
9
10  }
```

Nesse código tratamos a interrupção dos ports, toda vez que o botão que está ligado em P1.3 é pressionado, aumenta o duty cycle do PWM em 10% e limpa a flag de interrupção do pino P1.3.

Segue abaixo (figura 2) as formas de ondas geradas no pino P1.6:

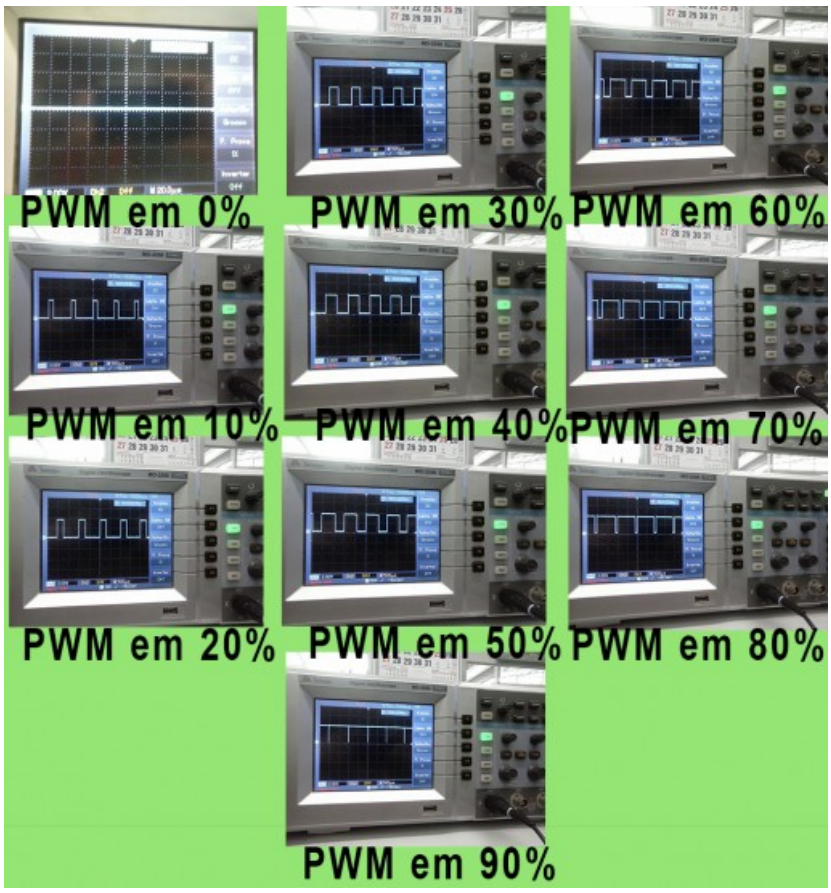

Figura 2: Formas de ondas monitoradas por osciloscópio no pino P1.6

Espero que com esse conteúdo vocês tenham aprendido como configurar o PWM no MSP430 e não deixem de aprofundar mais o conhecimento mudando os valores dos registradores, usando outras opções de configurações e implementando ainda mais os códigos.

Fiquem á vontade para críticas e sugestões.

Fique tranquilo, também não gostamos de spam.

Luiz Fernando Palarmini

Sou formado em Técnico em Mecatrônica, estudante do curso Análise e Desenvolvimento de Sistemas. Apaixonado por eletrônica, programação e empreendedorismo. Desde cedo sempre fui curioso em saber como tudo funcionava e aprender coisas novas. Gosto de observar pessoas que venceram na vida, saber como elas pensam e agem para aprender um pouco de cada uma e assim crescer pessoalmente, profissionalmente e um dia poder estar entre elas.