

# Tacômetro Óptico

Wilton Miro Barros Júnior  
FGA - Faculdade do Gama  
UnB - Universidade de Brasília  
Gama, Brasil  
wiltonjrfla@gmail.com

Igor de Alcantara Rabelo  
FGA - Faculdade do Gama  
UnB - Universidade de Brasília  
Gama, Brasil  
rabelo.alcantara.igor@gmail.com

**Resumo**—Este documento visa mostrar a elaboração do projeto de um tacômetro óptico com emissor e receptor infravermelho que será controlado pelo microcontrolador MSP430 e terá a visualização da medição em um display LCD.

**Keywords**—tacômetro, óptico, infravermelho, LCD

## I. INTRODUÇÃO

Os motores elétricos que são capazes de converter energia elétrica em energia mecânica e são utilizados em diversas máquinas que usamos no dia-a-dia[4]. Algumas vezes é necessário fazer testes de medições para saber se realmente o motor está em sua rotação ideal e o tacômetro é um dispositivo que é usado para obter o número de rotação de um motor. O laboratório de eletricidade da UnB-Gama possui esses motores elétricos que são usados para o aprendizado desde o manuseio até as configurações que são descritas pelo fabricante.

Esse projeto consiste em fazer um tacômetro usando sensores LED emissor e receptor infravermelho para medir a rotação do motor do laboratório da UnB-Gama e identificar através de um display de LCD16x2, se a rotação está adequada de acordo com as ligações que são usadas para o funcionamento adequado e também para verificar se está de acordo com o que foi proposto pelo fabricante. Será usado um microcontrolador Msp430g2553.

## II. DESENVOLVIMENTO

Para saber se o motor está realmente conforme o que está proposto pelo fabricante, planeja criar um tacômetro óptico que é um dispositivo capaz de medir a rotação do motor através de sensores LED's emissor e receptor infravermelho que ao incidir um feixe em uma fita refletiva fixada no eixo do motor que será capturada

pelo emissor e assim através do processamento do microcontrolador msp430g2553 será obtido um resultado que será informado pelo display de LCD 16x2.

### A. Descrição de Hardware

Para a realização deste projeto foi utilizados os seguinte materiais.

Tabela 1. Lista de materiais

Lista de materiais	
Item	Quantidade
MSP430g2553 LaunchPad	1
LED receptor infravermelho	1
LED emissor infravermelho	1
Jumpers	19
Protoboard	2
Display LCD 16x2	1
Potenciômetro 10k	1
Ferro de solda	1
Solda	1

O hardware consiste em dois LED's emissor e receptor infravermelho que são ligados no microcontrolador Msp430g2553. Ao ligar o LED emissor, um feixe infravermelho é lançado e esse feixe é refletido através de uma fita e volta para o receptor. No código que foi feito para o receptor, o LED receptor

foi ligado no pino A0, que é equivalente ao pino P1.0 da placa Msp430.

O LCD foi testado usando um exemplo da biblioteca do software energia que já possuía as pinagens corretas de conectar o display no MSP430. O display contém 16 entradas e os pinos que foram utilizados do msp430 foram: P2.0, P2.1, P2.2, P2.3, P2.4, P2.5, GND e VCC. Para a função RS e EN foram utilizados os pinos P2.0 e P2.1 que corresponde às entradas do display 4 e 6, para função DB4 a DB7 foram utilizadas P2.2, P2.3, P2.4, P2.5 que corresponde às entradas dos display 11 a 14, e enfim para alimentar o display foram utilizadas GND e VCC que correspondem às entradas do display 1,2,5,15 e 16. O funcionamento está de acordo com o diagrama de blocos do ANEXO E.

### B. Descrição de Software

Foi desenvolvido um código em linguagem C para msp430 usando a biblioteca msp430g2553.h no code composer studio 8.0.0. Foram utilizados 4 bits do display de lcd(D4 a D7). Existem outros 4 bits da placa de lcd que é o R/W, EN, RS, Contrast, Gnd e Vcc. O R/W é responsável pelo fluxo de dados entre a placa e o microcontrolador. Isso é, quando o R/W está ligado no Vcc ele está lendo os dados do sensor no display e quando R/W está no Gnd ele está escrevendo no display. O pino contraste é utilizado para aumentar ou diminuir o contraste do display de lcd através de um potenciômetro. Os pinos A e K são utilizados para iluminação do display.

Devido a configuração do display lcd ser de 4 bits, é necessário enviar 4 bits em dois pacotes, sendo que os 4 bits mais significativos serão enviados primeiro e depois os 4 menos significativos. Com isso foi preciso realizar um tratamento para fazer o envio dos dois pacotes. Esse tratamento consistiu em realizar o deslocamento dos 4 bits mais significativos e depois os 4 bits menos significativos de forma que os bits ficassem disponíveis nas portas conectadas ao lcd para posterior envio.

Para realizar a exibição no display, foi utilizado o botão P1.3 da msp430g2553, que estava operando como entrada, que a informação de acionamento do botão. O registrador P1DIR deve estar habilitado em nível 0 para que a porta P1.3 seja considerada como entrada. Esses registradores são responsáveis pela habilitação de resistor pull-up/pull-down(P1REN), seleção de modo pull-up(P1OUT), habilitação da interrupção (P1IE), seleção do tipo de borda da interrupção(P1IES) e limpeza do flag de interrupção(P1IFG). O registrador de habilitação para interrupção é fundamental para que o funcionamento da interrupção seja adequado. Sem esse registrador a interrupção não é ativada mesmo que o

botão de interrupção na porta P1.3 seja pressionado, a interrupção não funciona.

Após configurar os registradores da porta P1.3 e a interrupção, foi realizado o tratamento da interrupção gerada pelo acionamento do botão. Por último foi desenvolvido o código para transformar a variável int para char, porque o display lcd só aceita valores de uma variável char. Todos o código desenvolvido está descrito no apêndice C.

Em seguida foi desenvolvido o código do tacômetro. Esse código foi baseado de acordo com as interrupções que ocorriam na entrada P1.4 que foi utilizada para realizar a contagem das voltas do motor - foi utilizado um cooler no lugar do motor - e posteriormente foi inserido no código o cálculo da velocidade em RPM (Rotações por minuto) de acordo com a fórmula a seguir:

Equação 1. Cálculo RPM.

$$\text{rpm} = 1200000 / \text{miliseconnds} // \text{RPM}(20 \text{ turns}) * 60000 (\text{ms} = 1 \text{min}) / \text{miliseconds}.$$

O timer A0 que está descrito no código do anexo C é um temporizador de contagem de tempo, na qual é disparado uma interrupção em 1 milissegundos (ms). Para ativar a interrupção do Timer A0 foi utilizado a instrução TA0CCTL0 |= CCIE e utilizou TA0CCTL0 |= CCIE para setar em um ciclo de um período de 1 ms (frequência de 1 kHz).

O código em Assembly está descrito no anexo D. Foi feito a tradução do código C em Assembly usando o software IAR workbench. O código em Assembly é exatamente o mesmo que em C entretanto houve alguns erros de tradução principalmente nas partes de interrupção.

## III. RESULTADOS

Após a implementação do código na placa Msp430g2553 foi possível identificar no display LCD o nome do projeto “Tacometro” na primeira linha e na segunda linha a velocidade medida em rpm pelo sensor receptor ligado na porta P1.4 da placa. Foi ligado um cooler que funcionou como o motor em uma fonte de 5V e logo depois foi ligado a uma fonte de 12V. Assim foi possível identificar a diferença de velocidade entre os dois testes. Isso ocorreu devido a alimentação do cooler que suporta Vmax=12V e funciona em sua potência máxima, ou seja, o cooler possui maior rotação ligado em 12V. O circuito final consistiu na implementação do circuito que está no anexo A concatenado ao circuito que está representado no anexo B.

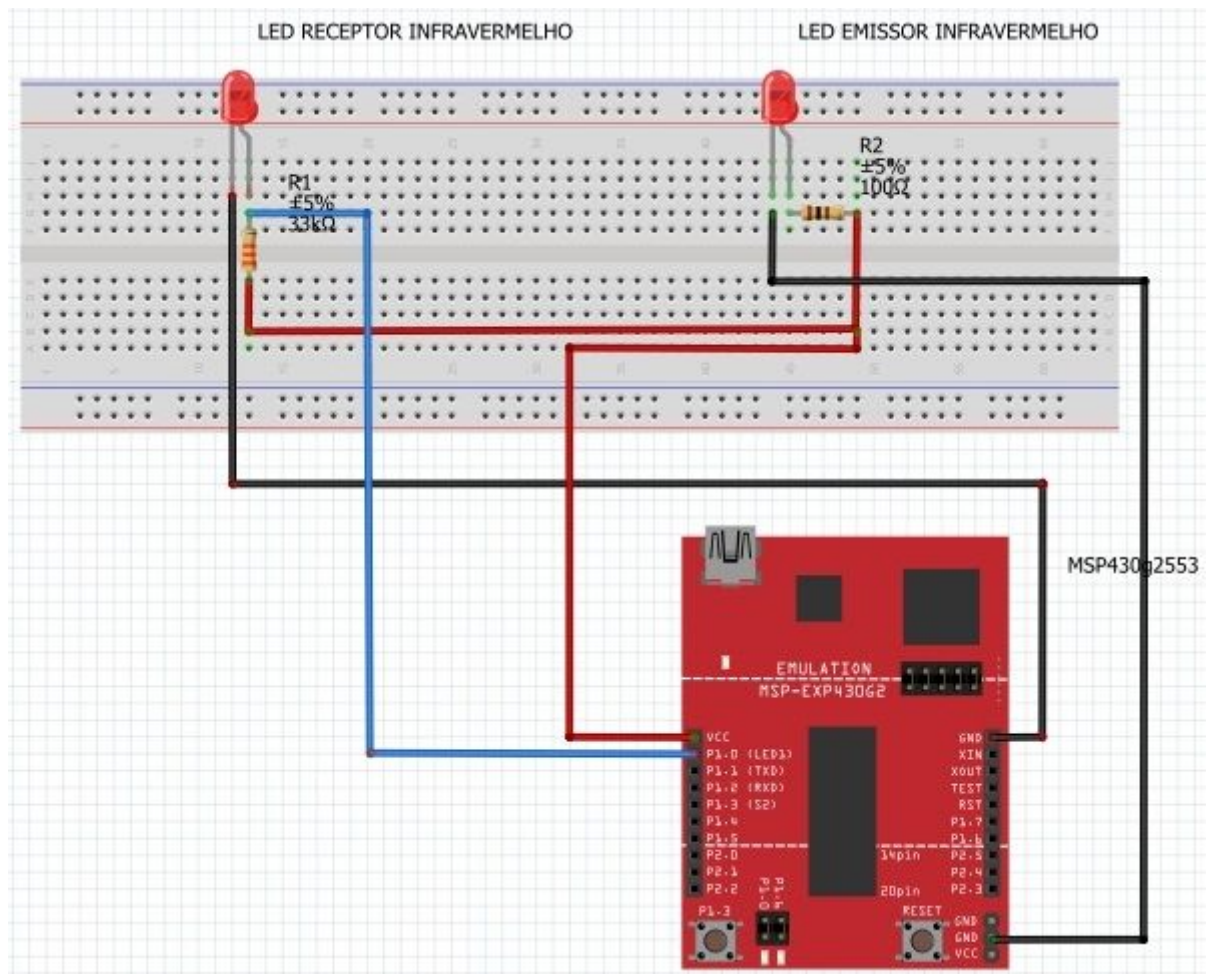
#### IV. CONCLUSÃO

Conclui-se que todo o código implementado do tacômetro funcionou de forma adequada. As velocidades medidas mostradas no display condizem com os valores teóricos pesquisados em fóruns. Entretanto vai ser preciso compará-los com um tacômetro óptico profissional para ter a real certeza de que o valor medido é próximo ou há um erro onde será preciso arrumá-lo.

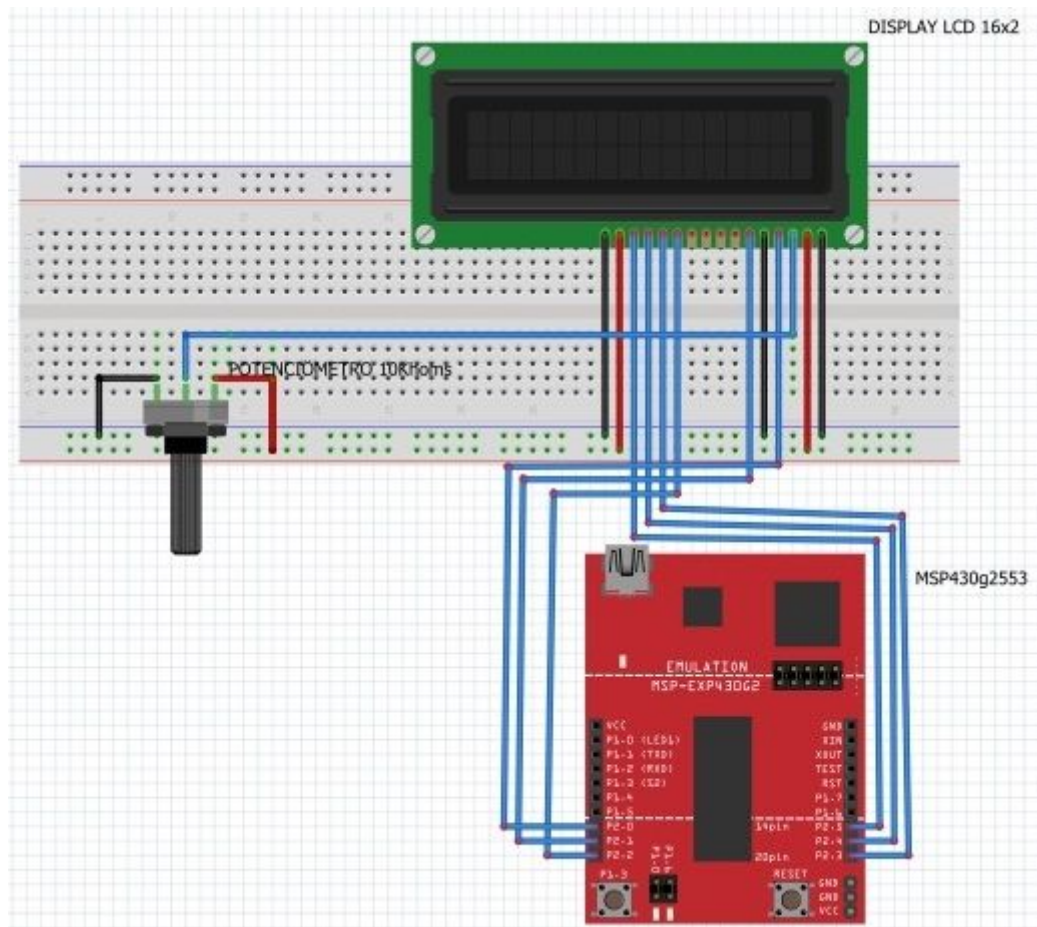
#### REFERÊNCIAS

- [1] <https://www.filipeflop.com/blog/controlando-um-lcd-16x2-com-arduino/>
- [2] <http://www.instructables.com/id/Interfacing-16x2-LCD-with-msp430-launchpad-in-8-bi/>
- [3] <https://www.circuitvalley.com/2011/12/16x2-char-lcd-with-ti-msp430-launch-pad.html>
- [4] <https://brasilecola.uol.com.br/fisica/electricidade-acionamento-motores-eletricos.htm>
- [5] Davies, J., MSP430 Microcontroller Basics, Elsevier, 2008
- [6] MSP430 Assembly Language Tools v18.1.0.LTS - User's Guide

## ANEXO A - Esquemático emissor e receptor infravermelho



## ANEXO B - Esquemático display LCD 16x2



## ANEXO C - Código do display e Tacômetro em C

```
#include "msp430g2553.h"

// LCD control pins definitions
#define DATA_REG P1OUT = BIT5
#define INST_REG P1OUT = (~BIT5)
#define ENABLE_PIN_HIGH P2OUT |= BIT0
#define ENABLE_PIN_LOW P2OUT &= (~BIT0)

//Variable declaration
int turn_counter = 0;
int miliseconds = 0;
int rpm = 0;
char char_rpm[5];
char char_miliseconds[5];

// Implementation of itoa function(convert values in strings to show on LCD)
char *itoa(int from, char to[])
{
    char const digit[] = "0123456789";
    char* p = to;
    int shifter;

    if(from < 0){
        *p++ = '-';
        from *= -1;
    }

    shifter = from;

    do{
        ++p;
        shifter = shifter / 10;
    }while(shifter);

    *p = '\0';

    do{
        *--p = digit[(from % 10)];
        from = from / 10;
    }while(from);

    return to;
}

void configure_clocks()
{
    WDTCTL = WDTPW + WDTHOLD;
    DCOCTL = CALDCO_1MHZ;
    BCSCTL1 = CALBC1_1MHZ;
    BCSCTL2 = 0x00;
    BCSCTL3 = 0x00;
}

void delay_us(unsigned int us)
```

```

{
    while(us)
    {
        __delay_cycles(1);
        us--;
    }
}

void delay_ms(unsigned int ms)
{
    while(ms)
    {
        __delay_cycles(1000);
        ms--;
    }
}

void data_write(void)
{
    ENABLE_PIN_HIGH;
    delay_ms(5);
    ENABLE_PIN_LOW;
}

void send_data(unsigned char data)
{
    unsigned char higher_nibble = 0x3c & (data >> 2);
    unsigned char lower_nibble = 0x3c & (data << 2);

    delay_us(200);

    DATA_REG;
    P2OUT = (P2OUT & 0xc3)|(higher_nibble);
    data_write();
    P2OUT = (P2OUT & 0xc3)|(lower_nibble);
    data_write();
}

void send_string(char *s)
{
    while(*s)
    {
        send_data(*s);
        s++;
    }
}

void send_command(unsigned char cmd)
{
    unsigned char higher_nibble = 0x3C & (cmd >> 2);
    unsigned char lower_nibble = 0x3C & (cmd << 2);

    INST_REG;

    P2OUT = (P2OUT & 0xC3)|(higher_nibble);
    data_write();

    P2OUT = (P2OUT & 0xC3)|(lower_nibble);
    data_write();
}

void lcd_init(void)

```

```

{
    P1DIR = 0x20;
    P1OUT = 0x00;
    P2DIR = 0x3D;
    P2OUT = 0x00;

    delay_ms(15);
    send_command(0x33);

    delay_us(200);
    send_command(0x32);

    delay_us(40);
    send_command(0x28);

    delay_us(40);
    send_command(0x0E);

    delay_us(40);
    send_command(0x01);

    delay_us(40);
    send_command(0x06);

    delay_us(40);
    send_command(0x80);
}
void calc_rpm(void){
    rpm=1200000/milliseconds;
    itoa(milliseconds,char_milliseconds);
    itoa(rpm, char_rpm);
    send_command(0xC0);
    send_string("Vel=");
    send_string(char_rpm);
    send_string(" RPM");
    send_command(0x0C);
}
int main(void)
{
    configure_clocks();

    lcd_init();
    send_command(0x80);
    send_string("TACOMETRO");

    P1REN |= 0X08;
    P1OUT |= 0X08;
    P1IE |= 0X18;
    P1IES |= 0x08;
    P1IES &= ~0x10;
    P1IFG &= ~0x18;

    // TIMER A0
    TA0CTL = TASSEL_2 + MC_1 + ID_0;

    TA0CCTL0 |= CCIE;
    TA0CCR0 = 1000-1;

```



```

__bis_SR_register(GIE);

milliseconds = 0;
turn_counter = 0;

while(1){
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
    milliseconds++;
}

#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void)
{
    if(turn_counter < 179){
        turn_counter++;
    }
    else{
        calc_rpm();
        turn_counter=0;
        TA0CTL |= TACLR;
        milliseconds = 0;
    }
    P1IFG &= ~0x18;
}

```

## ANEXO D - Subrotina em Assembly

lcd\_init:

```
mov.b    #0x20, &P1DIR
clr.b     &P1OUT
mov.b     #0x3D, &P2DIR
clr.b     &P2OUT
mov.w     #0xF, R12
call      #delay_ms
mov.b     #0x33, R12
call      #send_command
mov.w     #0xC8, R12
call      #delay_us
mov.b     #0x32, R12
call      #send_command
mov.w     #0x28, R12
call      #send_command
mov.b     #0x28, R12
call      #delay_us
mov.b     #0x28, R12
call      #send_command
mov.w     #0x28, R12
call      #delay_us
mov.b     #0xE, R12
call      #send_command
mov.w     #0x28, R12
call      #delay_us
mov.b     #0xE, R12
call      #send_command
mov.w     #0x28, R12
call      #delay_us
mov.w     #0x28, R12
call      #delay_us
mov.b     #0x1, R12
call      #send_command
mov.w     #0x28, R12
call      #delay_us
mov.b     #0x6, R12
call      #send_command
mov.w     #0x28, R12
call      #delay_us
mov.b     #0x80, R12
br        #0xC26E
```

itoa:

```
push.w    R10
push.w    R11
push.w    R8
sub.w     #0xC, SP
mov.w     R12, R10
mov.w     R13, R11
mov.w     SP, R12
add.w     #0x0, R12
mov.w     0xC000, R14
mov.w     #0xB, R13
call      #?CopyMemoryBytes
```

```

mov.w    R11, R8
tst.w    R10
jge      0xC0F0
mov.b    #0x2D, 0x0(R8)
inc.w    R8
mov.w    R10, R12
mov.w    #0xFFFF, R14
call     #?mul16
mov.w    R12, R10
mov.w    R10, R12
inc.w    R8
mov.w    #0xA, R14
call     #?DivMod16s
tst.w    R12
jne      0xC0F2
clr.b    0x0(R8)
add.w    #0xFFFF, R8
mov.w    R10, R12
mov.w    #0xA, R14
call     #DivMov16s
add.w    SP, R14
mov.b    @R14, 0x0(R8)
mov.w    R10, R12
mov.w    #0xA, R14
call     #?DivMods
mov.w    R12, R10
tst.w    R10
jne      0xC104
mov.w    R11, R12
add.w    #0xC, SP
br       #0xC418
mov.w    #0x4F80, R12
mov.w    #0x12, R13
mov.w    &milliseconds, R14
mov.w    R14, R15
inv.w    R15
rla.w    R15
subc.w   R15, R15
call     #?DivMod32s
mov.w    R12, &rpm
mov.w    #0x20B, R13
mov.w    &milliseconds, R12
call     #itoa
mov.w    #0x206, R13
mov.w    &rpm, R12
call     #itoa
mov.b    #0xC0, R12
call     #send_command
mov.w    #0xC00B, R12
call     #send_string
mov.w    #0x206, R12
call     #send_string
mov.w    #0xC010, R12
call     #send_string
mov.b    #0xC, R12
br       #0xC26E

```

main:

```

call     #configure_clocks

```

	call	#lcd_init
	mov.w	#0x80, R12
	call	#send_command
	mov.w	#0xC015, R12
	call	#send_string
	bis.b	#0x8, &P1REN
	bis.b	#0x8, &P1OUT
	bis.b	#0x8, &P1IE
	bis.b	#0x8, &P1IES
	bis.b	#0x10, &P1IES
	and.b	#0xE7, &P1IFG
	mov.w	#0x210, &TA0CTL
	bis.w	#0x10, &TA0CCTL0
	mov.w	#0x3E7, &TA0CCR0
	eint	
	clr.w	&milliseconds
	clr.w	&turn_counter
	jmp	0xC1DE
send_data:		
	push.w	R10
	mov.b	R12, R13
	mov.b	R13, R10
	clrc	
	rrc.b	R10
	rra.b	R10
	and.b	#0x3C, R10
	rla.b	R13
	rla.b	R13
	and.b	#0x3C, R13
	mov.w	#0xC8, R12
	call	delay_us
	mov.b	#0x20, &P1OUT
	mov.b	&P2OUT, R14
	and.b	#0xC3, R14
	bis.b	R10, R14
	mov.b	R14, &P2OUT
	call	#data_write
	mov.b	&P2OUT, R14
	and.b	#0xC3, R14
	bis.b	R10, R14
	mov.b	R14, &P2OUT
	call	#data_write
	mov.b	&P2OUT, R14
	and.b	#0xC3, R14
	bis.b	R13, R14
	mov.b	R14, &P2OUT
	call	#data_write
	pop.w	R10
	ret	
DivMod32s:		
	push.w	R9
	clr.w	R9
	tst.w	R15
	jge	0xC240
	inv.w	R14
	inv.w	R15
	inv.w	R14
	adc.w	R15

	bis.w	#0x1, R9
	tst.w	R13
	jge	0xC24E
	inv.w	R12
	inv.w	R13
	inc.w	R12
	adc.w	R13
	inv.w	R9
	call	?DivMod32u
	bit.w	#0x1, R9
	jeq	0xC25E
	inv.w	R12
	inv.w	R13
	inc.w	R12
	adc.w	R13
	bit.w	#0x2, R9
	jeq.w	0xC26A
	inv.w	R14
	inv.w	R15
	inc.w	R14
	adc.w	R15
	pop.w	R9
	ret	
send_command:		
	mov.b	R12, R13
	mov.b	R13, R14
	clrc	
	rrc.b	R14
	rra.b	R14
	and.b	#0x3C, R14
	rla.b	R13
	rla.b	R13
	and.b	#0x3C, R13
	mov.b	#0xDF, &P1OUT
	mov.b	&P2OUT, R15
	and.b	#0xC3, R15
	bis.b	R14, R15
	mov.b	R15, &P2OUT
	call	data_write
	mov.b	&P2OUT, R14
	and.b	#0xC23, R14
	bis.b	R13, R14
	mov.b	R14, &P2OUT
	br	#0xC420
DivMod32u:		
	push.w	R9
	push.w	R10
	push.w	R11
	clr.w	R10
	clr.w	R11
	mov.w	#0x20, R9
	rla.w	R12
	rlc.w	R13
	rlc.w	R10
	rlc.w	R11
	sub.w	R14, R10
	subc.w	R15, R11
	jnc	0xC2D2

	bis.w	#0x1, R12
	add.w	#0xFFFF, R9
	jne	0xC2BC
	jmp	0xC2DA
	add.w	R14, R10
	addc.w	R15, R11
	add.w	#0xFFFF, R9
	jne	0xC2BC
	mov.w	R10, R14
	mov.w	R11, R15
	pop.w	R11
	pop.w	R10
	pop.w	R9
	ret	
Port_1:		
	push.w	R13
	push.w	R12
	push.w	R15
	push.w	R14
	cmp.w	#0xB3, &turn_counter
	jge	0xC2FC
	inc.w	&turn_counter
	jmp	0xC2FC
	inc.w	&turn_counter
	jmp	0xC30C
	call	#calc_rpm
	clr.w	&turn_counter
	bis.w	#0x4, &TA0CTL
	clr.w	&milliseconds
	and.b	#0xE7, &P1IFG
	pop.w	R14
	pop.w	R15
	pop.w	R13
	reti	
DivMod8s:		
	sxt	R12
	sxt	R14
DivMod16s:		
	push.w	R9
	clr.w	R9
	tst.w	R14
	jge	0xC32E
	inv.w	R14
	inc.w	R14
	bis.w	#0x1, R9
	tst.w	R12
	jge	0xC338
	inv.w	R12
	inc.w	R12
	inv.w	R9
	call	#DivMod16u
	bit.w	#0x1, R9
	jeq	0x2, R9
	inv.w	R14
	inc.w	R14
	pop.w	R9
	ret	
Mul8:		

	and.b	#0xFF, R12
	and.b	#0xFF, R14
Mul16:		
Mul16to32u:		
	push.w	R9
	mov.w	R12, R9
	cmp.w	R14, R12
	jnc	0xC362
	mov.w	R14, R9
	mov.w	R12, R14
	clrc	
	clr.w	R15
	clr.w	R12
	clr.w	R13
	rrc.w	R9
	jnc	0xC370
	add.w	R14, R12
	addc.w	R15, R13
	rla.w	R14
	rlc.w	R15
	rrc.w	R9
	jne	0xC36A
	jnc	0xC37E
	add.w	R14, R12
	addc.w	R15, R13
	pop.w	R9
	ret	
configure_clocks:		
	mov.w	#0x5A80, &WDTCTL
	mov.b	&CALDCO_1MHZ, &CDOCTL
	mov.b	&CALBC1_1MHZ, &BCSCTL1
	clr.b	&BCSCTL2
	clr.b	&BCSCTL3
DivMod8u:		
	and.b	#0xFF, R12
	and.b	#0xFF, R14
	mov.w	R14, R15
	mov.w	R12, R13
	clr.w	R14
	mov.w	#0x1, R12
	rla.w	R13
	rlc.w	R14
	cmp.w	R15, R14
	jnc	0xC3B4
	sub.w	R15, R14
	rlc.w	R12
	jnc	0xC3AA
	ret	
send_string:		
	push.w	R10
	mov.w	R12, R10
	jmp	0xC3C8
	mov.b	@R10, R12
	call	#send_data
	inc.w	R10
	tst.b	0x0(R10)
	jne	0xC3C0
	pop.w	R10

```

        ret
__exit:
        push.w      R10
        decd.w      SP
        mov.w       R12, R10
        mov.w       R10, R13
        mov.w       SP, R13
        add.w       #0x0, R13
        mov.b       #0x1, R12
        call        #__DebugBreak
        jmp         0xC3D8
delay_ms:
        jmp         0xC3F6
        jmp         0xC3EC
        mov.w       #0x14B, R15
        add.w       #0xFFFF, R15
        jc          0xC3F0
        add.w       #0xFFFF, R12
        tst.w       R12
        jne        0xC3EA
        ret
__data16_memzero:
        mov.w       R12, R15
        add.w       R13, R15
        jmp         0xC408
        clr.b       0x0(R12)
        inc.w       R12
        cmp.w       R15, R12
        jne        0xC402
        ret
Epilogue8:
        pop.w       R5
Epilogue7:
        pop.w       R4
Epilogue6:
        pop.w       R7
Epilogue5:
        pop.w       R6
Epilogue4:
        pop.w       R9
Epilogue3:
        pop.w       R8
        pop.w       R11
        pop.w       R10
        ret
data_wirte:
        bis.b       #0x1, &P2OUT
        mov.w       #0x5, R12
        call        delay_ms
        bic.b       #0x1, &P2OUT
        ret
CopyMemoryBytes:
        push.w      R12
        mov.b       @R14+, 0x0(R12)
        inc.w       R12
        dec.w       R13
        jne        0xC434
        pop.w       R12

```



	ret	
delay_us:	jmp	0xC448
	nop	
	add.w	#0xFFFF, R12
	tst.w	R12
	jne	0xC444
	ret	
Timer_A:	inc.w	&milliseconds
	reti	
Exit:		
	br	#0xC458
_Exit:		
	br	#0xC3D2
__DebugBreak:		
	ret	
	and.b	@R15+, 0xFFFF(R15)

## ANEXO D - Diagrama de blocos

