Instituto Superior de Engenharia do Porto

Interligação de Sensores e de Actuadores

Aluno :
Sérgio Santos, *Nº*: 1020881

Docente/Orientador

Abel António de Azevedo Ferreira, *abe*
Lino Manuel Baptista Figueiredo, *lbf*

Unidade Curricular
LABSIS

14 de fevereiro de 2021

Led Pisca e circuito Dimmer

1 Introdução

Este relatório tem como objetivo a implementação de um **Dimmer** controlado por um potenciómetro [PC0] recorrendo a **PWM** (Power Width Modulation) sendo um **LED** a sua carga [PB1].
O PWM usado é de oito bits, dai o **ADC** tem que ser adaptado de forma a gerar oito bits para controlar o PWM, neste caso Fast PWM, um LED vai estar a piscar a 1 Hz no PORTD6 permanentemente em simultâneo.
O código deste trabalho primeiro será feito em Assembler e depois em C.

Mas primeiro vai ser feito quatro programas apenas concentrado a volta de pôr um led a piscar a **Um** Hertz no PORTD6 em assembly e C por software e depois utilizando interrupções.
Isto feito assim para servir como introdução ao microcontrolador da AVR [Atmega88] pelo "Meu Primeiro Programa ", que consist em por um **LED** (light emitting diode) a piscar no mundo dos **MCU** (Microcontroller Unit).

2 Arquitetura

O **CPU** (Unidade Central de Processamento) dos microcontroldores da Atmel de 8 e 32 bits são baseados na arquitetura avançada de **Harvard** na qual esta concebido para baixos consumos e performance.

Este tipo de arquitetura tem dois busses (barramentos) um dedicado a leitura das instruções a executar e outra para escrita e leitura de data (informação ou dados), isto assegura que uma nova instrução pode ser executada em cada ciclo de relógio, na qual elimina estados de espera quando não ha instruções prontas a executar.

Nos microcontroladores da AVR os barramentos estão configurados de forma a dar prioridade ao barramento das instruções do CPU acesso a memoria flash enquanto o barramento da CPU de dados tem prioridade de acesso a **SRAM** (Static Random Access Memory).

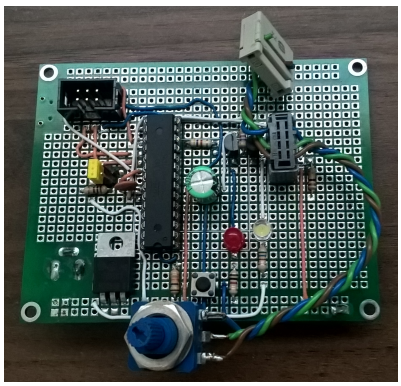
O espaço de memoria de dados é dividida em três, os **GPR** (General Purpose Registers) as **SFRs** (Special Function Registers) ou memoria de I/O e a data SRAM.

Os microcontroladores da AVR utiliza uma arquitetura de instruções **RISC** (Reduced Instruction Set Computer ou Reduced COMPLEXITY Instruction Set Computer) na qual reduz a complexidade dos circuitos na codificação de cada instrução.

Dai que os microcontroladores que se baseiam nestes tipos de arquitetura são sinonimo de código reduzido, alta performance e baixo consumo energético

3 Hardware

O micro-controlador a ser usado é um Atmega88, seu Datasheet (Manual do Componente) é uma peça fundamental para usar como suporte na sua utilização.
Para Facilitar seu desenvolvimento foi feito o circuito numa placa pre furada, e sua implementação podendo ser alimentada por uma fonte DC 12Volt por um Jack.



Para programação e debug do integrado é usado um Atmel-ICE, uma ferramenta de desenvolvimento que neste caso do i.c Atmega88 tem disponível programação via **ISP** e debug por **debugWIRE**.
Os Parâmetros de configuração do integrado são os seguintes, Device Signature do Atmega88=0x1E930A, **ISP** clock a 125Khz, BOOTZ=1024W_0C00, SPIEN=ON, BODLEVEL=2V7, SUT_CKSEL=Int. RC. Osc. 8MHZ_XX_16KCK_14CK_65MS , EXTENDED FUSE= 0xF9,HIGH FUSE=0xD5, LOW FUSE=0xE2 e Lock bits OFF ou seja 0xFF.

4 Software

Nesta secção é feito o código correspondente em por um LED a piscar a 1Hz no PORTD6 e depois o código com o PWM controlado por uma entrada analógica, nos casos mencionados na introdução.
A ferramenta de desenvolvimento (**IDE**) utilizado é o Atmel Studio 7 (versão 7.0.129)
Deve-se ter em atenção que no assembly a rotina JMP e CALL não funcionam no ATmega88 pois não fazem parte do seu conjunto de instruções, como indicado no seu Datasheet, mas recorrer a RJMP e RCALL respetivamente.
1Hz:

$$\boxed{F_{OCnx} = \frac{F_{clk\_I/O}}{2.N.(1+OCRnx)}} \quad \Longrightarrow \quad T_{OCRnx} = 2.N.(1+OCRNx) \times T_{clk\_I/O} \tag{1}$$

ADC:

$$\boxed{ADC = \frac{V_{IN}.1024}{V_{REF}}}$$

fast PWM:

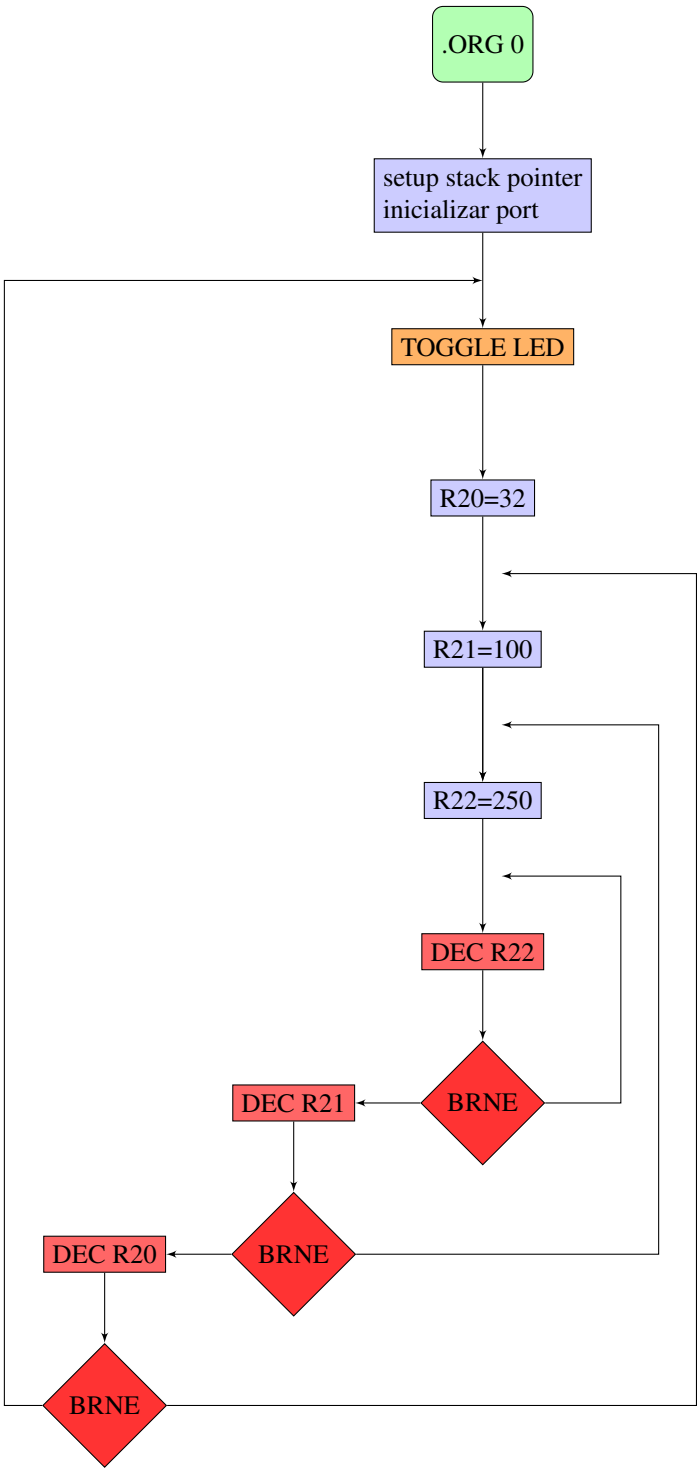
$$\boxed{F_{OCnxPWM} = \frac{F_{clk\_I/O}}{N.(1+TOP)}} \quad \textit{com resolução minima} \quad \boxed{R_{FPWM} = \frac{log(TOP+1)}{log(2)}}$$

Os parâmetros escolhidos para o ADC são o PINC0 como entrada com clock dividido por 128, o mais lento possível, a tensão de referencia é VCC com capacidade em Vref.

Utilizando fast PWM com N=256 e TOP=255, dai pode-se calcular sua respetiva frequência e duty cycle mínimo, pelas formulas. A saída do PWM é no pino OC1nA ou PORTB1, e o led a piscar a um Hertz no pino PORTD6, não ha necessidade de o mencionar o código auto explicativo, mesmo assim.

4.1 Led Pisca 1Hz em assembly por software.

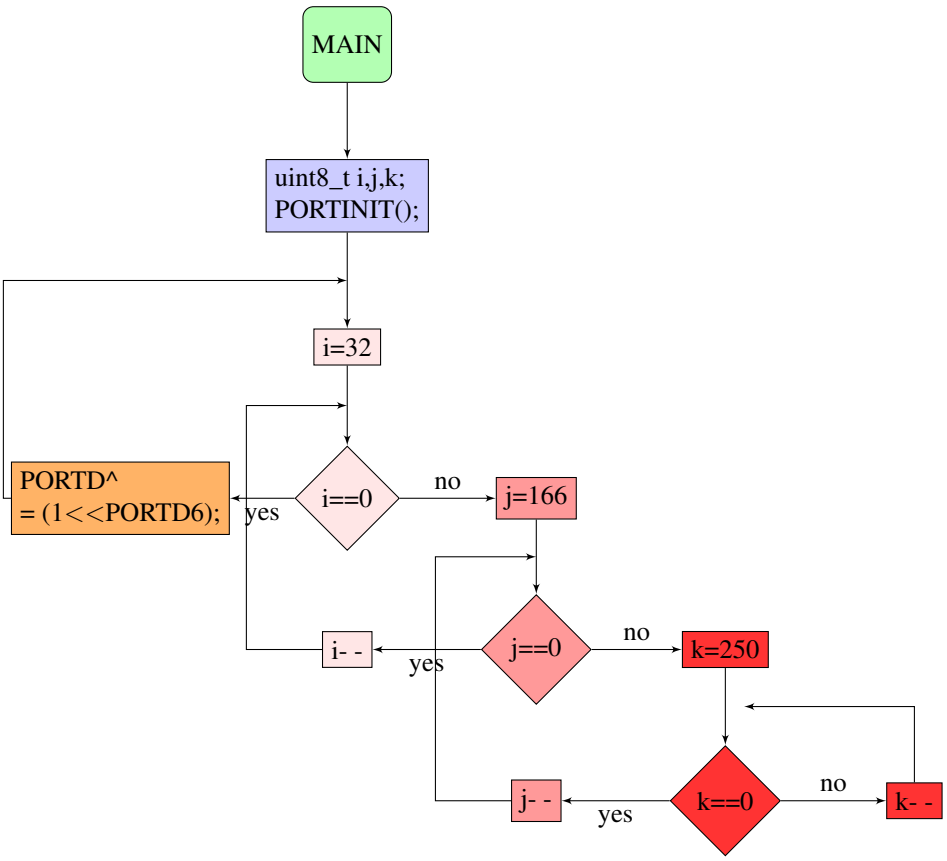
```
.INCLUDE “M88DEF.INC”
.ORG 0
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, (1<<6)
    LDI R17, (1<<6)
    SBI DDRD, 6
BACK:
    EOR R16, R17
    OUT PORTD, R16
    RCALL DELAY_HalfSec
    RJMP BACK
DELAY_HalfSec:
    LDI R20, 32
L1:     LDI R21, 100
L2:     LDI R22, 250
L3:
    NOP
    NOP
    DEC R22
    BRNE L3
      DEC R21
    BRNE L2
      DEC R20
    BRNE L1
      RET
;EOF
```



Cada ciclo de maquina demora $1/8Mhz$ que é igual a 125 nano segundos.
$Delay = 32 \times 100 \times 250 \times 5 \times 125ns \quad \Longrightarrow \quad Delay = 500ms$

4.2 Led pisca 1 Hz em C por Software.

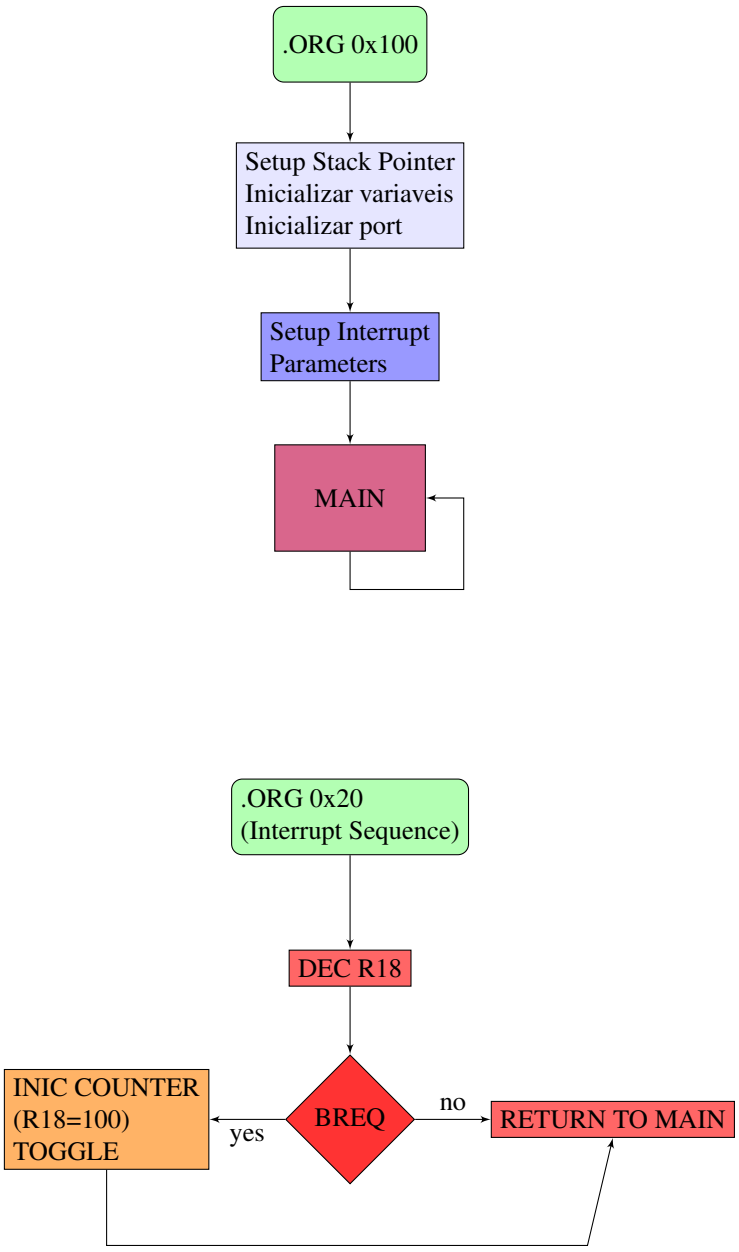
```
/***PreProcessor***/
#ifndef F_CPU
    #define F_CPU 8000000UL
#endif
/***Library***/
#include <avr/io.h >
#include <avr/interrupt.h >
#include <util/delay.h >
#include <avr/pgmspace.h >
/***Define and Macro***/
#define TRUE 1
/***Global Variable***/
/***Prototype***/
void PORTINIT(void);
/***MAIN***/
int main(void)
{
    uint8_t i,j,k;
    PORTINIT();
    while(TRUE)
    {
        for(i=32;i;i- -){
            for(j=166;j;j- -){
                for(k=250;k;k- -);
            }
        }
        PORTD^= (1<<PORTD6);
    }
}
/***Procedure and Function***/
void PORTINIT(void){
    DDRD=(1<<PORTD6);
    PORTD=(1<<PORTD6);
}
/***Interrupt***/
/***EOF***/
```



Os valores de i, j e k, foram do exercício anterior que a posterior foi ajustado de forma a obter a frequência desejada, com o auxilio de um osciloscópio.

4.3 Led pisca 1 Hz em Assembly por Interrupção.

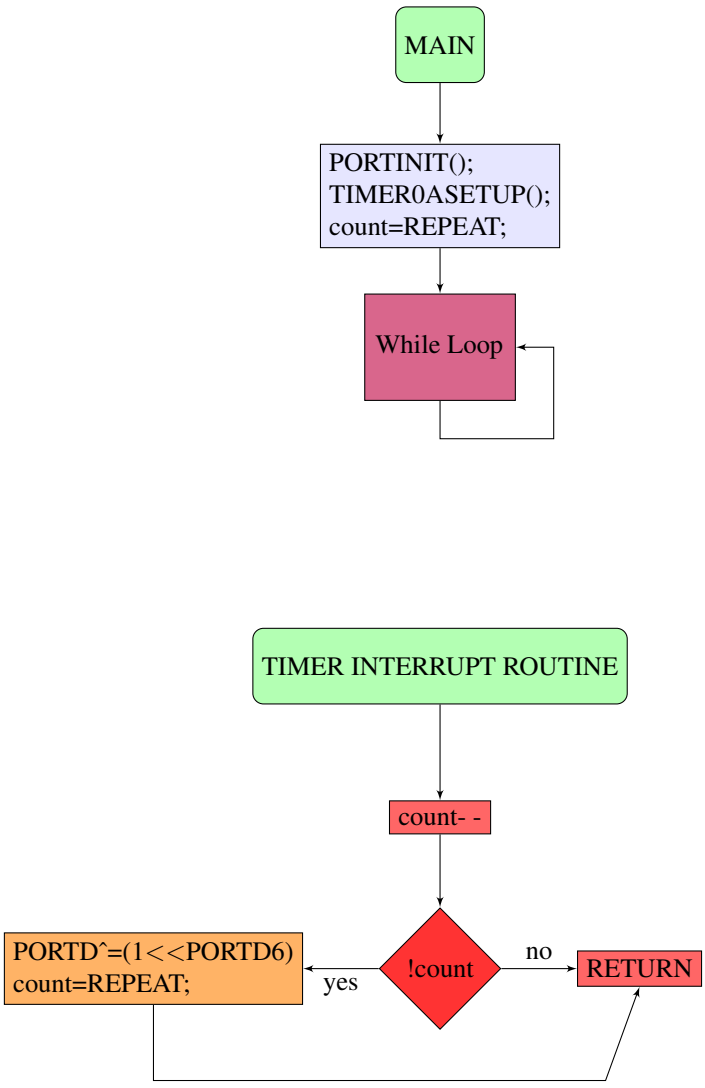
```
.EQU REPEAT = 100
.EQU MASK = (1<<6)
.INCLUDE “M88DEF.INC”
.ORG 0
    RJMP RESET
.ORG 0x20
    RJMP TIM0_COMPA
.ORG 0x100
RESET:
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, MASK
    LDI R17, MASK
    LDI R18, REPEAT
    SBI DDRD, 6
    SBI PORTD, 6
    RCALL INTRUP_10ms
    RJMP MAIN

; Setup Periodo 10ms
INTRUP_10ms:
    LDI R19, (1<<OCIE0A)
    STS TIMSK0, R19
    LDI R19, 156
    OUT OCR0A, R19
    LDI R19, (1<<WGM01)
    OUT TCCR0A, R19
    LDI R19, (1<<CS02)
    OUT TCCR0B, R19
    SEI

MAIN:
    RJMP MAIN

INIC:
    LDI R18, REPEAT
    EOR R16, R17
        OUT PORTD, R16

TIM0_COMPA:
    DEC R18
    BREQ INIC
    RETI
;EOF
```



Usando Parâmetros TIMSK0 com OCIE0A activado, wavegenmode=CTC, OCR0A=156 e por ultimo N=256, para obter um Período de 10ms com oscilação por cristal $F_{clk\_I/O} = 8Mhz$.

4.4 Led pisca 1 Hz em C por Interrupção.

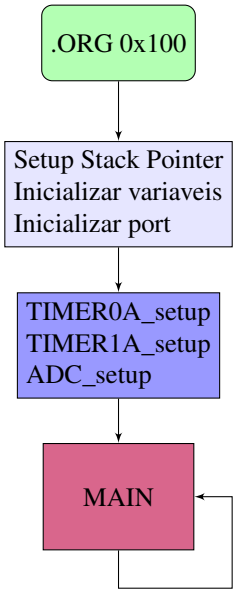
```
/***PreProcessor***/
#ifndef F_CPU
        #define F_CPU 8000000UL
#endif
/***Library***/
#include <avr/interrupt.h >
#include <util/delay.h >
#include <avr/io.h >
#include <avr/pgmspace.h >
/***Define and macro***/
#define TRUE 1
#define REPEAT 100
/***Global variable***/
int count;
/***Prototype***/
void PORTINIT(void);
void TIMER0ASETUP(void);
/***MAIN***/
int main(void)
{
    PORTINIT();
    TIMER0ASETUP();
    count = REPEAT;
    while(TRUE)
    {

    }
}
/***Procedure and function***/
void PORTINIT(void){
    DDRD = (1<<PORTD6);
    PORTD = (1<<PORTD6);
}
void TIMER0ASETUP(void){
    uint8_t sreg;
    sreg = SREG;
    cli();
/***Periodo de 10ms***/
    TCCR0A = (1<<WGM01);
    TIMSK0 = (1<<OCIE0A);
    OCR0A = 156;
    TCCR0B |= (1<<CS02);
    SREG = sreg;
    sei();
}
/***Interrupt***/
ISR(TIMER0_COMPA_vect){
    count- -;
    if(!count){
        PORTD^= (1<<PORTD6);
        count = REPEAT;
    }
}
/***EOF***/
```

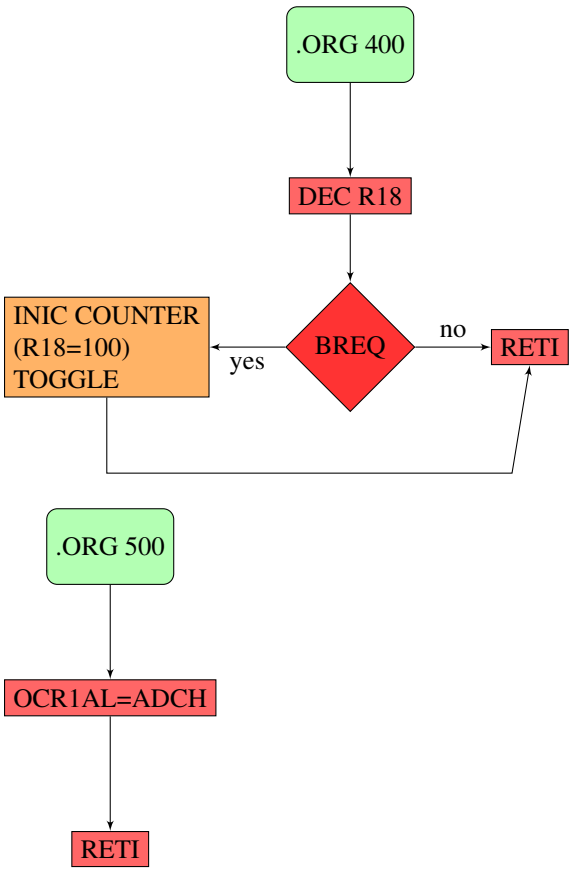


100× Periodo de 10ms $\implies$ 1sec de Periodo ou 1Hz

4.5 Código DIMMER e led Pisca em Assembler

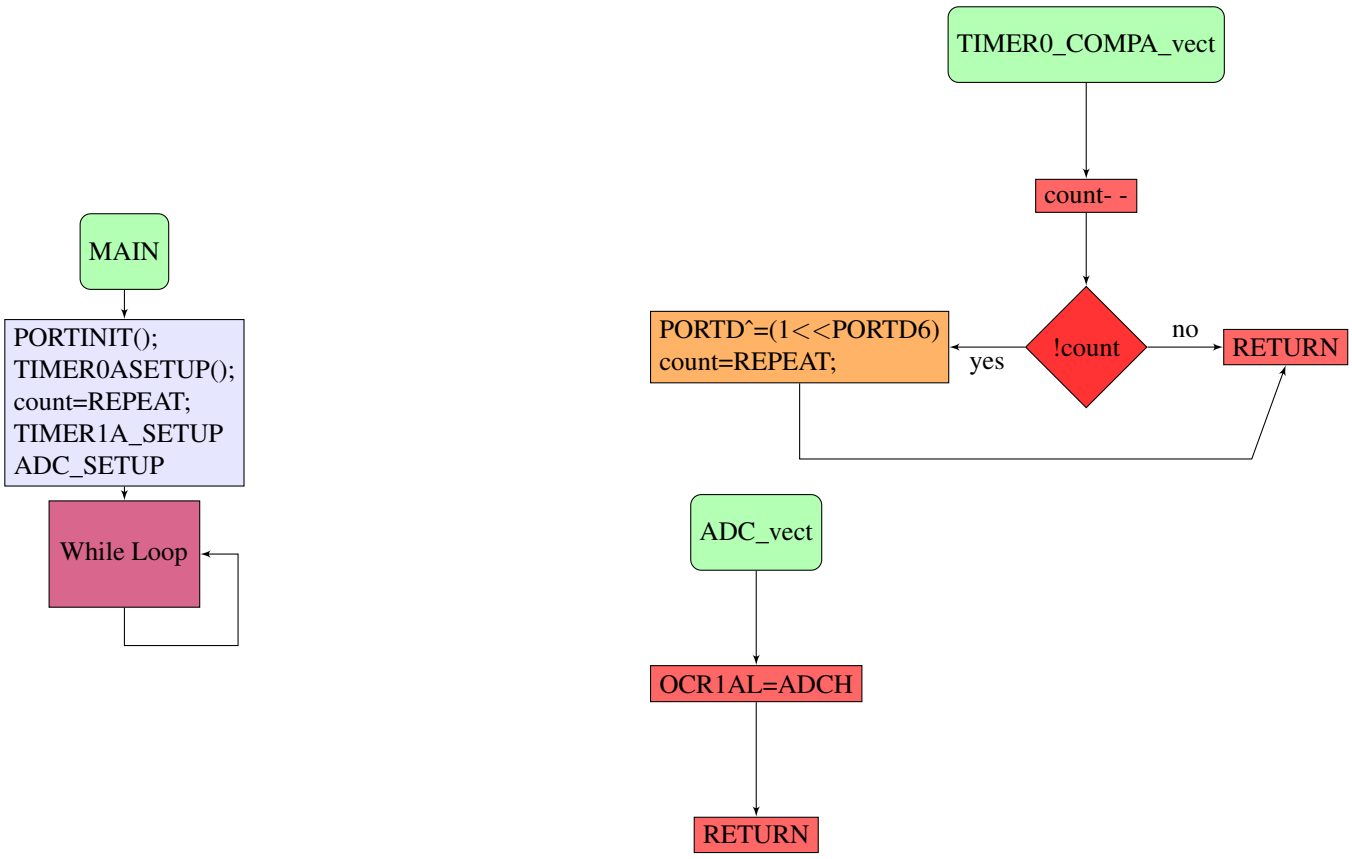
```
;assembly code
.EQU REPEAT = 100
.EQU MASK = (1<<PD6)
.INCLUDE “M88DEF.INC”
.ORG 0
        RJMP RESET
.ORG 0x015
        RJMP ADC_vect
.ORG 0X00E
        RJMP TIM0_COMPA_vect
.ORG 0x100
RESET:
        LDI R16, HIGH(RAMEND)
        OUT SPH, R16
        LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R16, MASK
        LDI R17, MASK
        LDI R18, REPEAT
        SBI DDRD, 6
        CBI DDRB, (1<<PB1)
        CBI PORTD, 6
        RCALL TIMER0A_setup
        RCALL TIMER1A_setup
        RCALL ADC_setup
        RJMP MAIN

;— 10ms Setup
TIMER0A_setup:
        LDI R19, (1<<OCIE0A)
        STS TIMSK0, R19
        LDI R19, 156 ;OCR0A
        OUT OCR0A, R19
        LDI R19, (1<<WGM01)
        OUT TCCR0A, R19
        LDI R19, (1<<CS02)
        OUT TCCR0B, R19
        RET
TIMER1A_setup:
        LDI R19, (1<<WGM10)
        ORI R19, (3<<COM1A0)
        STS TCCR1A, R19
        LDI R19, (1<<WGM12)
        STS TCCR1B, R19
        SBI DDRB, PB1
        LDI R19, 128
        STS OCR1AL, R19
        LDS R19, TCCR1B
        ORI R19, (1<<CS12)
        STS TCCR1B, R19
        RET

ADC_setup:
        CBI DDRC, PC0
        LDI R19, (1<<REFS0)
        ORI R19, (1<<ADLAR)
        STS ADMUX, R19
        LDI R19, (7<<ADPS0)
        ORI R19, (1<<ADIE)
        ORI R19, (1<<ADEN)
        ORI R19, (1<<ADSC)
        ORI R19, (1<<ADATE)
        STS ADCSRA, R19
        SEI
        RET

MAIN:
        RJMP MAIN

.ORG 400
TIM0_COMPA_vect:
        DEC R18
        BREQ INIC
        RETI
INIC:
        LDI R18, REPEAT
        EOR R16, R17
        OUT PORTD, R16

.ORG 500
ADC_vect:
        LDS R20, ADCL
        LDS R20, ADCH
        STS OCR1AL, R20
        RETI
;EOF
```


.ORG 0x100
Setup Stack Pointer
Inicializar variaveis
Inicializar port
TIMER0A_setup
TIMER1A_setup
ADC_setup
MAIN


.ORG 400
DEC R18
BREQ
yes
no
INIC COUNTER
(R18=100)
TOGGLE
RETI
.ORG 500
OCR1AL=ADCH
RETI

4.6 Código DIMMER e led Pisca em C

```
/***Pre Processor***/
#ifndef F_CPU
	#define F_CPU 8000000UL
#endif
/***Library***/
#include <avr/io.h >
#include <avr/interrupt.h >
#include <util/delay.h >
#include <avr/pgmspace.h >
/***Define and Macro***/
#define TRUE 1
#define REPEAT 100
/***Global variable***/
static volatile uint16_t adcL_tmp;
static volatile uint16_t adcH_tmp;
int count;
/***Prototype***/
void PORTINIT(void);
/******/
void TIMER0A_SETUP(void);
/******/
void TIMER1A_SETUP(uint8_t wagenmode, uint8_t compoutAmode,uint8_t interruptmask);
void TIMER1A_STARTSTOP(uint16_t prescaler);
void TIMER1A_trigger(uint16_t Atrigger);
void TIMER1_intcapture(uint16_t capture);
/******/
void ADC_ENABLE(uint8_t ADC_channel, uint8_t ADC_Vref, uint8_t ADC_lar, uint8_t ADC_Div);
/***MAIN_MAIN_MAIN***/
int main(void)
{
	//PORTINIT();
	/******/
	TIMER0A_SETUP();
	count=REPEAT;
	/******/
	TIMER1A_SETUP(5,2,0);
	TIMER1A_trigger(128);
	TIMER1A_STARTSTOP(256);
	/******/
	ADC_ENABLE(0, 1, 1, 128);
	while(TRUE)
	{
	}
}
/***Procedure and Function***/
void PORTINIT(void)
{
}
/*****/
void TIMER0A_SETUP(void)
{
	DDRD=(1<<PORTD6);
	/**T=10ms**/
	TCCR0A=(1<<WGM01);
	TIMSK0=(1<<OCIE0A);
	OCR0A=156;
	TCCR0B|=(1<<CS02);
}
/*****/
void TIMER1A_SETUP(uint8_t wavegenmode, uint8_t compoutAmode,uint8_t interruptmask)
{
	uint8_t TCCR1A_tmp=0x00;
	uint8_t TCCR1B_tmp=0x00;
	uint8_t TIMSK1_tmp=0x00;
	switch(wavegenmode){
		case 0:
			break;
		case 1:
			TCCR1A_tmp=(1<<WGM10);
			break;
		case 2:
			TCCR1A_tmp=(1<<WGM11);
			break;
		case 3:
			TCCR1A_tmp=(3<<WGM10);
			break;
		case 4:
			TCCR1B_tmp=(1<<WGM12);
			break;
		case 5:
			TCCR1A_tmp=(1<<WGM10);
			TCCR1B_tmp=(1<<WGM12);
			break;
		case 6:
			TCCR1A_tmp=(1<<WGM11);
			TCCR1B_tmp=(1<<WGM12);
			break;
		case 7:
			TCCR1A_tmp=(3<<WGM10);
			TCCR1B_tmp=(1<<WGM12);
			break;
		case 8:
			TCCR1B_tmp=(1<<WGM13);
			break;
		case 9:
			TCCR1A_tmp=(1<<WGM10);
			TCCR1B_tmp=(1<<WGM13);
			break;
		case 10:
			TCCR1A_tmp=(1<<WGM11);
			TCCR1B_tmp=(1<<WGM13);
			break;
		case 11:
			TCCR1A_tmp=(3<<WGM10);
			TCCR1B_tmp=(1<<WGM13);
			break;
		case 12:
			TCCR1B_tmp=(3<<WGM12);
			break;
		case 14:
			TCCR1A_tmp=(1<<WGM11);
			TCCR1B_tmp=(3<<WGM12);
			break;
		case 15:
			TCCR1A_tmp=(3<<WGM10);
			TCCR1B_tmp=(3<<WGM12);
			break;
		default:
			break;
	};
	switch(compoutAmode){
		case 0:
			break;
		case 1:
			TCCR1A_tmp |=(1<<COM1A0);
			break;
		case 2:
			TCCR1A_tmp |=(1<<COM1A1);
			break;
		case 3:
			TCCR1A_tmp |=(3<<COM1A0);
			break;
		default:
			break;
	};
	switch(interruptmask){
		case 0:
			break;
		case 1:
			TIMSK1_tmp=(1<<TOIE1);
			break;
		case 2:
			TIMSK1_tmp=(1<<OCIE1A);
			break;
		case 3:
			TIMSK1_tmp=(3<<TOIE1);
			break;
		case 4:
			TIMSK1_tmp=(1<<OCIE1B);
			break;
		case 5:
			TIMSK1_tmp=((1<<OCIE1B) |(1<<TOIE1));
			break;
		case 6:
			TIMSK1_tmp=((3<<OCIE1A));
			break;
		case 7:
			TIMSK1_tmp=(7<<TOIE1);
			break;
		case 8:
			TIMSK1_tmp=(1<<ICIE1);
			break;
		case 9:
			TIMSK1_tmp=((1<<ICIE1) |(1<<TOIE1));
			break;
		case 10:
			TIMSK1_tmp=((1<<ICIE1) |(1<<OCIE1A));
			break;
		case 11:
			TIMSK1_tmp=((1<<ICIE1) |(3<<TOIE1));
			break;
		case 12:
			TIMSK1_tmp=((1<<ICIE1) |(1<<OCIE1B));
			break;
		case 13:
			TIMSK1_tmp=((1<<ICIE1) |(1<<OCIE1B) |(1<<TOIE1));
			break;
		case 14:
			TIMSK1_tmp=((1<<ICIE1) |(3<<OCIE1A));
			break;
		case 15:
			TIMSK1_tmp=((1<<ICIE1) |(7<<TOIE1));
			break;
		default:
			break;
	};
	TCCR1A=TCCR1A_tmp;
	TCCR1B=TCCR1B_tmp;
	if(compoutAmode){
		DDRB |=(1<<PB1);
	}
	TIMSK1=TIMSK1_tmp;
}
void TIMER1A_STARTSTOP(uint16_t prescaler)
{
	uint8_t TCCR1B_tmp=0x00;
	switch(prescaler){
		case 0:
			TCCR1B&=~((1<<CS12) |(1<<CS11) |(1<<CS10));
			break;
		case 1:
			TCCR1B_tmp=(1<<CS10);
			break;
		case 8:
			TCCR1B_tmp=(1<<CS11);
			break;
		case 64:
			TCCR1B_tmp=(3<<CS10);
			break;
		case 256:
			TCCR1B_tmp=(1<<CS12);
			break;
		case 1024:
			TCCR1B_tmp=((1<<CS12) |(1<<CS10));
			break;
		case 2:
			TCCR1B_tmp=(3<<CS11);
			break;
		case 3:
			TCCR1B_tmp=((1<<CS12) |(1<<CS10));
			break;
		default:
			TCCR1B_tmp=(7<<CS10);
	};
	TCCR1B|=TCCR1B_tmp;
}
void TIMER1A_trigger(uint16_t Atrigger)
{
	OCR1A=Atrigger;
}
void TIMER1_intcapture(uint16_t capture)
{
	ICR1=capture;
}
/*****/
void ADC_ENABLE(uint8_t ADC_channel, uint8_t ADC_Vref, uint8_t ADC_lar, uint8_t ADC_Div)
{
	uint8_t ADMUX_tmp;
	uint8_t ADCSRA_tmp;
	switch(ADC_channel){
		case 0:
			DDRC&=~(1<<PC0);
			ADMUX_tmp=0x00;
			break;
		case 1:
			DDRC&=~(1<<PC1);
			ADMUX_tmp=0x01;
			break;
		case 2:
			DDRC&=~(1<<PC2);
			ADMUX_tmp=0x02;
			break;
		case 3:
			DDRC&=~(1<<PC3);
			ADMUX_tmp=0x03;
			break;
		case 4:
			DDRC&=~(1<<PC4);
	ADMUX_tmp=0x04;
			break;
		case 5:
			DDRC&=~(1<<PC5);
			ADMUX_tmp=0x05;
			break;
		case 6:
			ADMUX_tmp=0x06;
			break;
		case 7:
			ADMUX_tmp=0x07;
			break;
		default:
			ADMUX_tmp=0x00;
	};
	switch(ADC_Vref){
		case 0:
			break;
		case 1:
			ADMUX_tmp |=(1<<REFS0);
			break;
		case 3:
			ADMUX_tmp |=(3<<REFS0);
			break;
		default:
			ADMUX_tmp |=(1<<REFS0);
	};
	switch(ADC_lar){
		case 0:
			break;
		case 1:
			ADMUX_tmp |=(1<<ADLAR);
			break;
		default:
			break;
	};
	switch(ADC_Div){
		case 1:
			ADCSRA_tmp=(1<<ADPS0);
			break;
		case 2:
			ADCSRA_tmp=(1<<ADPS0);
			break;
		case 4:
			ADCSRA_tmp=(1<<ADPS1);
			break;
		case 8:
			ADCSRA_tmp=(3<<ADPS0);
			break;
		case 16:
			ADCSRA_tmp=(1<<ADPS2);
			break;
		case 32:
			ADCSRA_tmp=(5<<ADPS0);
			break;
		case 64:
			ADCSRA_tmp=(3<<ADPS1);
			break;
		case 128:
			ADCSRA_tmp=(7<<ADPS0);
			break;
		default:
			ADCSRA_tmp=(7<<ADPS0);
			break;
	};
	ADMUX=ADMUX_tmp;
	ADCSRA_tmp |=(1<<ADIE);
	ADCSRA_tmp |=(1<<ADEN);
	ADCSRA_tmp |=(1<<ADSC);
	ADCSRA_tmp |=(1<<ADATE);
	ADCSRA=ADCSRA_tmp;
	ADCSRB&=~(7<<ADTS0);
	sei();
}
/***Interrupt***/
ISR(TIMER0_COMPA_vect){
	uint8_t sREG=SREG;
	cli();
	count- -;
	if(!count){
		PORTD ^=(1<<PORTD6);
		count=REPEAT;
	}
	SREG=sREG;
	sei();
}
/******/
ISR(ADC_vect)
{
	adcL_tmp=ADCL;
	adcH_tmp=ADCH;
	adcL_tmp=adcL_tmp |(adcH_tmp<<8);
	TIMER1A_trigger(adcH_tmp);
}
/***EOF***/
```



Neste caso exagerei um bocado pois fiz umas funções genérica para preencher os respetivos registos de configuração.

5 Resultados

No geral foi conseguido os objetivos do relatório, o manuseamento e parametrização do Atmega88 com o auxilio do seu datasheet, e a ajuda das ferramentas Atmel ICE e o Atmel Studio 7. Foi de forma a ter em conta para o código ser percetível e ao mesmo tempo amigo do utilizador, criado uma abstração simples e lógica na perspetiva humana, de fácil acompanhamento.
Na parte de hardware a pouco a dizer devido aos circuitos utilizados já serem muito conhecidos na electrónica, e se surgir duvidas facilmente com pesquisa há centenas de fontes de informção ao dispor como a Internet ou literatura.

6 Conclusões

O assembly é maquina dependente e de programação detalhada muito ligada ao hardware, sua programação em esparguete que não é aconselhável na linguagem C como por exemplo a instrução **goto** em **C**, da origem a código difícil de seguir e se perceber. O **C** é mais liberal maquina independente, mas o utilizador não tem controle sobre o mapeamento das variáveis na memoria nem na sua compilação. A linguagem também é consebido com uma estrutura mais lógica sendo sua transferência para **Flowchart** intuitiva.

Fazendo **Delays** ou **Polling** interrompe o programa ficando a espera, dai as interrupções são mais úteis, só chamando sua rotina quando necessário.

O domínio de interpretação e manipulação do datasheet em conjunto com seu component cosidero como objetivo prioritário, perceber seus conceitos e modos de funcionamento, interpretar os esquemas electrónicos e sua interligação com a parte de programação, suas definições e características ter uma visão do que é possível fazer da combinação de hardware e software, tendo em conta suas limitações e virtudes, podemos com esta aprendizagem facilmente nos adaptar a outras arquiteturas e fabricantes, o hardware em principio será igual o modo de o manipular por software o paradigma de aproximação poderá ser diferente mas a eletrónica em principio é universal.

Este trabalho esta publicado no github, link abaixo:
***h*ttps://github.com/sergio1020881/LABSIS20202021**

Definições

Definição 1 *Capacitância*

$$\begin{aligned} Q_c(t) &= \int^t i(t) \quad dt \\ &= Q_c(0^-) + \int_{0^-}^t i(t) \quad dt \\ V_c(t) &= \frac{Q_c(t)}{C} \\ &= \frac{1}{C} \quad \int^t i_c(t) \quad dt \\ &= \frac{Q_c(0^-)}{C} + \frac{1}{C} \quad \int_0^t i_c(t) \quad dt \\ &= V(0^-) + \frac{1}{C} \quad \int_0^t i_c(t) \quad dt \\ i_c(t) &= C \quad \frac{dV_c(t)}{dt} \end{aligned}$$

Definição 2 *Indutância*

$$\begin{aligned} \psi_L(t) &= \int^t V_L(t) \quad dt \\ &= \psi_L(0^-) + \int_{0^-}^t V_L(t) \quad dt \\ V_L(t) &= L \quad \frac{di_L(t)}{dt} \\ i_L(t) &= \frac{\psi_L(t)}{L} \\ &= \frac{1}{L} \quad \int^t V_L(t) \quad dt \\ &= \frac{\psi_L(0^-)}{L} + \frac{1}{L} \quad \int_0^t V_L(t) \quad dt \\ &= i_L(0^-) + \frac{1}{L} \quad \int_0^t V_L(t) \quad dt \end{aligned}$$

Definição 3 *Resistência*

$$\begin{aligned} V_R(t) &= R \quad i_R(t) \\ i_R(t) &= \frac{V_R(t)}{R} \end{aligned}$$

Definição 4 *Valor Médio*

$$X_{av} = \frac{1}{T} \int_0^T X(t)dt$$

Definição 5 *Valor Eficaz*

$$X_{ef} = \sqrt{\frac{1}{T} \int_0^T X^2(t)dt}$$

[]

Bibliografia

[1] *The C Programming Language*. Prentice Hall, •.

[2] *Curso de Introdução ao LATEX*.

[3] *Teach Yourself Electricity and Electronics*. McGraw-Hill, •.

[4] *Electrónica Analógica*. McGraw Hill, 1993.

[5] *Cálculo Diferencial e Integral em* $\mathbb{R}$ *e* $\mathbb{R}^n$. McGraw Hill, 1995.

[6] *Power Electronic Converter Harmonics*. IEEE Press Editorial Board, 1996.

[7] *Power Electronic Control in Electrical Systems*. Newnes Power Engeneering Series, 2002.

[8] *The Maxima Book*. •, 2004.

[9] *C Programming for Microcontrollers*. SMILEY MICROS, 2005.

[10] *HIGHER ENGINEERING MATHEMATICS*. Published by Elsevier Ltd, 2006.

[11] *PSIM® User's Guide*. Powersim Inc., 2009.

[12] *the avr microcontroller and embedded systems*. Prentice Hall, 2011.

[13] Fidalgo, André: *Sistemas Eléctricos de Corrente Alternada*. Em *Unidade de Ensino 2*.

[14] Fidalgo, André: *Sistemas Eléctricos de Corrente Contínua*. Em *Unidade de Ensino 1*.

1

[1] Apontamentos