Instituto Superior de Engenharia do Porto

Sistema Mínimo

Aluno :
Sérgio Santos, *Nº*: 1020881

Docente/Orientador
Lino Manuel Baptista Figueiredo, *lbf*

Unidade Curricular
LABSIS

Sistema Simplificado

1 Introdução

Este relatório tem como objetivo a introdução aos microcontroladores da AVR começando pelo "Meu Primeiro Programa ", que é por um **LED** (light emitting diode) a piscar no mundo dos **MCU** (Microcontroller Unit).

Vai se recorrer ao Assembler e Linguagem C para cumprir estas tarefas, a temporização vai primeiro ser feita por software e de seguida por intermédio de interrupções, á frequência de oscilação de $1Hz$.

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. É ligado uma ficha ISP aos respectivos pinos para programação e debug.
Para Facilitar seu desenvolvimento foi feito numa placa pre furada sua implementação de forma a ser alimentada diretamente ao PC pela alimentação da porta USB.

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 1Mhz, BOOTZ=1024W_0C00, SPIEN=ON, BODLEVEL=2V7, SUT_CKSEL=EXTXOSC_8MHZ_XX_16KCK_14CK_65MS , EXTENDED FUSE= 0xF9,HIGH FUSE=0xDD, LOW FUSE=0xFF com os restantes bits OFF.

4 Software

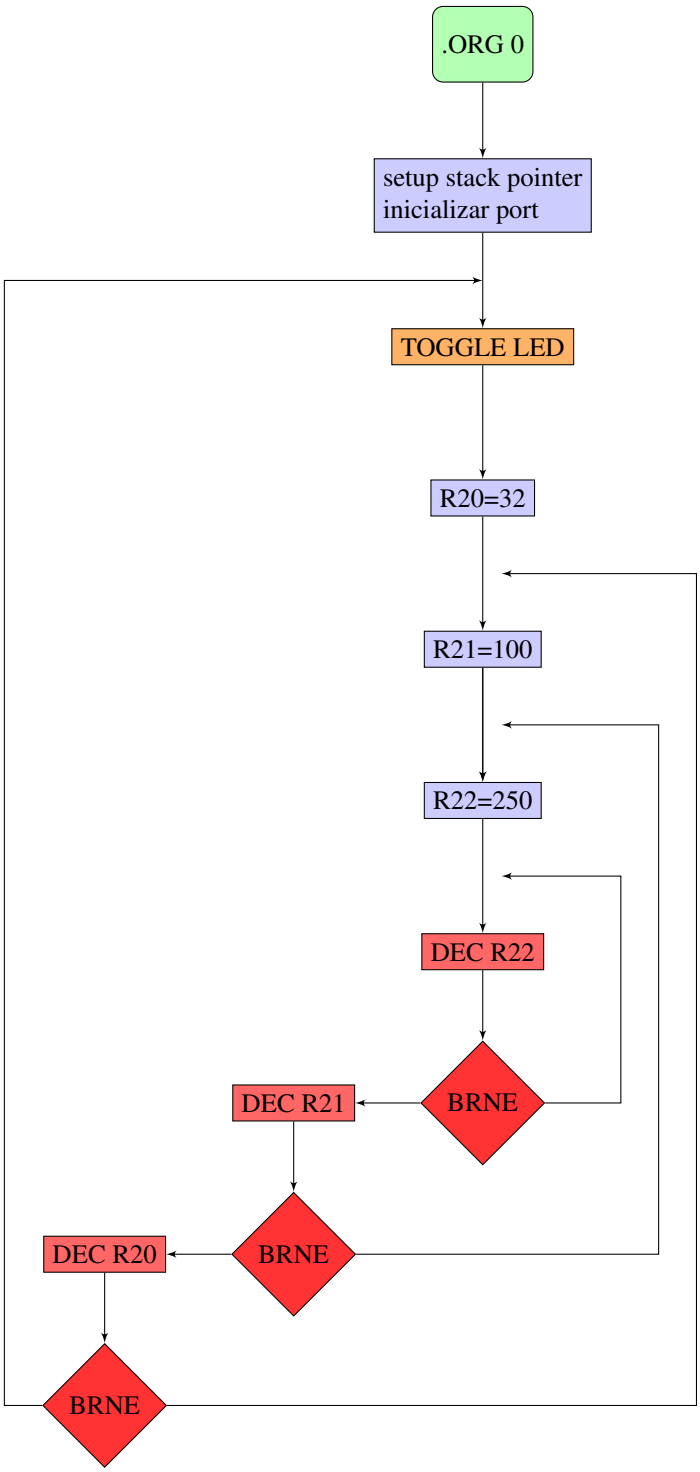
Nesta secção é feito o código correspondente em por um LED a piscar a 1Hz no PORTD6, nos casos mencionados na introdução.

A ferramenta de desenvolvimento (**IDE**) utilizado é o Atmel Studio 6 (versão 6.2)
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.

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

4.1 Led Pisca 1Hz em assembly por software.

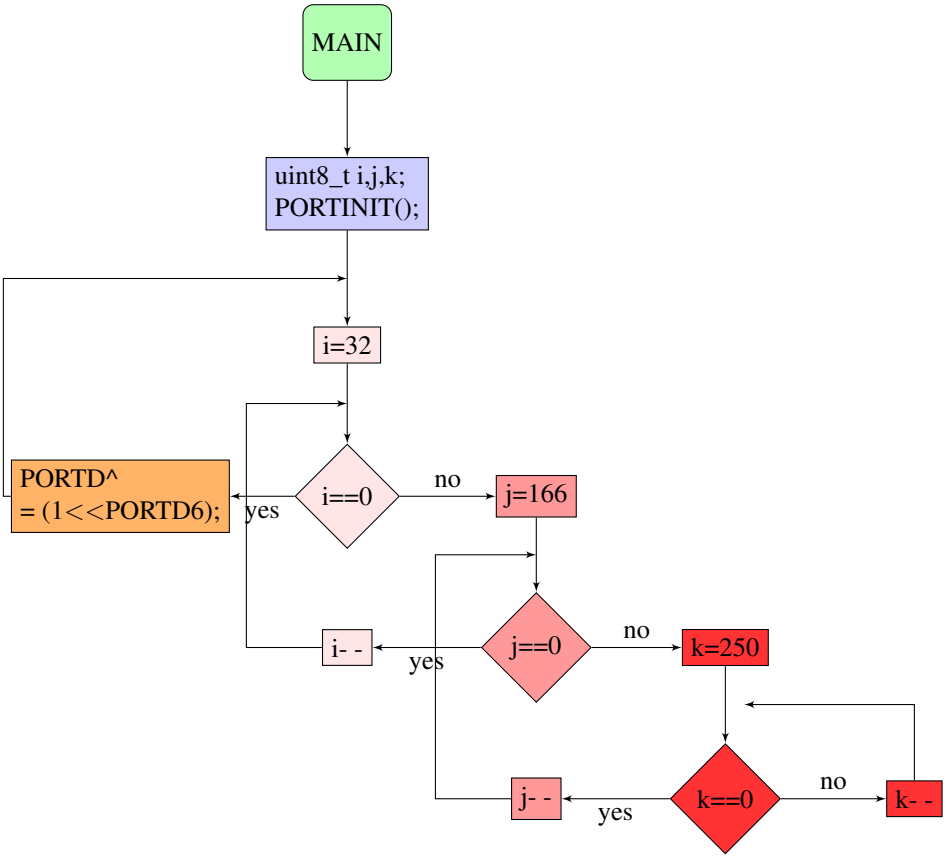
```
.INCLUDE “M88DEF.INC”
.ORG 0
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, (1<<6)
    LDI R17, (1<<6)
    SBI DDRD, 6
BACK:
    EOR R16, R17
    OUT PORTD, R16
    RCALL DELAY_HalfSec
    RJMP BACK
DELAY_HalfSec:
    LDI R20, 32
L1:     LDI R21, 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.

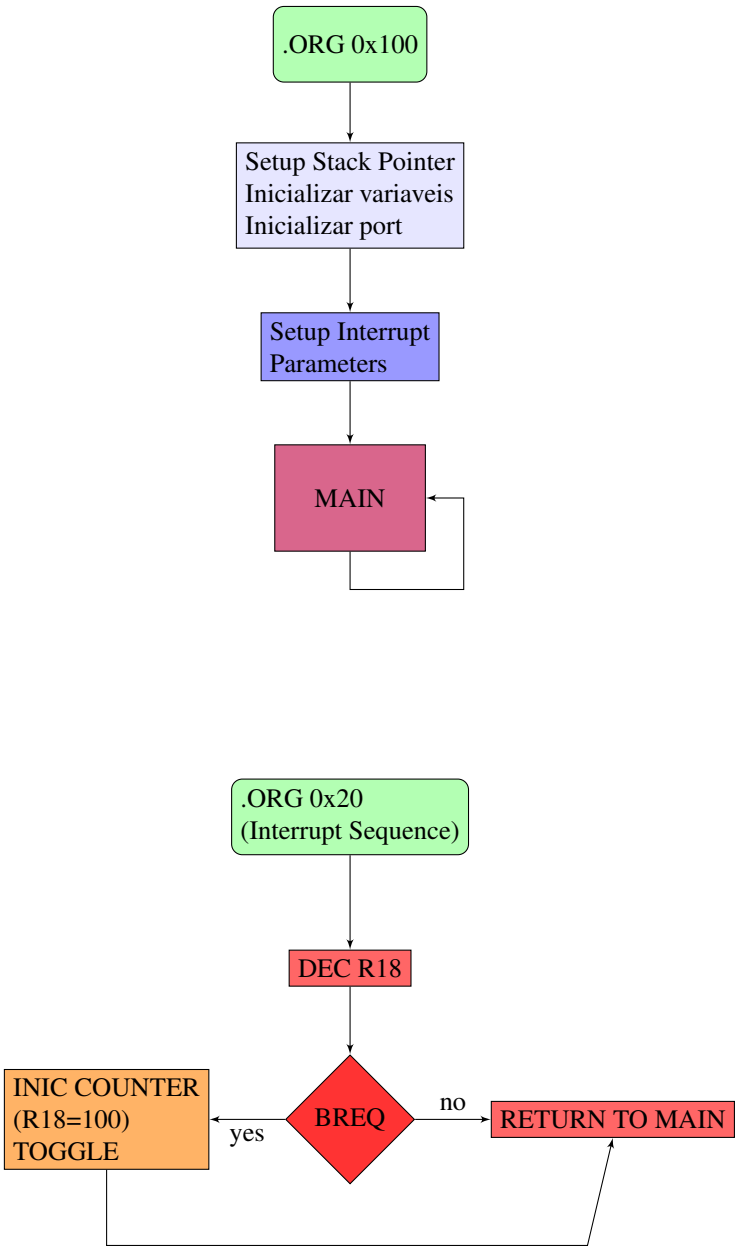
```
/***PreProcessor***/
#ifndef F_CPU
    #define F_CPU 8000000UL
#endif
/***Library***/
#include <avr/io.h >
#include <avr/interrupt.h >
#include <util/delay.h >
#include <avr/pgmspace.h >
/***Define and Macro***/
#define TRUE 1
/***Global Variable***/
/***Prototype***/
void PORTINIT(void);
/***MAIN***/
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.

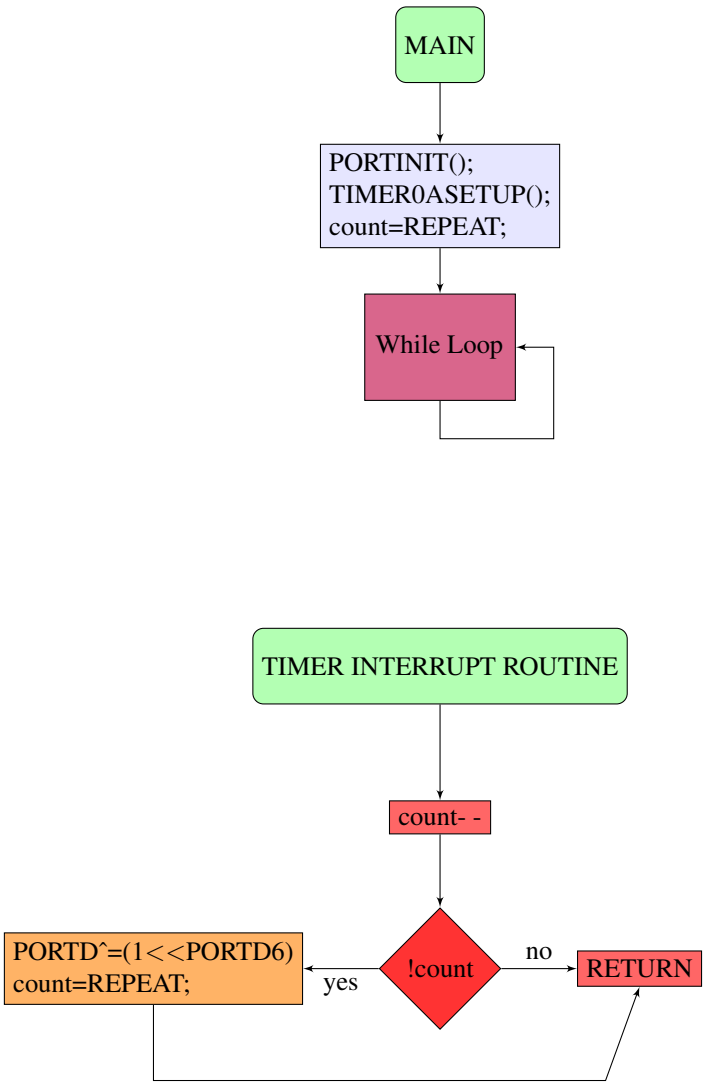
```
.EQU REPEAT = 100
.EQU MASK = (1<<6)
.INCLUDE “M88DEF.INC”
.ORG 0
    RJMP RESET
.ORG 0x20
    RJMP TIM0_COMPA
.ORG 0x100
RESET:
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, MASK
    LDI R17, MASK
    LDI R18, REPEAT
    SBI DDRD, 6
    SBI PORTD, 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 $\Longrightarrow$ 1sec de Periodo ou 1Hz

5 Resultados

No geral foi conseguido os objetivos do relatório, o manuseamento e parametrização do Atmega88 com o auxilio do datasheet é uma ferramenta importante, tendo também um bom debugger ajuda bastante no troubleshooting e deteção de problemas em conjunto com o Atmega Studio 6.

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 avariáveis na memoria nem na sua compilação. A linguagem também é consebido com uma estrutura mais lógica sendo sua transferência para **Flowchart** mais intuitiva e façil.

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

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