Bem, primeiramente venho pedir desculpas pelo imprevisto, sei que o erro foi totalmente meu ao deixar para gravar o vídeo de última hora, mas eu não imaginava que o meu maldito dente do siso resolveria nascer. Em resumo meu dente do siso resolveu nascer e não apenas satisfeito a nascer ele resolveu nascer deitado, o dentista teve mo trabalho de tirar ele, isso no sábado e ele ainda está inchado e não estou com uma dicção compreensível graças a ele. Mas voltando ao assunto principal, pro projeto tinha que apresentar/explicar o código em vídeo e não vou conseguir nesse momento fazer isso então estarei explicando por aqui o código e caso o senhor queira eu posso tentar gravar na terça ou até mesmo na segunda se estiver melhor, mas enfim era só pra explicar mesmo, mas estarei gravando a simulação no SimulIde.

* O código de divisão foi pego nesse link <https://sites.google.com/site/avrasmintro/home/2b-basic-math>

start:

ldi r16, 0xff

out ddrb, r16 🡪portas B como output

ldi r16, 0x00

out ddrc, r16 🡪portas C como input

call inicializarADC

call inicializarPWM

loop1:

call lerADC

lds r20, ADCL 🡪lendo registradores do ADC

lds r21, ADCH

ldi r16, 0x40 🡪carregando 64 para a divisão

ldi r17, 0x00

call DIV3216 🡪dividindo ADC por 64 (65536 / 64 = 1024)

sts OCR1AL, r0 🡪carregando resultado da divisão no limite do PWM

sts OCR1AH, r1

jmp loop1

inicializarPWM:

ldi r16, 0b10000011

ldi r17, 0b00001001

sts TCCR1A, r16 🡪PWM na porta OC1A

sts TCCR1B, r17 🡪prescaler 1024

ret

inicializarADC:

ldi r16, 0b01100000

sts admux, r16 🡪mux do adc na porta PC0

ldi r16, 0b10000000

sts adcsra, r16 🡪ativando ADC

ret

lerADC:

ldi r16, 0b11000000

sts adcsra, r16 🡪iniciar conversão ADC

lendoADC:

lds r16, adcsra

sbrs r16, 4 🡪pula a próxima instrução se a flag ADIF estiver setada (fim da conversão)

jmp lendoADC

ldi r16, 0b10010000

sts adcsra, r16 🡪reseta a flag para a próxima conversão

ret

DIV3216:

.DEF res1 = R0 🡪resposta em 32bits

.DEF res2 = R1 🡪byte 1 (\*256)

.DEF res3 = R2 🡪byte 2 (\*256\*256)

.DEF res4 = R3 🡪byte 3 (\*256\*256\*256)

.DEF REM1 = R4 🡪resto em 32bits

.DEF REM2 = R5

.DEF REM3 = R6

.DEF REM4 = R7

.DEF dZERO = R8 🡪zero para operações de carry

.DEF A1 = R20 🡪dividendo em 32bits

.DEF A2 = R21

.DEF A3 = R22

.DEF A4 = R23

.DEF dL = R16 🡪low byte do divisor

.DEF dH = R17 🡪high byte do divisor

.DEF C = R22 🡪contador para o loop

CLR dZERO

MOVW res2:res1,A2:A1

MOVW res4:res3,A4:A3

LDI C,33 🡪carrega 33 no contador (1+ num de bits)

CLR REM1 🡪zerando resto

CLR REM2

CLR REM3

CLR REM4

LOOP: ROL res1 🡪passando a resposta uma casa para a esquerda

ROL res2

ROL res3

ROL res4

DEC C 🡪decrementando contador

BREQ DONE 🡪se o contador é 0, então todos os bits foram processados

ROL REM1

ROL REM2

ROL REM3

ROL REM4

SUB REM1,dL 🡪subtraindo divisor do resto

SBC REM2,dH

SBC REM3,dZERO

SBC REM4,dZERO

BRCC SKIP 🡪branch se o resultado for negativo

ADD REM1,dL 🡪somando o divisor no resto

ADC REM2,dH

ADC REM3,dZERO

ADC REM4,dZERO

CLC

RJMP LOOP

SKIP: SEC

RJMP LOOP

DONE:

ret