Rampa eletrônica de lava carros feito com pic16F628A

Gabriel Machado[1], Juliana Batista[1], Iago Costa das Flores[1]

[1]Faculdade de Computação e Engenharia Elétrica – Universidade Federal do Sul e Sudeste do Pará (UNIFESSPA)
Marabá – PA – Brazil

{univercomput,juliana.batista, gabriel.machado}@unifesspa.edu.br

Resumo. *Este relatório tem como objetivo, descrever de forma detalhada o funcionamento de um código escrito em assembly, com a finalidade de simular uma catraca elétrica de estacionamento, onde um dos motores é responsável por fazer o dispositivo abrir, junto de um led verde como indicativo, e outro motor para fechar a catraca, também com um led, na cor vermelha, como indicativo. O código em assembly foi utilizado para programar o hardware descrito acima, com o auxílio do programa Proteus.*

Palavras-chave—Assembly, pic, programação.

1. Introdução

O projeto foi elaborado para simular uma catraca eletrônica, a lógica da aplicação foi implementada utilizando o TIMER0 como temporizador, assim quando um dos botões correspondentes a um dos motores for pressionado, o motor em questão irá ficar ativo por um certo tempo, tempo esse, que foi definido pelo TIMER0.

O código feito em assembly foi escrito e compilado utilizando o programa MPLAB, após a compilação do script, um arquivo .hex foi gerado e utilizado no programa Proteus, onde o hardware para o funcionamento do programa foi esquematizado com 2 motores, 2 leds, um pic16f628a, 4 resistores, 2 botões, 1 cristal e 2 capacitores, após realizar as devidas ligações, a simulação no Proteus foi iniciada para que fosse possível observar o funcionamento do hardware programado em assembly.

2. Fundamentação Teórica

Para os microcontroladores e microprocessadores exercerem determinadas atividades é necessário um conjunto de passos a serem seguidos, sabendo disso, é tido como desafio entender a característica do problema para encontrar soluções válidas com o uso dos componentes. Tendo o pic referente aos estudos de laboratório foi determinado como projeto a implementação de uma atividade onde por meio de assembly o componente consegue exercer sua atividades, onde ludicamente por meio das simulações é possível observar como os motores podem ser utilizados para completar atividades

Vale lembrar que os leds inseridos no circuito também ajudam na compreensão da execução das tarefas propostas para que a implementação seja iniciada.

3. Metodologia

Como projeto foi disposto o objetivo em implementar uma solução prática em Assembly utilizando microcontroladores da família pic, com hardware e software, então com auxílio de alguns programas como o MPLAB foram aplicados passos em assembly para ser inseridos no simulador buscando demonstrar como está o funcionamento das aplicações indicadas, são utilizados leds, motores, resistores e outros itens para completar essa tarefa, quando o motor recebe o pulso que vem da placa é possível observar os leds e o motor entrando em ação, com o fim do timer requerido na aplicação, temos o fim da ação e também é possível reverter essa ação com o outro motor.

4. Definição do Projeto

O projeto consiste em dois servos motores do tipo DC (corrente contínua), dois leds e dois botões. Cada botão aciona um motor e um led. Os motores irão girar em sentidos contrários quando acionados. Fazendo alusão a subida e descida da rampa. Outros equipamentos necessários para reproduzir fisicamente o projeto foram abstraídos, sendo usado apenas os componentes eletrônicos ligados diretamente ao pic16F628A.

Na figura 1 é possível ver nas linhas 4 até a linha 34 temos a configuração padrão para usar o pic16F628A juntamente com a definição das portas de entrada e saída utilizadas no projeto. Além disso é definido registradores adicionais para configurar o Timer0

```
4   ; --- Arquivos Inclusos no Projeto ---
5   #include    <p16F628a.inc>          ; inclui o arquivo do PIC 16F628A
6   ; --- FUSE Bits ---
7   ; - Cristal de 4MHz ; - Desabilitamos o Watch Dog Timer ; - Habilitamos o Power Up Timer ; - Browr
8   ; - Sem programacao em baixa tensao, sem protecao de codigo, sem protecao da memoria EEPROM
9   __config _XT_OSC & _WDT_OFF & _PWRTE_ON & _BOREN_OFF & _LVP_OFF & _CP_OFF & _CPD_OFF & _MCLRE_ON
10
11  ; --- Paginacao de Memoria ---
12  #define bank0 bcf STATUS, RP0   ; Cria um mnemonico para selecionar o banco 0 de memoria
13  #define bank1 bsf STATUS, RP0   ; Cria um mnemonico para selecionar o banco 1 de memoria
14
15  ; --- Mapeamento de Hardware ---
16  #define MOTOR1 PORTA, 1                 ; MOTOR1 ligado ao pino RA1
17  #define MOTOR2 PORTA, 3                 ; MOTOR2 ligado ao pino RA3
18  #define BOTAO1 PORTB, 0                 ; BOTAO1 ligado ao pino RB0
19  #define BOTAO2 PORTB, 1                 ; BOTAO2 ligado ao pino RB1
20  #define S3 PORTA, 5                     ; S3 ligado ao pino RA5/MCLRE (sera usado como entrada)
21
22  ; --- Registradores de Uso Geral ---
23  cblock  H'20'   ; Inicio da memoria disponivel para o usuario
24  W_TEMP          ; Registrador para armazenar o conteudo temporario de work
25  STATUS_TEMP     ; Registrador para armazenar o conteudo temporario de STATUS
26  counter1        ; Registrador auxiliar para contagem
27  endc            ; Final da memoria do usuario
28
29  ; --- Vetor de RESET ---
30  org     H'0000'         ; Origem no endereco 00h de memoria
31  goto    inicio  ; Desvia para a label inicio
32
33  ; --- Vetor de Interrupcao ---
34  org     H'0004'         ; As interrupcoes deste processador apontam para este endereco
```

Figura 1. Configurações iniciais

Na figura 2 temos as configurações do Timer0 desde o salvamento de contexto até o tratamento dos registradores, nas linhas 49 até 52 temos o tratamento do registrador counter 1 usado para aumentar o tempo em que o Timer0 continua ativo.

```
36  ; -- Salva Contexto --
37  movwf   W_TEMP          ; Copia o conteudo de Work para W_TEMP
38  swapf   STATUS,W        ; Move o conteudo de STATUS com os nibbles invertidos para Work
39  bank0                   ; Seleciona o banco 0 de memoria (padrao do RESET)
40  movwf   STATUS_TEMP     ; Copia o conteudo de STATUS com os nibbles invertidos para STATUS_TEMP
41  ; -- Final do Salvamento de Contexto --
42
43  ; Trata ISR...
44  btfss   INTCON, TOIF    ; Ocorreu um overflow no Timer0?
45  goto    exit_ISR        ; Nao, desvia para saida da interrupcao
46  bcf     INTCON,TOIF     ; Sim, limpa a flag
47  movlw   D'0'            ; Sim, move a literal 10d para work
48  movwf   TMR0            ; reinicializa TMR0 em 10d. Timer0 = 256 - 10 = 246
49  decfsz  counter1, F     ; Decrementa counter1. Chegou em zero?
50  goto    exit_ISR        ; Nao, desvia para saida da interrupcao
51  movlw   D'128'          ; move a literal 128d para work
52  movwf   counter1        ; reinicializa counter 1 em 18d
53  bcf     MOTOR1          ; desliga MOTOR1
54  bcf     MOTOR2          ; desliga MOTOR2
55  bcf     INTCON, TOIE    ; desabilita interrupcao do Timer0
56
57  ; -- Recupera Contexto (Saida da Interrupcao) --
58  exit_ISR:
59      swapf   STATUS_TEMP, W      ; Copia em Work o conteudo de STATUS_TEMP com os nibbles invertidos
60      movwf   STATUS              ; Recupera o conteudo de STATUS
61      swapf   W_TEMP, F           ; W_TEMP = W_TEMP com os nibbles invertidos
62      swapf   W_TEMP, W           ; Recupera o conteudo de Work
63      retfie                      ; Retorna da interrupcao
```

Figura 2. Configurações padrão do Timer0

Na figura 3 na linhas 65 até 84 temos o início do programa com limpeza e inicialização do Timer0 com o counter 1 e das saídas e entradas.

```
65  inicio:
66      bank1               ; Seleciona o banco 1 de memoria
67      movlw H'06'         ; w = 06h
68      movwf OPTION_REG    ; CONFIGURA OPTION_REG...
69  ; -> Pull Ups internos habilitados
70  ; -> Timer0 incrementa com ciclo de maquina
71  ; -> Prescaler 1:128 associado ao Timer0
72      movlw H'A0'     ; w=B'0000 0000
73      movwf TRISA     ; TRISA=H'A0' (todos bits sao saidas)
74      bank0           ; Seleciona o banco 0 de memoria
75      movlw H'07'     ; w = 7h
76      movwf CMCON     ; CMCON = 7h desabilita os comparadores internos
77      movlw H'80'     ; w = 80h
78      movwf INTCON    ; CONFIGURA INTCON...
79  ; -> Habilita Interrupcao Global
80  ; -> Interrupcao do Timer0 inicia desligada
81      movlw D'0'      ; move a literal 0d para work
82      movwf TMR0      ; inicializa TMR0 em 10d. Timer0 = 256 - 10 = 246
83      movlw D'128'            ; move a literal 128d para work
84      movwf counter1          ; inicializa counter1 em 18d
```

Figura 3. Início do programa

Na figura 04 temos a lógica do sistema com o loop infinito e as sub-rotinas de ligar os motores em conjunto com os leds. O timerOn1 liga o motor 1 e o led 1 da mesma forma para o timerOn2 que liga o motor 2 e o led 2.

```
86      loop:
87              btfss BOTAO1            ; BOTAO1 pressionado?
88              goto timerOn1           ; Sim, desvia para timerOn1
89              btfss BOTAO2            ; BOTAO2 pressionado?
90              goto timerOn2           ; Sim, desvia para timerOn2
91              goto loop               ; Nao, continua testando botao
92
93      timerOn1:
94              bsf     MOTOR1                  ; Liga MOTOR1
95              bsf     INTCON,TOIE             ; Habilita interrupcao do Timer0
96              goto loop                       ; Desvia de volta para loop
97
98      timerOn2:
99              bsf     MOTOR2                  ; Liga MOTOR2
100             bsf     INTCON,TOIE             ; Habilita interrupcao do Timer0
101             goto loop                       ; Desvia de volta para loop
102     end                             ; Final do Programa
103
```

Figura 4. loop do programa

5. Arquitetura

Na parte da arquitetura temos o código fonte escrito e compilado com a ajuda da IDE MPLAB X v5.35. Juntamente com a simulação montada e simulada no software Proteus.

O código foi escrito em assembly com uso do timer0 do pic16F628a para realizar a contagem do tempo de subida e descida da rampa eletrônica.

Na figura 5 podemos observar a montagem do circuito eletrônico com o PIC16F628A com os botões, motores e leds. Usou-se alguns resistores para manter a corrente adequadas para os motores e leds.

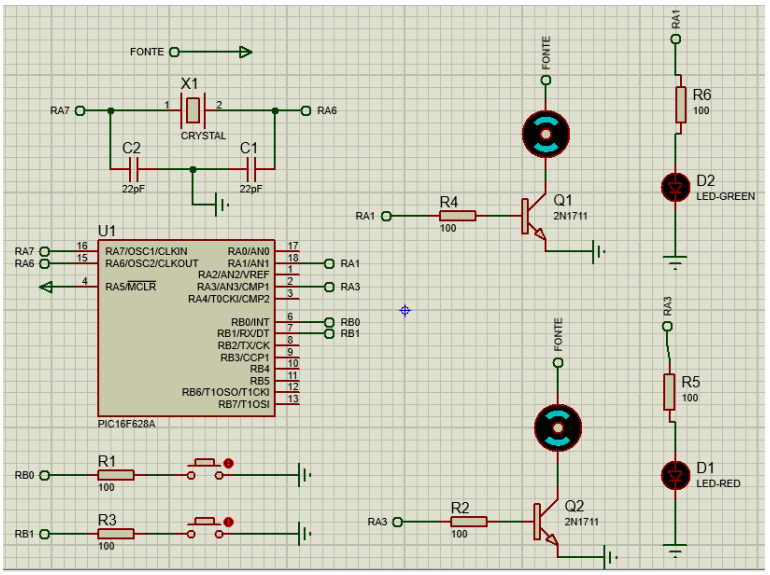

Figura 5. Arquitetura do projeto na simulação

6. Conclusão

Durante o desenvolvimento do projeto, surgiram alguns obstáculos, como a dificuldade em elaborar um projeto sem grande complexidade, tendo em vista que as aplicações escritas em assembly por si só já não são simples, devido ao baixo nível da própria linguagem utilizada, que limita o uso de operações a serem realizadas durante a manipulação das variáveis. Além disso, foi necessário uma pesquisa mais detalhada sobre o uso do TIMER0 do pic16f628a, para que fosse possível utilizá-lo no projeto.

Apesar das dificuldades citadas, foi possível desenvolver um projeto que atendesse aos requisitos do roteiro, como o uso do TIMER0, e que fosse ao mesmo tempo uma aplicação que poderia facilmente ser implementada em um caso real.

7. Referências

MANZANO, J. A. Programação assembly: padrão IBM - PC 8086/8088. 6ªed. Ed.Erica, 2012.

8. Apêndice

```
; --- Listagem do Processador Utilizado ---
list    p=16F628A           ; Utilizado PIC16F628A

; --- Arquivos Inclusos no Projeto ---
#include    <p16F628a.inc>      ; inclui o arquivo do PIC 16F628A
; --- FUSE Bits ---
; - Cristal de 4MHz ; - Desabilitamos o Watch Dog Timer ; - Habilitamos o Power Up Timer ; -
Brown Out desabilitado
; - Sem programacao em baixa tensao, sem protecao de codigo, sem protecao da memoria EEPROM
__config _XT_OSC & _WDT_OFF & _PWRTE_ON & _BOREN_OFF & _LVP_OFF & _CP_OFF & _CPD_OFF &
_MCLRE_ON

; --- Paginacao de Memoria ---
#define bank0 bcf STATUS, RP0   ; Cria um mnemonico para selecionar o banco 0 de memoria
#define bank1 bsf STATUS, RP0   ; Cria um mnemonico para selecionar o banco 1 de memoria

; --- Mapeamento de Hardware ---
#define MOTOR1 PORTA, 1         ; MOTOR1 ligado ao pino RA1
#define MOTOR2 PORTA, 3         ; MOTOR2 ligado ao pino RA3
#define BOTAO1 PORTB, 0         ; BOTAO1 ligado ao pino RB0
#define BOTAO2 PORTB, 1         ; BOTAO2 ligado ao pino RB1
#define S3 PORTA, 5         ; S3 ligado ao pino RA5/MCLRE (sera usado como entrada)

; --- Registradores de Uso Geral ---
```

```
cblock  H'20'    ; Inicio da memoria disponivel para o usuario
W_TEMP      ; Registrador para armazenar o conteudo temporario de work
STATUS_TEMP     ; Registrador para armazenar o conteudo temporario de STATUS
counter1    ; Registrador auxiliar para contagem
endc        ; Final da memoria do usuario

; --- Vetor de RESET ---
org H'0000'     ; Origem no endereco 00h de memoria
goto    inicio  ; Desvia para a label inicio

; --- Vetor de Interrupcao ---
org H'0004'     ; As interrupcoes deste processador apontam para este endereco

; -- Salva Contexto --
movwf   W_TEMP      ; Copia o conteudo de Work para W_TEMP
swapf   STATUS,W        ; Move o conteudo de STATUS com os nibbles invertidos para Work
bank0               ; Seleciona o banco 0 de memoria (padrao do RESET)
movwf   STATUS_TEMP     ; Copia o conteudo de STATUS com os nibbles invertidos para
STATUS_TEMP
; -- Final do Salvamento de Contexto --

; Trata ISR...
btfss   INTCON, T0IF    ; Ocorreu um overflow no Timer0?
goto    exit_ISR    ; Nao, desvia para saida da interrupcao
bcf INTCON,T0IF ; Sim, limpa a flag
movlw   D'0'        ; Sim, move a literal 10d para work
movwf   TMR0        ; reinicializa TMR0 em 10d. Timer0 = 256 - 10 = 246
decfsz  counter1, F ; Decrementa counter1. Chegou em zero?
goto    exit_ISR    ; Nao, desvia para saida da interrupcao
movlw   D'128'      ; move a literal 128d para work
movwf   counter1    ; reinicializa counter 1 em 18d
bcf MOTOR1      ; desliga MOTOR1
bcf MOTOR2      ; desliga MOTOR2
bcf INTCON, T0IE    ; desabilita interrupcao do Timer0

; -- Recupera Contexto (Saida da Interrupcao) --
exit_ISR:
   swapf   STATUS_TEMP, W  ; Copia em Work o conteudo de STATUS_TEMP com os nibbles invertidos
   movwf   STATUS      ; Recupera o conteudo de STATUS
   swapf   W_TEMP, F       ; W_TEMP = W_TEMP com os nibbles invertidos
   swapf   W_TEMP, W       ; Recupera o conteudo de Work
   retfie          ; Retorna da interrupcao

inicio:
   bank1       ; Seleciona o banco 1 de memoria
   movlw H'06'     ; w = 06h
   movwf OPTION_REG    ; CONFIGURA OPTION_REG...
; -> Pull Ups internos habilitados
; -> Timer0 incrementa com ciclo de maquina
; -> Prescaler 1:128 associado ao Timer0
   movlw H'A0'     ; w=B'0000 0000
   movwf TRISA     ; TRISA=H'A0' (todos bits sao saidas)
   bank0       ; Seleciona o banco 0 de memoria
   movlw H'07'     ; w = 7h
   movwf CMCON     ; CMCON = 7h desabilita os comparadores internos
   movlw H'80'     ; w = 80h
   movwf INTCON    ; CONFIGURA INTCON...
; -> Habilita Interrupcao Global
```

```
; -> Interrupcao do Timer0 inicia desligada
   movlw D'0'       ; move a literal 0d para work
   movwf TMR0       ; inicializa TMR0 em 10d. Timer0 = 256 - 10 = 246
   movlw D'128'         ; move a literal 128d para work
   movwf counter1       ; inicializa counter1 em 18d

loop:
   btfss BOTAO1         ; BOTAO1 pressionado?
   goto timerOn1        ; Sim, desvia para timerOn1
   btfss BOTAO2         ; BOTAO2 pressionado?
   goto timerOn2        ; Sim, desvia para timerOn2
   goto loop        ; Nao, continua testando botao

timerOn1:
   bsf MOTOR1           ; Liga MOTOR1
   bsf INTCON,T0IE      ; Habilita interrupcao do Timer0
   goto loop            ; Desvia de volta para loop

timerOn2:
   bsf MOTOR2           ; Liga MOTOR2
   bsf INTCON,T0IE      ; Habilita interrupcao do Timer0
   goto loop            ; Desvia de volta para loop
end             ; Final do Programa
```