



**UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
LABORAT. DE APLIC. COM MICROCOMPUTADORES**

**PROJETO 1:
Sistema de assistência ao motorista (ADAS) para detecção de
aproximação de veículos utilizando fotosensor**

Antonio Gabriel Sousa Borralho

**São Luís, MA
6 de julho de 2018**

Antonio Gabriel Sousa Borralho

PROJETO 1:

Título do projeto 1

Relatório referente ao primeiro projeto, para obtenção da primeira nota da disciplina Laboratório de Aplic. com Microcomputadores do curso de Engenharia Elétrica da UFMA no período de 2018.1.

Prof. André Borges Cavalcante.

São Luís, MA - Brasil

6 de julho de 2018

Resumo

Neste relatório buscou-se a implementação de um **Sistema de assistência ao motorista (ADAS) para detecção de aproximação de veículos utilizando fotosensor**. Ao realizar a leitura de um sensor de luminosidade foi possível acionar LEDs baseado em condições estabelecidas, onde esses LEDs simularam o que pode ser um painel de advertência incorporando ao painel principal do veículo, para o auxílio ao motorista. Além de que botões representavam o acelerador e freio do veículo para prever o que aconteceria caso o condutor respondesse com alguma ação. O funcionamento do projeto foi observado na prática

Palavras chave: HCS12; Microcomputadores; Conversores A/D.

Sumário

1	INTRODUÇÃO	5
2	DESCRIÇÃO DO PROBLEMA	6
2.1	Proposta de Metodologia	7
3	PROCEDIMENTOS EXPERIMENTAIS	8
3.1	Materiais	8
3.2	Métodos	9
4	RESULTADOS	10
4.1	Pseudocódigo	10
4.2	Fluxograma das Funções Construídas	12
4.3	Discussão do Funcionamento do Sistema	15
5	CONCLUSÃO	16
	REFERÊNCIAS	17
	APÊNDICES	18
	APÊNDICE A – MÓDULO EVENT	19
	APÊNDICE B – <i>MAIN</i> : MÓDULO PROJECT	22

1 Introdução

O *CodeWarrior^{TM1}* é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela *NXP Semiconductors* para edição, compilação e depuração de software para vários microcontroladores e microprocessadores ([NXP, 2018](#)). Integrada ao *CodeWarriorTM*, a tecnologia *Processor Expert* é um sistema de desenvolvimento para criar, configurar, otimizar, migrar e fornecer componentes de software para o ambiente do *CodeWarriorTM*.

A tecnologia *Processor Expert* torna muito mais fácil lidar com as complexidades de baixo nível de uma plataforma de hardware de uma maneira ideal. Podendo, assim, ser criados *drivers* periféricos personalizados, ideais para as necessidades, sem precisar saber tudo sobre o hardware.

O microprocessador HCS12 possui alta velocidade, processamento de 16 bits, que possui um modelo de programação idêntica à da indústria padrão da unidade de processamento central (CPU), M68HC11. O conjunto de instruções do HCS12 é um superconjunto da M68HC11, assim o conjunto de instruções e código fonte do M68HC11 é aceito pelo HCS12.

O Kit de Desenvolvimento HCS12 é uma ferramenta econômica para desenvolver código e avaliá-lo em microcontroladores da família HCS12 Dx, A e B. Este kit de fácil manipulação combina o *Multilink* do modo de depuração em segundo plano (*BDM*, *Background Debug Mode*) e um painel de avaliação (EVB, *Freescale Evaluation Board*). O EVB simplifica a avaliação do usuário de hardware e software protótipo, fornecendo o tempo essencial do microcontrolador, bem como uma área de protótipo para permitir a interface personalizada. O *BDM Multilink* faz a interface com o EVB através do conector BDM de 6 pinos para emulação em tempo real e programação rápida de flash ².

As duas técnica mais importantes no projeto da lógica de programas baseada em algoritmos denomina-se programação modular e Linguagem de Projeto de Programação (LPP)([MANZANO, 2010](#)). A LPP visa um forma preliminar de escrever um programa de computador sem preocupar-se com o rigor da linguagem de programação real a ser utilizada. Os Fluxogramas são formas gráficas utilizadas para representar uma sequência de passos a serem executados. O motivo da utilização de tais técnicas é agilizar a codificação da escrita e da programação, facilitar a depuração da leitura, permitira verificação de possíveis falhas apresentadas pelos programas, e facilitar as atualizações dos programas, bem como o processo de manutenção contribuindo para a fácil leitura de quem tem a necessidade de compreender o código implementado por outra pessoa.

¹ https://www.nxp.com/support/developer-resources/software-development-tools/codewarrior-development-tools:CW_HOME

² <https://www.nxp.com/docs/en/quick-reference-guide/S12QSG.pdf>

2 Descrição do Problema

Considere o cenário representado na Figura 1. Os carros A e B estão em movimento em uma rodovia. O motorista do carro A adormece ao volante fazendo com que o carro A acelere diminuindo a distância entre o carro A e o carro B. Deseja-se construir um sistema de assistência ao motorista, também conhecido como ADAS, para detectar que a distância entre o carro A e o carro B diminuiu e alertar o motorista do carro B através de um painel como ilustrado na Figura 2.

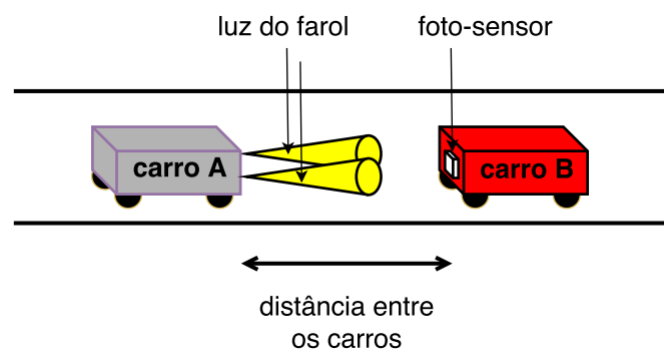


Figura 1 – Dois carros em movimento em uma rodovia. Deseja-se construir um ADAS para detectar que a distância entre o carro A e o carro B diminuiu e alertar o motorista do carro B.

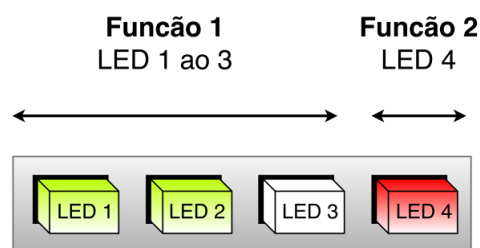


Figura 2 – *Dashboard* do carro B com *array* de LED para alerta ao motorista de duas funções

No cenário ilustrado na Figura 1, vamos assumir que os faróis do carro A estão sempre ligados. Além disso, vamos assumir que inicialmente os carros A e B tem mesma velocidade. E que a aceleração do carro do carro B é sempre zero. Neste caso, a distância entre o carro A e o carro B é apenas função da aceleração do carro A. Note que a medida que o carro A acelera, a distância entre os carros diminuirá. Uma vez que a distância entre os carros A e B diminui, espera-se que a intensidade de luz recebida no foto-sensor do carro B aumente. Dessa forma, talvez seja possível detectar a aproximação do carro A baseado na variação de intensidade de luz no foto-sensor. Para alertar o motorista do carro B sobre a possível aproximação, deve-se utilizar um *array* de LED que poderia ser instalado no

dashboard do carro. O sistema de alerta deve possuir duas funcionalidades. A primeira função do sistema de alerta é indicar a distância entre os carros A e B. Esta função pode ser realizada ligando de um a três LED de acordo com a intensidade de luz recebida no foto-sensor. Especificamente, quanto maior a intensidade de luz recebida, maior o número de LED ligados. A segunda função do sistema de alerta é indicar que a distância entre os dois carros está diminuindo. Especificamente, o sistema deve acender um quarto LED se e somente a intensidade de luz em um instante é menor que no instante de medição anterior. O sistema de alerta descrito acima é ilustrado na Figura 2.



Figura 3 – Simule os pedais acelerador e freio utilizando os botões do kit *NXP*

- *Upgrade 1 - Acelerador*

Após perceber uma aproximação, o motorista do carro B decide acelerar o veículo, aumentando a distância entre o carro A e o carro B. Neste caso, os LED do *dashboard* devem responder adequadamente a mudança de distância. Utilizando uma botão do kit *NXP*, simule um acelerador (Figura 3). Reescreva o controle dos LED para que o sistema de alerta ao motorista funcione adequadamente.

- *Upgrade 2 - Freio*

Implemente um o pedal de freio do carro B utilizando o segundo botão do Kit *NXP* (Figura 3). Note que o funcionamento do pedal de freio deve ser independente do acelerador. Mais uma vez os LED do *dashboard* devem responder adequadamente quando o carro sofrer uma redução de velocidade causada pelo pedal de freio. É importante que essa a resposta do *dashboard* seja gradual para que seja possível efetuar verificação visual.

2.1 Proposta de Metodologia

Para implementar as funcionalidades acima utilize um modelo matemático para calcular distância entre os carros como exemplo

$$distancia = aceleração - (luz + freio), \quad (2.1)$$

No qual distancia entre A e B aumenta quando o carro A acelera e diminui caso o carro A freie, ou o carro B se aproxima.

3 Procedimentos Experimentais

3.1 Materiais

Para montagem do referido experimento, utilizaram-se os seguintes materiais:

- Simulador dos faróis do carro A: lanterna.
- Simulador do acelerador e freio: botões.
- Foto-sensor do carro B: fotosensor do kit *NXP* com microcontrolador *HCS12C*.
- Temporização e amostragem: módulo *timer* do *HCS12C* em conjunto com o conversor AD.
- Processamento aritmético: realizado no *HCS12C* utilizando linguagem C.
- Interface de alerta ao motorista no *dashboard* do carro B: *array* de LED do kit com microcontrolador *HCS12C*.

A Figura 4 mostra o kit com microcontrolador *HCS12C*:

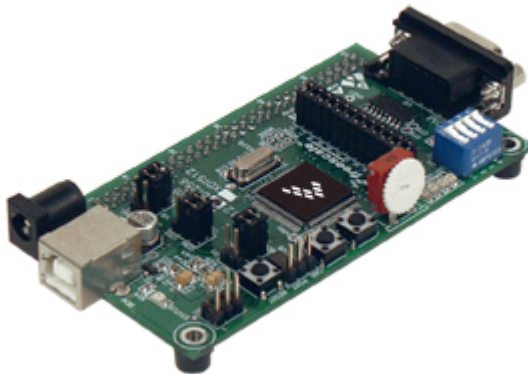


Figura 4 – *Kit de Desenvolvimento HCS12* da NXP

3.2 Métodos

Ao criar um novo projeto no *CodeWarriorTM* ativou-se a ferramenta *Processor Expert¹* pois ele gerou um ambiente com todos os itens preparados para utilização, facilitando a implementação do projeto. Inicialmente definiram-se as entradas e saídas do sistema com o auxílio do guia do usuário do HCS12², tendo em vista que os botões devem ser lidos por entradas digitais e os LEDs devem ser acionados por saídas digitais. E também, o fotosensor deve ser lido através de um conversor A/D. Após isso criou-se uma função de Conversor A/D para ler o fotosensor além de funções para gerar interrupções, temporizadas ou não. Estas funções serviram para melhor construção do código.

Para o funcionamento do conversor A/D, fez-se necessário a habilitação e a inicialização do mesmo na função principal do código, da seguinte forma:

```
1 void main(void)
2 {
3     PE_low_level_init();
4     /** End of Processor Expert internal initialization.***/
5
6     AD1_EnableEvent(); //Habilita o conversor A/D no módulo Event
7     AD1_Start(); //Inicializa o conversor A/D
8
9     for(;;){}
10 }
```

¹ <https://www.nxp.com/docs/en/user-guide/CWPEXUG.pdf>

² <https://www.nxp.com/docs/en/user-guide/APS12DT256SLKUG.pdf>

4 Resultados

4.1 Pseudocódigo

A seguir temos o pseudocódigo do projeto:

```

1  PROGRAMA  EVENTO
2
3  DECLARA  vAC,vFR,anterior,setValue:INTEIRO NÃO SINALIZADO
4  setValue=1000
5  anterior=0
6
7  função  AD1_OnEnd
8      DECLARA  sensorValue,value,load,adj:INTEIRO NÃO SINALIZADO
9
10     LEIA  sensorValue
11     load=sensorValue
12     LEIA  acelerador
13     LEIA  freio
14
15     SE  acelerador==0
16         vAC=vAC+1
17     SENÃO SE  freio==0
18         ATIVA  led4
19         vFR=vFR+1
20
21     SE  load>=vFR
22         value=load+vAC-vFR
23
24     adj=0
25     ENQUANTO  adj+1<value
26         adj=adj+1
27
28     SE  value<0 E  value>=setValue
29         value=sensorValue
30         vAC=0
31         vFR=0
32
33     SE  value<=setValue
34         ATIVA  led1

```

```
35     SE value< 2*setValue/3
36         ATIVA led2
37     SENÃO
38         APAGA led2
39     SE value<setValue/3
40         ATIVA led3
41     SENÃO
42         APAGA led3
43
44     ATIVA Interrupção
45
46     SENÃO
47         APAGA led1
48         APAGA led2
49         APAGA led3
50 FIM    AD1_OnEnd
51
52 função Interrupção
53     DECLARA sensorValue,atual: INTEIRO NÃO SINALIZADO
54     LEIA sensorValue
55
56     atual=sensorValue
57
58     SE anterior<atual+5
59         APAGA led4
60     SENÃO
61         ATIVA led4
62
63     anterior=atual
64 FIM Interrupção
65 FIM EVENTO
```

4.2 Fluxograma das Funções Construídas

Utilizando o conceito de modularidade ([MANZANO, 2010](#)) pode-se representar através de fluxogramas o código implementado no projeto.

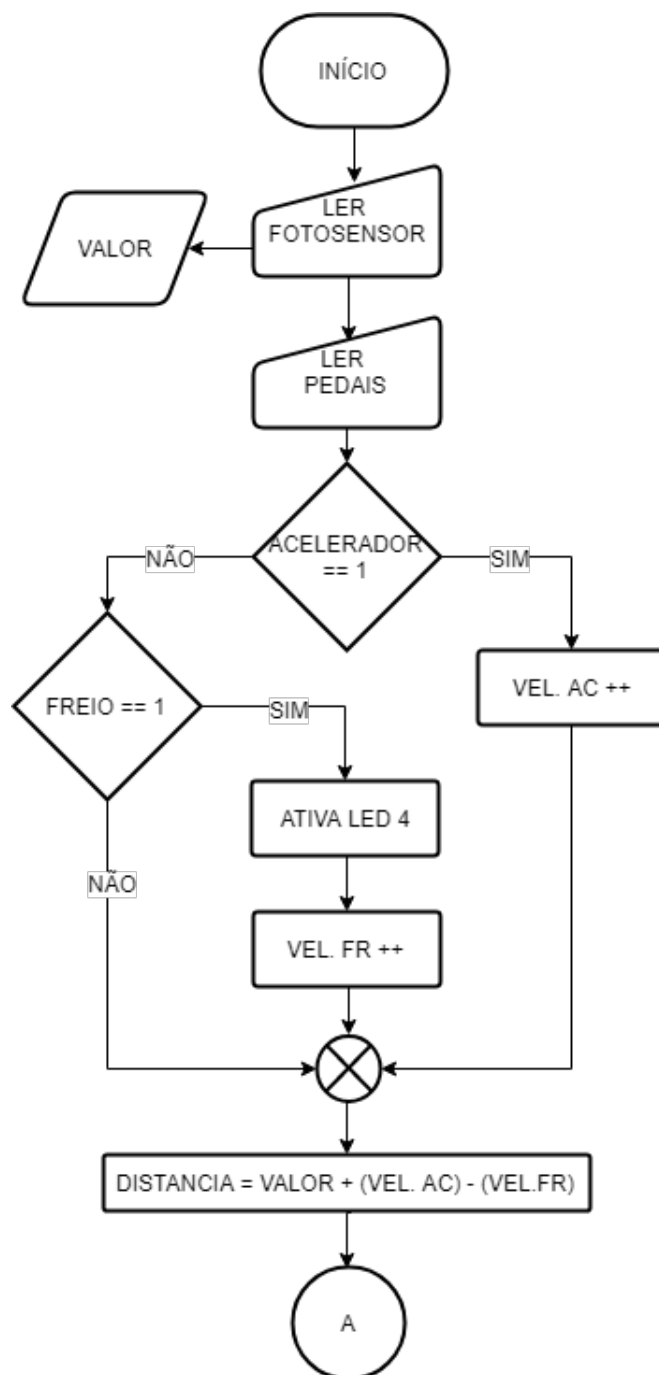


Figura 5 – Início do Código

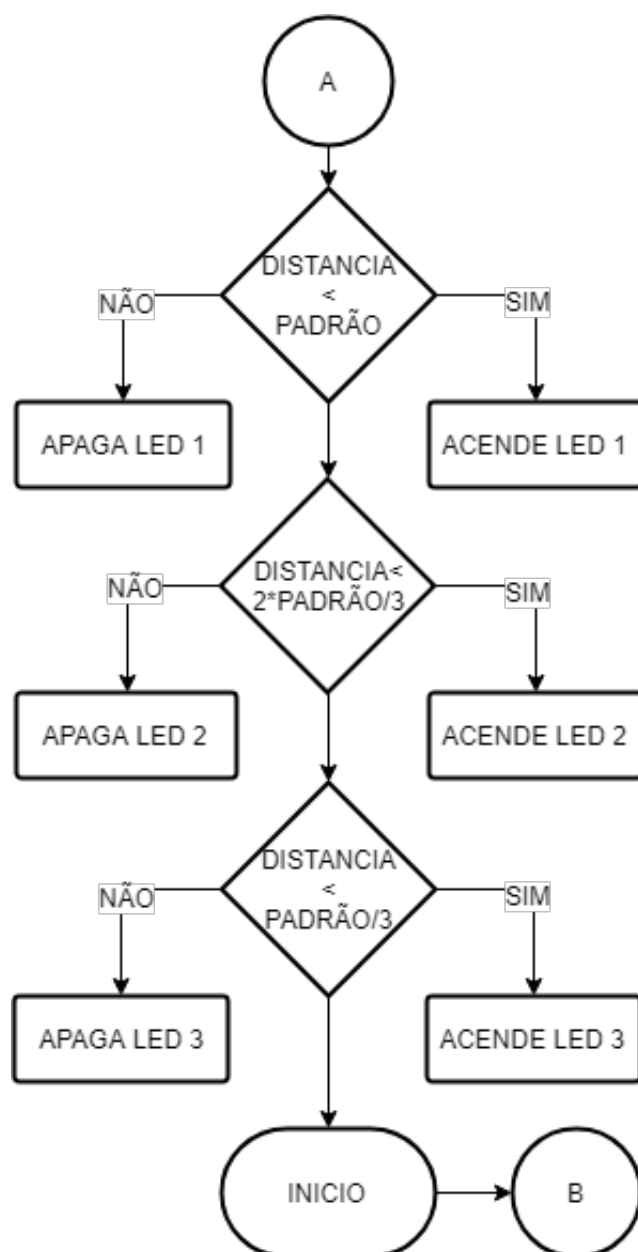


Figura 6 – Variação Gradual

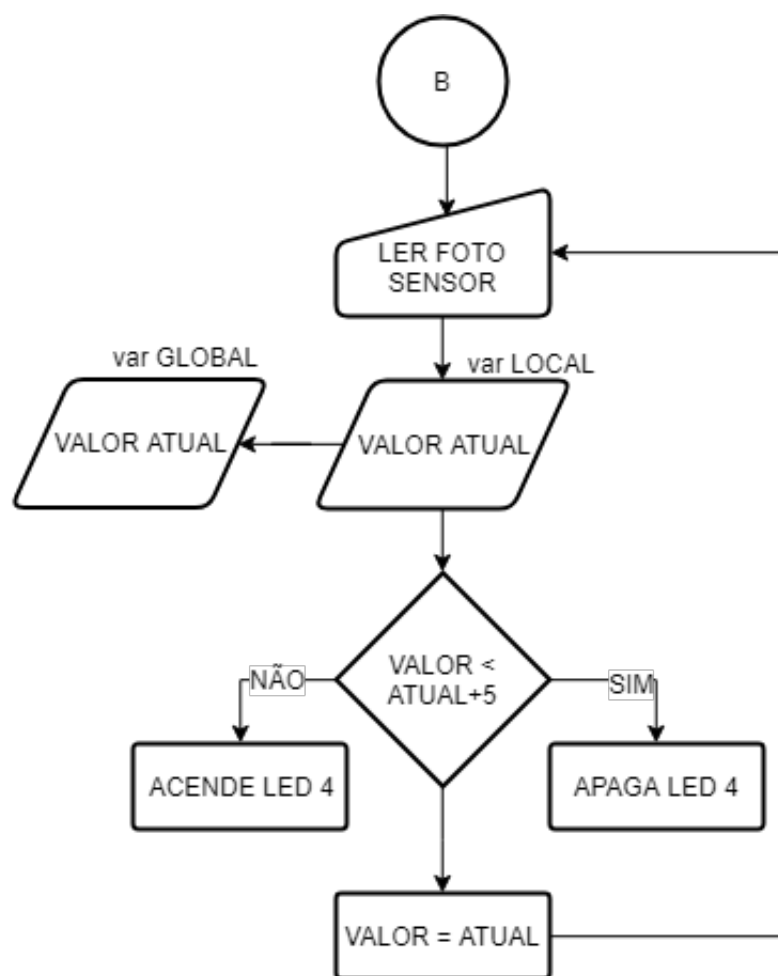


Figura 7 – Interrupção por variação

4.3 Discussão do Funcionamento do Sistema

Ao aproximar uma fonte de luz mais intensa que a luz ambiente próximo ao fotosensor pôde-se observar que os LEDs acendiam conforme a luz se aproximava indicando que o veículo A se aproximava atrás do veículo B, a referência de luminosidade foi definida como sendo a luminosidade do ambiente, em que deveria-se calibrar esse valor de acordo com cada ambiente. Para um projeto futuro e mais avançado pode-se implementar uma solução que se adapte e realize uma calibração automática para cada mudança de ambiente. Enquanto a fonte de luz se aproximava o quarto LED era acionado, porém o mesmo se mantinha apagado quando a fonte de luz se afastava ou permanecia com a mesma distância, ou seja, quando não havia variação de distância.

- *Upgrades* - Botões de Acelerador e Freio

Ao ser pressionado o botão correspondente ao acelerador, os LEDs se apagavam na mesma proporção, ou seja, quando mais tempo o acelerador era pressionado mais LEDs apagavam, caso os veículos estivessem muito próximos, indicando que os veículos se afastaram. Caso os veículos estivessem com uma distância considerável, nada acontecia com os LEDs. Neste momento um problema surgiu, o *underflow*, ou seja, quando a distância ultrapassava um valor mínimo negativamente, o valor da distância tendia para um valor máximo e todos os LEDs acendiam, onde deveriam permanecer todos apagados, indicando que os veículos estavam consideravelmente afastados. Para corrigir este problema foi criada uma condição que impedia que o *underflow* ocorresse. Este trecho do código é mostrado a seguir:

```
1      if ((value < 0) || (value >= setValue)) {  
2          value = sensorValue ;  
3          vAC = 0;  
4          vFR = 0;  
5      }
```

Onde **value** corresponde à última leitura realizada pelo conversor A/D do fotosensor, e **setValue** ao valor definido como luminosidade ambiente. $(value < 0)$ é a condição para evitar o *underflow* causado pelo acelerador e $(value \geq setValue)$ é a condição para evitar o *overflow* causado pelo freio (*Upgrade 2*). No caso do freio o mesmo problema ocorria porém um *overflow*, ou seja, quando a distância ultrapassava o valor de **setValue**, o valor da distância tendia para um valor negativo e todos os LEDs apagavam onde deveriam permanecer acesos indicando que a distância tendia a zero. Outra possível solução seria realizar o complemento de dois dos valores de *underflow* e *overflow*.

5 Conclusão

Um sistema de assistência ao motorista (ADAS) para detecção de aproximação de veículos pode ser muito útil em situações de adversidade, como por exemplo a falta de atenção acentuada do condutor veicular. Nesse contexto, a implementação de tal sistema se faz necessária. Como forma de aprendizagem e simulação foi desenvolvido um ADAS simples utilizando o Kit de Desenvolvimento com HCS12 que possui dentre seus componentes, um fotosensor. O problema proposto ilustra uma situação em que o motorista perde sua atenção causada pelo sono, ao perceber que um veículo se aproxima rapidamente, o sistema sinaliza ao motorista o risco corrido evitando um acidente. Tais funcionalidades foram testadas e observadas no Kit de Desenvolvimento com HCS12 executando o código implementado. Mesmo realizando inicialmente o esboço das ideias iniciais através de fluxogramas, ferramenta muito útil tanto para compreensão, aprendizagem e leitura de códigos difíceis quanto algoritmos mais simples, foi necessário ir melhorando essas ideias durante a implementação do código. Não foram previstos no pseudocódigo e nos fluxogramas iniciais os problemas de *underflow* e *overflow*, porém a construção dos fluxogramas auxiliaram para a identificação do erro. O funcionamento de todo sistema foi observado na prática.

Referências

MANZANO, J. A. N. G. *ALGORÍTIMOS: lógica para desenvolvimento de programação de computadores*. 24. ed. São Paulo: Érica, 2010. Citado 2 vezes nas páginas 5 e 12.

NXP. *Site da NXP*. 2018. Disponível em: <<https://www.nxp.com/>>. Acesso em: 30 jun 2018. Citado na página 5.

Todo o projeto pode ser acessado de forma livre no link contido no QR-Code abaixo:



Apêndices

APÊNDICE A – Módulo Event

A seguir temos o código implementado em *event*:

```

1  /** ##### PROJETO 1 #####
2  **  LAB. APLIC. COM MICROCOMPUTADORES
3  **  Filename   : Events.c
4  **  Project    : Projeto_01 - ADAS
5  **  Professor  : André Cavalcante
6  **  Processor  : MC9S12C128CFU16 (HCS12)
7  **  Component  : Events
8  **  Version    : Driver 01.04
9  **  Compiler   : CodeWarrior HC12 C Compiler
10 **  Date/Time  : 06/05/2018, 09:24
11 **  Aluno      : Antonio Gabriel Sousa Borralho
12 **  Ultima Mod: 20/05/2018
13 **
14 **  Observações:
15 **  Todos os LEDs são iniciados em nível 1 (Apagados)
16 **  Interrupções são feitas a cada 250ms
17 **
18 ** ##### */
19
20 /* MODULE Events */
21 #include "Cpu.h"
22 #include "Events.h"
23 #include "math.h"
24 #pragma CODE_SEG DEFAULT
25
26 // VARIÁVEIS GLOBAIS
27 int unsigned vAC,vFR,anterior=0;
28 int unsigned setValue=1000; //Calibrar de acordo com o ambiente
29
30 //===== INICIO-CONVERSOR_AD =====//
31 void AD1_OnEnd(void)
32 {
33     int unsigned sensorValue,value,load,adj;
34
35     AD1_GetChanValue(0x00,&sensorValue);
36     load=sensorValue;

```

```
37
38 //----- ACELERADOR/FREIO -----//
39 if(!ACELERADOR_GetVal()){
40 //É verdadeiro quando o ACELERADOR é pressionado
41     vAC++;
42 }else if(!FREIO_GetVal()){
43 //É verdadeiro quando o FREIO é pressionado
44     LED4_ClrVal();
45     vFR++;
46 }
47 //-- Modelo matemático para variação da distância (Botões)--//
48 if(load>=vFR){
49     value=load+vAC-vFR;
50 }
51 //----- Remove Buffer do Teclado -----//
52 adj=0;
53 while(adj+1<value){
54     adj++;
55 }
56 //---Evita comportamentos inesperados quando os botões são
57     ativados---//
58 if((value<0)|| (value>=setValue)){
59     value=sensorValue;
60     vAC=0;
61     vFR=0;
62 }
63
64 //----- LED-GRADUAL -----//
65 if(value<=setValue){
66     LED1_ClrVal(); // Acende o LED 1.
67     if(value<((2*setValue)/3))
68         LED2_ClrVal(); // Acende o LED 2.
69     else
70         LED2_SetVal(); // Apaga o LED 2.
71     if(value<setValue/3)
72         LED3_ClrVal(); // Acende o LED 3.
73     else
74         LED3_SetVal(); // Apaga o LED 3.
75
76
77
```

```

78 //----- LED-DE-APROXIMAÇÃO -----//
79 //Chama a Interrupção de variação de distância
80 dX_OnInterrupt;
81
82 //----- Apagar todos os LEDs -----//
83 }else{
84     LED1_SetVal();
85     LED2_SetVal();
86     LED3_SetVal();
87 }
88 }
89 //===== FIM-CONVERSOR_AD =====//
90
91 //== INCIO-INTERRUPÇÃO DE VARIAÇÃO DA DISTÂNCIA ==//
92 void dX_OnInterrupt(void)
93 {
94     int unsigned sensorValue, atual;
95     AD1_GetChanValue(0x00, &sensorValue);
96
97     atual=sensorValue;
98
99     if(anterior<atual+5){
100         LED4_SetVal();
101     }else{
102         LED4_ClrVal();
103     }
104     anterior=atual; //Atualiza o valor antigo
105 }
106 //== FIM-INTERRUPÇÃO DE VARIAÇÃO DA DISTÂNCIA ==//
107
108 /* END Events */
109
110 /** #####
111 **      This file was created by Processor Expert 3.02 [04.44]
112 **      for the Freescale HCS12 series of microcontrollers.
113 ** #####*/

```

APÊNDICE B – *Main*: Módulo Project

A seguir temos a função principal (*main*) implementado em *Project-1*:

```

1  /* MODULE Project_1 */
2
3  /* Including needed modules to compile this module/procedure */
4  #include "Cpu.h"
5  #include "Events.h"
6  #include "AD1.h"
7  #include "LED1.h"
8  #include "LED2.h"
9  #include "LED3.h"
10 #include "LED4.h"
11 #include "ACELERADOR.h"
12 #include "FREIO.h"
13 #include "dX.h"
14 /* Include shared modules, which are used for whole project */
15 #include "PE_Types.h"
16 #include "PE_Error.h"
17 #include "PE_Const.h"
18 #include "IO_Map.h"
19
20
21 void main(void)
22 {
23     PE_low_level_init();
24     /*** End of Processor Expert internal initialization.***/
25
26     AD1_EnableEvent(); //Habilita o conversor A/D no módulo Event
27     AD1_Start(); //Inicializa o conversor A/D
28
29     for(;;){}
30 }
31
32 /* END Project_1 */

```