PROJETO DE SISTEMAS COMPUTACIONAIS EMBARCADOS - STOCK CAR

por Lucas Paixão Cruz de Castro - RA: 791408

Departamento de Computação
10 de julho de 2025

Ministrante:
Prof. Emerson Carlos Pedrino

Conteúdo

1 Introdução

O presente projeto foi desenvolvido como parte da segunda avaliação da disciplina de ***Projetos de Sistemas Computacionais Embarcados***, ministrada no primeiro semestre de 2025. O objetivo principal foi implementar um jogo do tipo *stock car* utilizando um microcontrolador da família 8051 (especificamente o *AT89C52*) e um display gráfico *LCD* (*GLCD*), promovendo a aplicação prática de conceitos fundamentais de sistemas embarcados, controle de hardware e programação em linguagem *C*.

O jogo consiste em controlar um carro, representado por um *sprite* gráfico, através de dois botões físicos conectados ao microcontrolador. O jogador deve desviar dos limites laterais da pista e de obstáculos que surgem aleatoriamente ao longo do caminho. O carro é renderizado na última linha do *GLCD*, enquanto a pista e os obstáculos são atualizados dinamicamente a cada quadro, criando o efeito de rolagem vertical contínua.

Do ponto de vista do hardware, o sistema é composto por:

- **Microcontrolador *AT89C52* (derivado da arquitetura *Intel 8051*)**;
- **Display gráfico *LCD* (*GLCD*) de** $128x64$ ***pixels*, dividido em duas páginas (*CS1* e *CS2*)**;
- **Dois botões físicos para controle do carro (movimento para esquerda e direita)**;
- **Alimentação, resistores de *pull-up* e conexões de controle apropriadas**.

A implementação do software foi realizada integralmente em linguagem *C*, utilizando recursos de manipulação direta de registradores e portas de *I/O* do microcontrolador. As funções principais do programa incluem inicialização e controle do *GLCD*, movimentação do carro, geração de obstáculos, verificação de colisões e reinício do jogo em caso de falha. Além disso, uma mecânica de dificuldade progressiva foi implementada, fazendo com que o jogo aumente de velocidade a cada 10 segundos de sobrevivência contínua, reduzindo o tempo de atraso entre os quadros.

O código completo apresenta uma arquitetura modular, com clara separação entre lógica de jogo, controle gráfico e funções de hardware, promovendo legibilidade e facilitando a manutenção do sistema.

2 Desenvolvimento Prático

O desenvolvimento do projeto consistiu em duas etapas principais: a construção do circuito eletrônico no ambiente Proteus e a implementação do código em linguagem *C* para o microcontrolador *AT89C52*.

2.1 Circuito no Proteus

A simulação do sistema foi realizada no software *Proteus*, conforme mostra a Figura 1. O microcontrolador *AT89C52* é o componente central do circuito, responsável por controlar a lógica do jogo e a comunicação com o display gráfico *LCD* (*GLCD*) do tipo AMPIRE $128x64$.

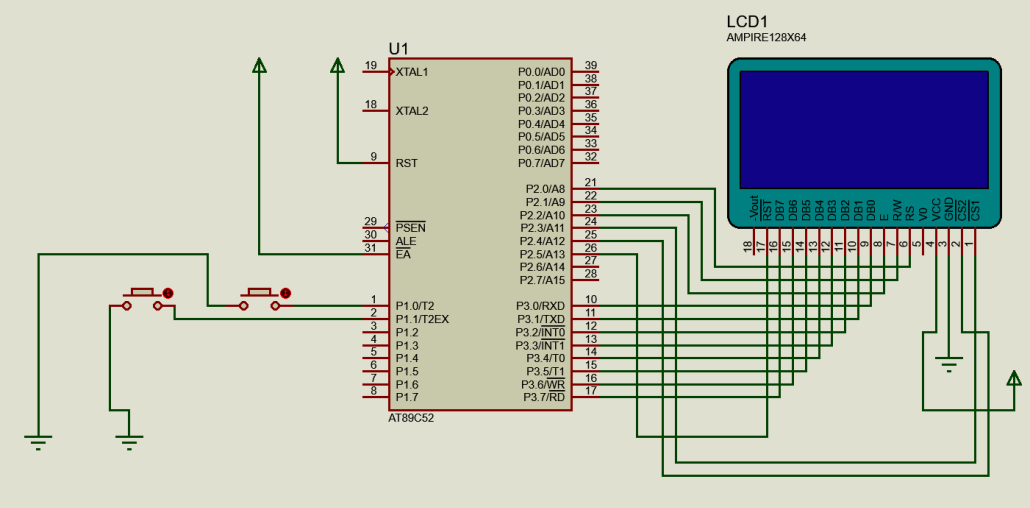

Figura 1: Circuito simulado no Proteus com o microcontrolador *AT89C52* e *GLCD* $128x64$.

O *GLCD* é conectado às portas P2 e P3 do microcontrolador, utilizando os sinais de controle `RS`, `RW`, `EN`, `CS1`, `CS2` e `RST`, além do barramento de dados de 8 bits. Dois botões físicos estão conectados às entradas `P1.0` e `P1.1`, permitindo ao jogador mover o carro para a direita e para a esquerda. O circuito conta ainda com a fonte de alimentação apropriada e os pinos de oscilador conectados ao cristal de clock.

2.2 Implementação do Software

A lógica do jogo foi implementada inteiramente em linguagem *C*, utilizando recursos do compilador para microcontroladores *8051*. O código-fonte completo está disponível na Listagem 1.

O programa inicializa o *GLCD* e exibe uma contagem regressiva antes de começar a corrida. A pista é renderizada dinamicamente linha por linha, simulando um efeito de rolagem vertical contínua. O jogador controla um carro que aparece fixo na parte inferior da tela, podendo se mover horizontalmente. Obstáculos são gerados aleatoriamente dentro dos limites da pista e descem em direção ao carro. Caso ocorra uma colisão com a borda da pista ou com um obstáculo, o jogo reinicia automaticamente após uma breve pausa.

Além disso, foi implementada uma lógica de aumento gradual de dificuldade: a cada 100 ciclos do jogo (aproximadamente 10 segundos), a velocidade aumenta por meio da redução do atraso entre quadros, tornando a jogabilidade progressivamente mais desafiadora.

```

// ---------------------- STOCK CAR ------------------------ //
// Jogo de corrida simples para microcontrolador AT89C52 com GLCD
// Desenvolvido por: Lucas Paixao Cruz de Castro
// Data: Julho de 2025
// Descricao: Jogo de corrida onde o jogador controla um carro por meio de dois botoes e deve desviar de obstaculos.
// ---------------------------------------------------------- //

// ------------------------------- //
// --- Declaracoes de variaveis --- //
// ------------------------------- //

// Importando Bibliotecas necessarias
#include <reg51.h>
#include <stdlib.h>

// Definicoes de hardware
#define GlcdDataBus  P3 // Porta de dados do GLCD
#define NUM_LINHAS 8 // Numero de linhas do GLCD (8 paginas)
#define LARGURA_PISTA 48 // Largura da pista (48 colunas)
#define LARGURA_CARRO 6 // Largura do carro (6 colunas)
#define LARGURA_OBSTACULO 7 // Largura do obstaculo (7 colunas)
#define MAX_OBSTACULOS 1 // Maximo de obstaculos na tela em um ciclo

// Definicoes de pinos no microcontrolador
sbit RS = P2^0;
sbit RW = P2^1;
sbit EN = P2^2;
sbit CS1 = P2^3;
sbit CS2 = P2^4;
sbit RST = P2^5;

// Botoes
sbit BTN_DIREITA = P1^0;
sbit BTN_ESQUERDA = P1^1;

// Variaveis globais
unsigned char posicoes_pista[NUM_LINHAS]; // Posicoes horizontais da pista
unsigned char posicao_carro = 61;  // Posicao inicial do carro (central)
unsigned char obstaculos_x[MAX_OBSTACULOS];  // Posicoes horizontais (0 a 127)
unsigned char obstaculos_y[MAX_OBSTACULOS];  // Posicoes verticais (0 a 7)
unsigned int delay_jogo = 80000; // Delay do jogo (inicialmente 80.000 ms)
unsigned int contador_frames = 0; // Contador de frames para aumentar a velocidade do jogo

// Sprites
const char code num_3[] = {0x21, 0x41, 0x45, 0x4B, 0x31, 0x00}; // "3"
const char code num_2[] = {0x42, 0x61, 0x51, 0x49, 0x46, 0x00}; // "2"
const char code num_1[] = {0x00, 0x21, 0x7F, 0x01, 0x00, 0x00}; // "1"
const char code letra_G[] = {0x3E, 0x41, 0x49, 0x49, 0x3A, 0x00}; // "G"
const char code letra_O[] = {0x3E, 0x41, 0x41, 0x41, 0x3E, 0x00}; // "O"
const char code letra_exclamacao[] = {0x00, 0x00, 0x5F, 0x00, 0x00, 0x00}; // "!"
const char code carro[] = {0xE7,0x42,0xFF,0xFF,0x42,0xE7}; // Carro principal 6 colunas
const char code carro_2[] = {0xE7,0x42,0xFF,0xE7,0xFF,0x42,0xE7}; // Carro secundario 7 colunas

// ----------------------------- //
// ----- Funcoes auxiliares ----- //
// ----------------------------- //

// Delay simples
void delay(int d) {
    while(d--);
}
```

```
63
64  // GLCD funcoes basicas
65  void Glcd_SelectPage0() { CS1 = 0; CS2 = 1; }
66  void Glcd_SelectPage1() { CS1 = 1; CS2 = 0; }
67
68  void Glcd_CmdWrite(unsigned char cmd) {
69      GlcdDataBus = cmd;
70      RS = 0; RW = 0; EN = 1; EN = 0;
71  }
72
73  void Glcd_DataWrite(unsigned char dat) {
74      GlcdDataBus = dat;
75      RS = 1; RW = 0; EN = 1; EN = 0;
76  }
77
78  void Glcd_Clear() {
79      unsigned char page, col;
80      for (page = 0; page < 8; page++) {
81          Glcd_SelectPage0(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40);
82          for (col = 0; col < 64; col++) Glcd_DataWrite(0x00);
83          Glcd_SelectPage1(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40);
84          for (col = 0; col < 64; col++) Glcd_DataWrite(0x00);
85      }
86  }
87
88  void Glcd_Init() {
89      Glcd_SelectPage0(); Glcd_CmdWrite(0x3F);
90      Glcd_SelectPage1(); Glcd_CmdWrite(0x3F);
91      Glcd_Clear();
92  }
93
94  void Glcd_DisplayChar4x6(const char *ptr)
95  {
96      int i;
97      for (i = 0; i < 4; i++)
98          Glcd_DataWrite(ptr[i]);
99      Glcd_DataWrite(0x00); // espaco entre caracteres
100 }
101
102 void Glcd_Print6x8(const char *letra, int col_start, int page) {
103     int i;
104     for (i = 0; i < 6; i++) {
105         if (col_start + i < 64) {
106             Glcd_SelectPage0();
107             Glcd_CmdWrite(0xB8 | page);
108             Glcd_CmdWrite(0x40 | (col_start + i));
109             Glcd_DataWrite(letra[i]);
110         } else {
111             Glcd_SelectPage1();
112             Glcd_CmdWrite(0xB8 | page);
113             Glcd_CmdWrite(0x40 | ((col_start + i) - 64));
114             Glcd_DataWrite(letra[i]);
115         }
116     }
117 }
118
119 void Glcd_Clear6x8(int col_start, int page) {
120     int i;
121     for (i = 0; i < 6; i++) {
122         if (col_start + i < 64) {
123             Glcd_SelectPage0();
124             Glcd_CmdWrite(0xB8 | page);
125             Glcd_CmdWrite(0x40 | (col_start + i));
126             Glcd_DataWrite(0x00);
127         } else {
128             Glcd_SelectPage1();
```

```
            Glcd_CmdWrite(0xB8 | page);
            Glcd_CmdWrite(0x40 | ((col_start + i) - 64));
            Glcd_DataWrite(0x00);
        }
    }
}

void Contador_Largada() {
    int col;
    int page;
    int total_largura;
    int col_go;

    col = (128 - 6) / 2; // centraliza caractere 6 colunas
    page = 3;

    // 3
    Glcd_Print6x8(num_3, col, page);
    delay(100000);
    Glcd_Clear6x8(col, page);

    // 2
    Glcd_Print6x8(num_2, col, page);
    delay(100000);
    Glcd_Clear6x8(col, page);

    // 1
    Glcd_Print6x8(num_1, col, page);
    delay(100000);
    Glcd_Clear6x8(col, page);

    // G O !
    total_largura = 3 * 6;
    col_go = (128 - total_largura) / 2;

    Glcd_Print6x8(letra_G, col_go, page);
    Glcd_Print6x8(letra_O, col_go + 6, page);
    Glcd_Print6x8(letra_exclamacao, col_go + 12, page);

    delay(100000);

    Glcd_Clear6x8(col_go, page);
    Glcd_Clear6x8(col_go + 6, page);
    Glcd_Clear6x8(col_go + 12, page);
}

// Desenha bordas verticais da pista
void desenhar_bordas(unsigned char page, unsigned char x_borda_esq) {
    unsigned char x_borda_dir = x_borda_esq + LARGURA_PISTA;

    if (x_borda_esq < 64) {
        Glcd_SelectPage0(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40 | x_borda_esq)
            ;
        Glcd_DataWrite(0xFF);
    } else {
        Glcd_SelectPage1(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40 | (x_borda_esq
             - 64));
        Glcd_DataWrite(0xFF);
    }

    if (x_borda_dir < 64) {
        Glcd_SelectPage0(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40 | x_borda_dir)
            ;
        Glcd_DataWrite(0xFF);
    } else {
        Glcd_SelectPage1(); Glcd_CmdWrite(0xB8 | page); Glcd_CmdWrite(0x40 | (x_borda_dir
```

```
            - 64));
        Glcd_DataWrite(0xFF);
    }
}

// Desenha o carro na linha 7
void desenhar_carro(unsigned char x) {
    unsigned char i;
    unsigned char chip, pos;

    for (i = 0; i < LARGURA_CARRO; i++) {
        unsigned char col = x + i;
        chip = (col < 64) ? 0 : 1;
        pos = (col < 64) ? col : (col - 64);

        if (chip == 0) Glcd_SelectPage0();
        else Glcd_SelectPage1();

        Glcd_CmdWrite(0xB8 | 7);    // Linha 7 (ultima)
        Glcd_CmdWrite(0x40 | pos);  // Posicao horizontal
        Glcd_DataWrite(carro[i]);
    }
}

// Inicializa pista centralizada
void inicializar_pista() {
    unsigned char i;
    unsigned char centro = (127 - LARGURA_PISTA) / 2;
    for (i = 0; i < NUM_LINHAS; i++) {
        posicoes_pista[i] = centro;
    }
}

// Faz scroll da pista (apenas atualiza posicoes)
void scroll_pista() {
    unsigned char i;
    unsigned char nova_pos;

    for (i = NUM_LINHAS - 1; i > 0; i--) {
        posicoes_pista[i] = posicoes_pista[i - 1];
    }

    nova_pos = posicoes_pista[0];
    if ((rand() % 2) && nova_pos < (127 - LARGURA_PISTA)) nova_pos++;
    else if (nova_pos > 1) nova_pos--;
    posicoes_pista[0] = nova_pos;
}

// Desenha a pista inteira
void desenhar_pista() {
    unsigned char i;
    for (i = 0; i < NUM_LINHAS; i++) {
        desenhar_bordas(i, posicoes_pista[i]);
    }
}

// Atualiza posicao do carro com base nos botoes
void atualizar_carro() {
    if (!BTN_DIREITA && posicao_carro < (127 - LARGURA_CARRO))
        posicao_carro++;
    if (!BTN_ESQUERDA && posicao_carro > 0)
        posicao_carro--;

    desenhar_carro(posicao_carro);
}
```

```
bit verificar_colisao() {
    unsigned char x_borda = posicoes_pista[7]; // posicao da pista na linha 7
    unsigned char x_fim_borda = x_borda + LARGURA_PISTA;

    if (posicao_carro < x_borda || (posicao_carro + LARGURA_CARRO - 1) > x_fim_borda) {
        return 1; // Colidiu
    }
    return 0; // Seguro
}

void inicializar_obstaculos() {
    unsigned char i;
    for (i = 0; i < MAX_OBSTACULOS; i++) {
        obstaculos_y[i] = 255; // 255 = inativo
    }
}

void gerar_obstaculo() {
    unsigned char i;
    for (i = 0; i < MAX_OBSTACULOS; i++) {
        if (obstaculos_y[i] == 255) {
            // Gera dentro da pista na linha 0
            unsigned char x_borda = posicoes_pista[0];
            obstaculos_x[i] = x_borda + 2 + (rand() % (LARGURA_PISTA - LARGURA_OBSTACULO
                - 4));
            obstaculos_y[i] = 0;
            break;
        }
    }
}

void atualizar_obstaculos() {
    unsigned char i;
    for (i = 0; i < MAX_OBSTACULOS; i++) {
        if (obstaculos_y[i] < NUM_LINHAS) {
            obstaculos_y[i]++;
            if (obstaculos_y[i] >= NUM_LINHAS) {
                obstaculos_y[i] = 255; // Apaga se saiu da tela
            }
        }
    }

    // Chance aleatoria de gerar novo obstaculo
    if (rand() % 5 == 0) {
        gerar_obstaculo();
    }
}

void desenhar_obstaculos() {
    unsigned char i, j;
    for (i = 0; i < MAX_OBSTACULOS; i++) {
        if (obstaculos_y[i] < NUM_LINHAS) {
            unsigned char y = obstaculos_y[i];
            unsigned char x = obstaculos_x[i];

            for (j = 0; j < LARGURA_OBSTACULO; j++) {
                unsigned char col = x + j;
                unsigned char chip = (col < 64) ? 0 : 1;
                unsigned char pos = (col < 64) ? col : (col - 64);

                if (chip == 0) Glcd_SelectPage0();
                else Glcd_SelectPage1();

                Glcd_CmdWrite(0xB8 | y);        // Pagina
                Glcd_CmdWrite(0x40 | pos);      // Posicao horizontal
                Glcd_DataWrite(carro_2[j]);     // Desenha
```

```
            }
        }
    }
}

bit verificar_colisao_obstaculos() {
    unsigned char i;
    for (i = 0; i < MAX_OBSTACULOS; i++) {
        if (obstaculos_y[i] == 7) {
            // Linha do carro
            if ((posicao_carro + LARGURA_CARRO - 1) >= obstaculos_x[i] &&
                posicao_carro <= (obstaculos_x[i] + LARGURA_OBSTACULO - 1)) {
                return 1; // Colidiu
            }
        }
    }
    return 0;
}

void game_over() {
    Glcd_Clear();
    delay(60000);
    posicao_carro = 61;
    inicializar_pista();
    inicializar_obstaculos();
    delay_jogo = 80000;         // Reinicia velocidade
    contador_frames = 0;        // Reinicia contador
}

// ------------------------------- //
// ------- Loop principal ------- //
// ------------------------------- //

void main() {
    Glcd_Init();
    P1 = 0xFF;
                Contador_Largada();
    inicializar_pista();
    inicializar_obstaculos();

    while (1) {
        Glcd_Clear();
        scroll_pista();
        atualizar_obstaculos();
        desenhar_pista();
        desenhar_obstaculos();
        atualizar_carro();

        if (verificar_colisao() || verificar_colisao_obstaculos()) {
            game_over();
        }

        delay(delay_jogo);

        contador_frames++;
        if (contador_frames >= 100 && delay_jogo >= 20000) {
           delay_jogo -= 5000; // Aumenta a velocidade
           contador_frames = 0; // Reinicia o contador
        };
    }
}
```

Listing 1: Código em *C* do jogo *Stock Car*.

3 Resultados

A implementação do jogo *Stock Car* foi concluída com sucesso, atendendo aos objetivos propostos da atividade. A Figura 2 mostra uma captura da tela do jogo durante a execução, exibindo o carro controlado pelo jogador, um obstáculo e as bordas da pista.

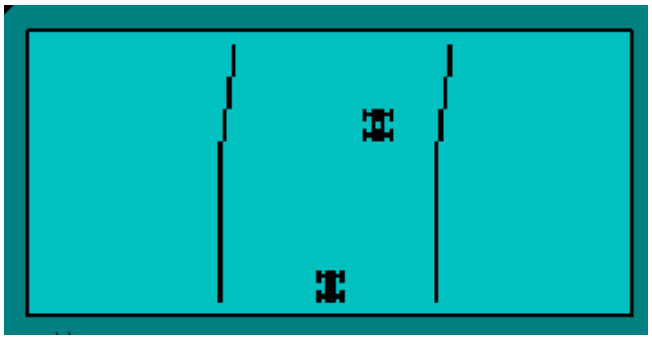

Figura 2: Execução do jogo no GLCD, com obstáculo e bordas visíveis.

Durante os testes, o jogo demonstrou comportamento estável, com o sistema de colisão funcionando corretamente tanto com os obstáculos quanto com as bordas da pista. O aumento progressivo da velocidade a cada 10 segundos de sobrevivência também foi observado com sucesso, tornando a jogabilidade mais desafiadora com o passar do tempo.

Entretanto, algumas possíveis melhorias foram identificadas:

- **Responsividade dos botões:** O sistema de controle por botões apresentou um pequeno atraso devido à forma como o delay é implementado no loop principal. A substituição do delay por timers ou interrupções poderia melhorar significativamente a resposta do movimento do carro.
- **Métricas de desempenho:** A implementação de uma contagem de tempo de sobrevivência ou pontuação poderia tornar o jogo mais envolvente e competitivo.

De modo geral, o projeto cumpriu sua proposta de desenvolver um jogo funcional em um sistema embarcado limitado, evidenciando o domínio das técnicas de controle de hardware, manipulação gráfica e lógica de jogo em linguagem C.

Referências

[1] STOCK CAR – JTYONE ZX81 Archive. Disponível em: `https://www.zx81stuff.org.uk/zx81/jtyone.html?track=StockCar.tzx.zip@0&title=Stock%20Car&scale=2&speed=4`. Acesso em: 10 jul. 2025.