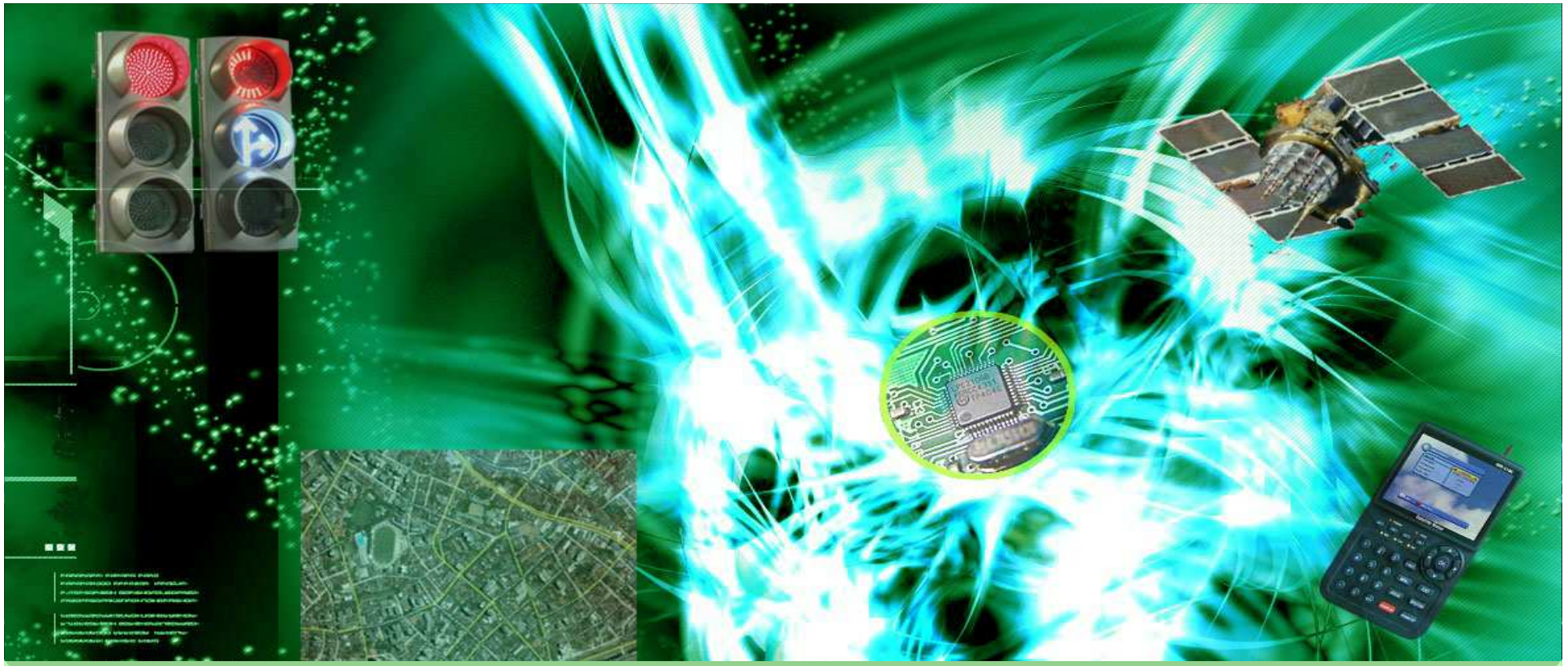


PROJETO DE SOFTWARE EMBUTIDO

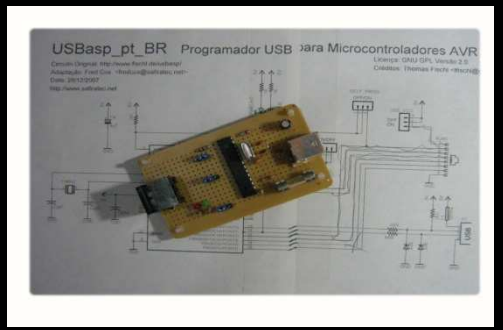
Professor: Dr. Fernando Castor

Alunos: Fred Cox
Joas Souza

1. INTRODUÇÃO

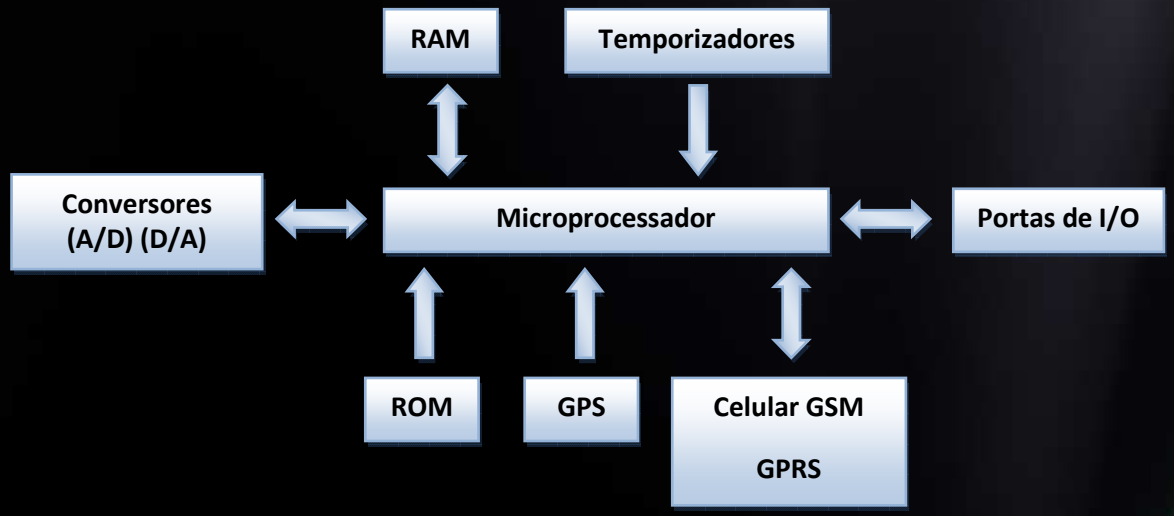
Um **sistema embarcado**, ou **sistema embutido**, é um sistema microprocessado no qual o computador possui recursos de armazenamento, consumo e desempenho específico para a aplicação.

Diferente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas pré-definidas, geralmente com requisitos específicos. Já que o sistema é dedicado à tarefas específicas, através de engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo do produto.



2. MICROCONTROLADORES X MICROPROCESSADORES

Projeto de um ***Rastreador Veicular*** com **Microprocessador**



- **Complexidade do circuito**
- **Microprocessador é um dispositivo de propósito geral**
- **Baixa imunidade à ruídos**

2. MICROCONTROLADORES X MICROPROCESSADORES

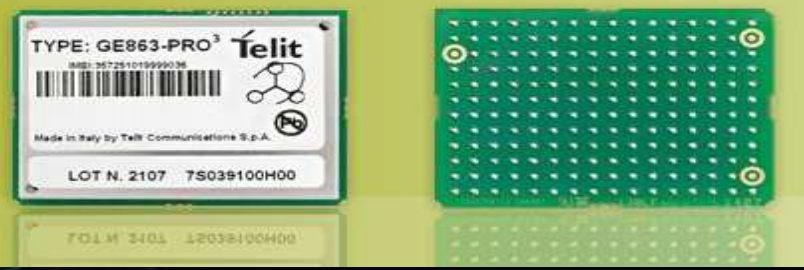
Projeto de um ***Rastreador Veicular*** com **Microcontrolador**

- **Circuito mais simples**
- **Microcontrolador é dedicado e de propósito específico**
- **Alguns microcontroladores são produzidos para o segmento automobilistico.**

http://www.telit.com/

GE863-PRO³

Embedded



3. MICROCONTROLADORES

Microcontroladores mais utilizados

- **8051**
- **Atmel AVR**
- **Texas MSP430**
- **Microchip PIC**
- **ARM7/ARM9**

4. SEGMENTAÇÃO DE SOFTWARE EMBARCADO

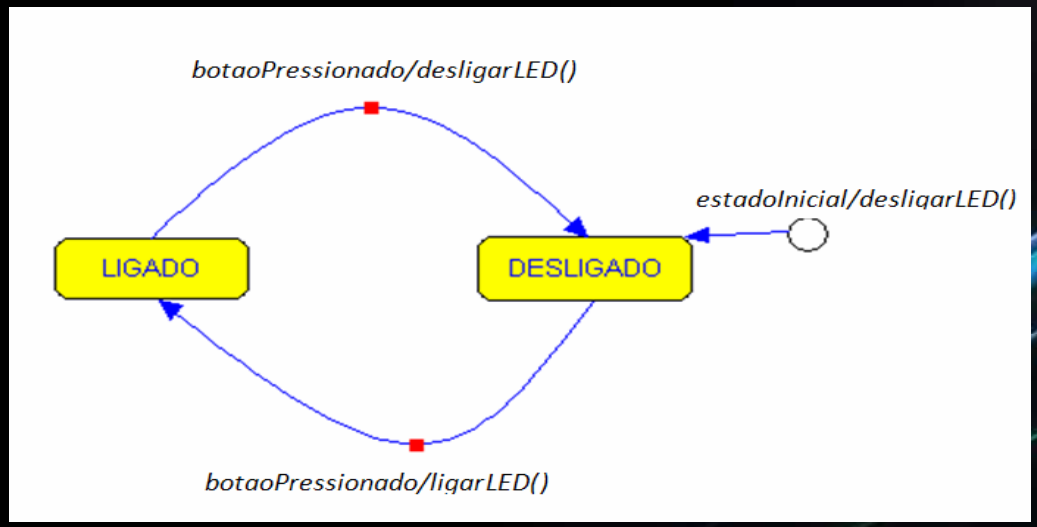
Padrão arquitetural de desenvolvimento software embarcado.

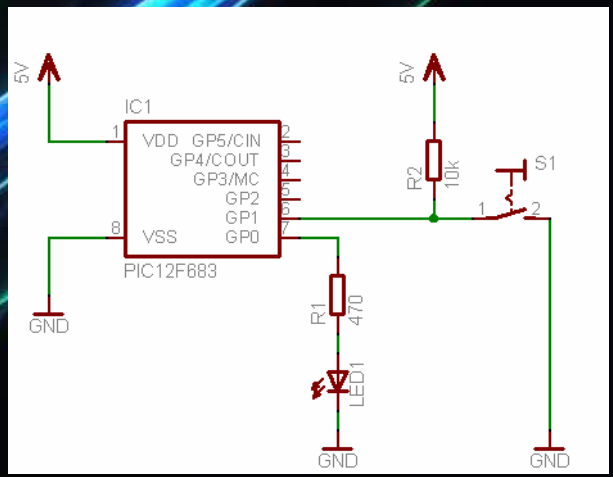
Item	Descrição
Drivers	São funções de baixo nível que realizam a interface do sistema embarcado com o hardware.
Maquinas de Estados	Realizam transições dos estados através de eventos que produzem uma ação no sistema.
Monitor de Eventos	Monitoram as entradas/saídas do sistema disparando eventos.
Interrupções	Subrotinas que interrompem a execução do sistema.

5. DESENVOLVIMENTO

5.1 Máquina de Estados

Uma máquina de estados finitos ou Autômato Finito é uma modelagem de um comportamento de um sistema, composto por estados, transições e ações. Um estado armazena informações sobre o passado, isto é, ele reflete as mudanças desde a entrada num estado, no início do sistema, até o momento presente.
Uma transição indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra.
Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento





5. DESENVOLVIMENTO

Implementação da máquina de Estados em C

```
10/**********************/
11/* Define os Estados */
12typedef enum {LIGADO,DESLIGADO} ESTADOS;
13/**********************/
14/* Define os eventos */
15typedef enum {botaoPressionado} EVENTOS;
16/**********************/
17#define BOTAO   REO
18#define LED LATE1
19
20/*****************************/
21/* Drivers                   */
22/*****************************/
23void ligarLED(void) {
24    LED = 1;          //acende o LED
25}
26void desligarLED(void) {
27    LED = 0;          //apaga o LED
28}
```

```
30/*****************************/
31/* Maquina de Estados        */
32/*****************************/
33void transicao(EVENTOS e) {
34    static ESTADOS estado   = DESLIGADO;
35    switch (estado) {
36        case LIGADO:
37            estado      = DESLIGADO;
38            desligarLED();
39            break;
40        case DESLIGADO:
41            estado      = LIGADO;
42            ligarLED();
43            break;
44    }
45}
```

```
49void main(void)
50{
51    /* Estado inicial [Desliga o LED] */
52    transicao(botaoPressionado);
53    while (1){
54        /* Monitor de Eventos */
55        if(!BOTAO) {
56            transicao(botaoPressionado);
57        }
58    }
59}
```

5. DESENVOLVIMENTO

Há diversas ferramentas de modelagem de sistemas utilizando máquinas de estados finitos, entre elas podemos citar:

- **IAR Visual State** – Este aplicativo proporciona o desenho visual da máquina de estados e gera o código fonte em C ou C++ automaticamente para o projeto. Pode ser obtido através do seguinte site: *http://www.iar.com*
- **Quantum Leaps** – Disponibiliza um framework de implementação de máquinas de estados voltado para eventos. Site: *http://www.quantum-leaps.com/*
- **ZMECH - PIC Edition** – Uma suite case para microcontroladores PIC.

6. TOLERÂNCIA A FALHAS

Tolerância a falhas é um mecanismo para lidar com os problemas que potencialmente possam **afetar os sistemas**.

Diferente da **prevenção de falhas**, tolerar as falhas do sistema, implica em reconhecer que **as falhas são inevitáveis**; tendo origem em erros de projeto ou de implementação, **desgaste do material** ou **colapsos na fonte de energia**; e oferecer alternativas que permitam ao sistema **manter o funcionamento desejado mesmo na ocorrência de falhas**.

Ainda que todo cuidado tenha sido empregado, utilizando técnicas formais de especificação e refinamento dos projetos e verificações de que a implementação dos algoritmos é correta, o software depende do hardware para executar suas funções, estando este sujeito ao desgaste físico do material, que é inevitável.

6. TOLERÂNCIA A FALHAS

Software Embarcado (Ambientes inóspitos)

Muitos sistemas embarcados operam em **ambientes inóspitos** ou mesmo inacessíveis, demandando a necessidade de **proteção contra choques**, **vibrações**, **flutuações na fonte de energia**, calor excessivo, fogo, água, corrosão, etc.

Em tais ambientes a possibilidade de falha causada por um sinistro é elevada, o que demanda um bom **mecanismo de tolerância a falhas**.

7. TÉCNICAS DE TOLERÂNCIA A FALHAS

BLOCOS DE RECUPERAÇÃO (RECOVERY BLOCKS)

Técnica que consiste na **construção de blocos tolerantes a falhas**.

A estrutura, basicamente consiste de três elementos: uma **rotina primária**, que executa funções críticas; um **teste de aceitação**, que testa a saída da rotina primária após cada execução; e uma **rotina alternativa**, que realiza a mesma função da rotina primária (só que com menor capacidade ou mais devagar) e é disparada pelo teste de aceitação ao detectar uma falha.

6. TÉCNICAS DE TOLERÂNCIA A FALHAS

PROGRAMAÇÃO N-VERSÕES

Consiste na geração de **N versões** de uma aplicação. Com **N** maior ou igual a **2**, sendo estas aplicações originárias da mesma especificação, mas programadas com **ferramentas, linguagens, métodos e testes diferentes**. Essa técnica utiliza um sistema de votação das **N-versões**, para geração do resultado final, e portanto, trabalha com um **mascaramento de falhas**.

As diferentes versões são executadas paralelamente ou seqüencialmente e os resultados das versões são transmitidos ao votador, responsável por determinar um resultado consensual para a computação, de acordo com algum critério. Por exemplo com maior freqüência. Isto é, o resultado será aquele dado pela maioria dos componentes. Caso o votador não seja capaz de determinar o resultado, então uma execução é sinalizada.

6. TÉCNICAS DE TOLERÂNCIA A FALHAS

```
56 void serialStateMachine(void) {
57     static unsigned char subState   = 0;
58     unsigned char index = 0;
59     switch (subState) {
60         case 1:
61             /* Espera por CR e LF */
62             if((serial.buffer[serial.index - 2]==13) && (serial.buffer[serial.index - 1]==10)) {
63                 subState        = 2;
64             }
65             /* Se o delimitador de final de protocolo nao chegou em 3 ms - Abandone! */
66             if(timeOutSerial>3) {
67                 subState        = 3;
68             }
69             break;
70         case 2:
71             analisaComando();
72             subState            = 3;
73             break;
74         case 3:
75             resetSerialBuffer();
76             subState            = 0;
77             break;
78         default:
79             /* Verifica se chegou algum caracter */
80             if(serial.index>0) {
81                 /* Espera chegada de delimitador de inicio do protocolo */
82                 if(serial.buffer[serial.index - 1]== '$') {
83                     subState        = 1;
84                     timeOutSerial   = 0;
85                 } else {
86                     subState        = 3;
87                 }
88             }
89             break;
90     }
91 }
```

Referências:

- www.safiratec.net
  Blog de eletrônica e sistemas embarcados (Fred)

- Software Fault Tolerance, Chris Inacio
  Carnegie Mellon University
  18-849b Dependable Embedded Systems
  http://www.ece.cmu.edu/~koopman/des_s99/sw_fault_tolerance/

FIM