C para embarcados

Slide 5

Rafael Corsi - corsiferrao@gmail.com

March 7, 2016

Instituto Mauá de Tecnologia
EEN251 - Microcontroladores e Sistemas Embarcado

Conteúdo

1. bits
2. Constantes
3. Parâmetros do compilador (GCC)
4. Entradas e Saídas

bits

Mudar ou Configurar um bit

31.6.10 PIO Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Set Output Data**

0: No effect.

1: Sets the data to be driven on the I/O line.

Mudar ou Configurar um bit

Zerando bits :

```
PORTH &= 0x F5; // Configurando bits de [1 3] para zero
PORTH &= ~0x0A; // Mesmo que anterior porem mais explicito
```

Setando bits :

```
PORTH |= 0x0A; // Configurando bits de [1 3] para um usando OR
```

Constantes

Constantes

Esse tipo de declaração de função diz que a função não irá modificar a variável passada para a função.

```
void print_string( char const * the_string );
```

Constantes

Esse tipo de declaração de função diz que a função não irá modificar a variável passada para a função.

```
void print_string( char const * the_string );
```

Em microcontroladores é comum possuímos menos memória RAM do que ROM :

```
char * months[] = {
        "January", "February", "March",
        "April", "May", "June",
        "July", "August", "September",
        "October", "November", "December",
    };
```

Por isso é importante utilizar constantes :

```
char const * const months[] = { ... };
```

Parâmetros do compilador (GCC)

Static

O uso do pragma static, permite ao compilador identificar quando a variável sofre ou não modificação, dando liberdade para ele escolher se irá alocar na RAM ou ROM :

```
static char * months[] = { ... };
```

Volatile

volatile diz ao compilador para não otimizar determinada variável e sempre carregar-la da RAM (não permite o cache).

```
volatile int *dst = somevalue;
```

É bastante utilizado quando uma variável é alterada pelo hardware ou interrupção.

https://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Volatiles.html#Volatiles http://www.avrfreaks.net/forum/extern-static-and-volatile-variables

Volatile

- **leitura** : compiladores supõem que uma vez que o programa lê uma variável em um registro, ele não precisa de a ler essa variável cada vez que o código-fonte menciona-la, usando o valor em cache no registo. Isso funciona muito bem com os valores normais em ROM e RAM, mas falha com periféricos de entrada. O mundo externo (temporizadores internos e contadores) poem mudar frequentemente a variável, fazendo com que o valor em cache obsoleto e irrelevante.

Volatile

- **escrita** : "sem "volátil", compiladores C assumem que não importa a ordem em que a escrita acontece na variável, e apenas a última gravação para uma determinada variável realmente importa. Isso funciona muito bem com os valores normais na RAM, mas falha com periféricos de saída típicas.S

#ifdef

Sempre que possível devemos deixar o código modular, o uso de *#ifdef* é muito eficiente

```
if(test_var(x) >= MAX_VALUE){
   return(-1);

#ifdef DEBUG_EN
   printf("[DEBUG] Var > Permitido : %d \n", x);
#endif
}
```

Nesse caso, se definirmos #DEBUG_EN ativamos uma parte do código que lida com o debug.

Entradas e Saídas

Funções de entrada e saída

- printf : imprime na tela uma string formatada
- fopen : abre um arquivo
- fputc : escreve um arquivo
- ...

Mas como isso funciona em um microcontrolador ? Já que não temos teclado nem monitor.

Solução

Devemos mapear as entradas e saídas para componentes que possuímos no microcontrolador.

O printf pode ser usada via :

- comunicação serial (UART)
- via debug JTAG.

A leitura de arquivo pode ser :

- ler da RAM;
- SDCARD

Solução UART