C++ para Microcontroladores

Luiz de Barros Oliveira Neto

Lboneto@gmail.com

08/11/2008

Sistema Embarcado

- Sistema desenvolvido para desempenhar uma ou algumas funções específicas e bem definidas.

Microcontrolador

- Unidade de processamento simplificada. Normalmente integrada com Núcleo, memória e I/O

Arquitetura de Hardware Sistemas Embarcados

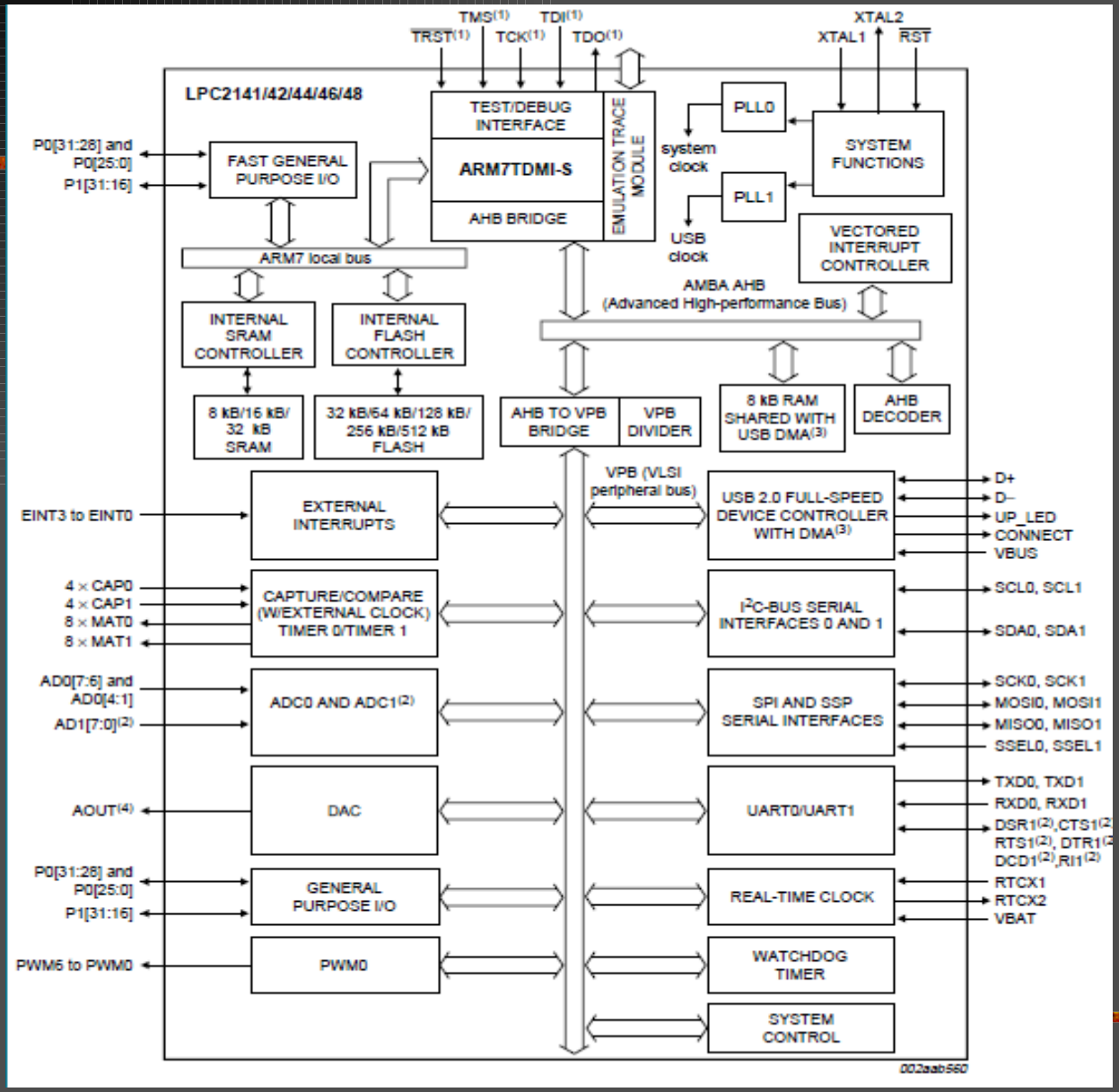
Componentes Básicos

- Núcleo
- Memória RAM (interna/externa)
- Memória Flash
- Controlador Interrupções
- Temporizador/Contador
- SPI
- I2C
- UARTs
- GPIO

Componentes Adicionais

- Ethernet
- USB (Host/Device/OTG)
- CAN
- LVDS (monitor LCD)
- Áudio
- MMU (Gerenciamento de Memória)
- AHB
- SD/MMC
- Compact Flash

Arquitetura Hardware LPC2148



Algumas Famílias de Microcontroladores

8 Bit

- AVR 8-bit
- 80x51
- Z80 based
- HCS8

16 Bit

- Microchip PIC
- 8018x
- Renesas M16
- HCS12
- S12X

32 Bit

- ARM7
- Cortex M3
- ARM9
- PIC32
- POWERPC
- X86
- Renesas M32, R32

Recursos a Gerenciar no Projeto de um Sistema Embarcado Diferenças em Relação ao Ambiente Desktop

- Com ou sem Sistema Operacional / RTOS
- Multithreading ou single loop
- Ocupação de Memória
- Fragmentação de Memória (alocação dinâmica)
- Utilização da CPU
- Controle de Interrupções
- Custo Total da Solução (TCO)
- Recursos Limitados

C++ para Microcontroladores: Mito ou Verdade?

- Mais lento
- Binários maiores
- Abstração gera ineficiência
- Objetos são grandes
- Funções Virtuais são lentas
- C++ não é ROMable
- ISRs não podem ser feitos em C++

Embedded C++

Subset do ISO/ANSI C++ sem suporte a:

- Heranças múltiplas
- Classes base virtuais
- Run-time type identifications (RTTI)
- Excessões
- Templates e STL
- Namespaces
- New-style casts (dynamic_cast, static_cast, reinterpret_cast)

Extended Embedded C++

Inclue Suporte aos seguintes recursos:

- Namespaces
- Templates e STL compacta
- New-style casts (dynamic_cast, static_cast, reinterpret_cast)

Ambientes de Desenvolvimento

- IAR Embedded Workbench
- Keil MDK
- GNU C/C++
- Codewarrior
- Green Hills MULTI
- TI Code Composer

Customizando o GCC para C++ Embarcado

- Compilador Código Aberto e licença GNU GPL que atende às necessidades da maioria dos projetos;
- O G++ é um compilador C++ e precisa ser restrito para se comportar como um compilador embarcado (Embedded C++).
- Sem os devidos ajustes, um código em C++ pode ser muitos KBytes maior do que em C, tornando-se inviável para a maioria dos microcontroladores.

Ajustando o Makefile*

```
CPPFLAGS = -gdwarf-2 -c -mcpu=$(ARM_CPU) -mthumb-
    interwork -O \
    -mlong-calls -ffunction-sections    -fno-rtti -fno-exceptions
    -Wall

CCFLAGS = -c -mcpu=$(ARM_CPU) -mthumb-interwork -O2 \
    -mlong-calls -ffunction-sections \
    -fno-rtti -fno-exceptions -Wall -DNDEBUG

ASMFLAGS = -gdwarf2 -mcpu=$(ARM_CPU) -mthumb-interwork
```

* Referência completa em

http://www.state-machine.com/arm/Building_bare-metal_ARM_wi

Eliminando Funções não-DIET

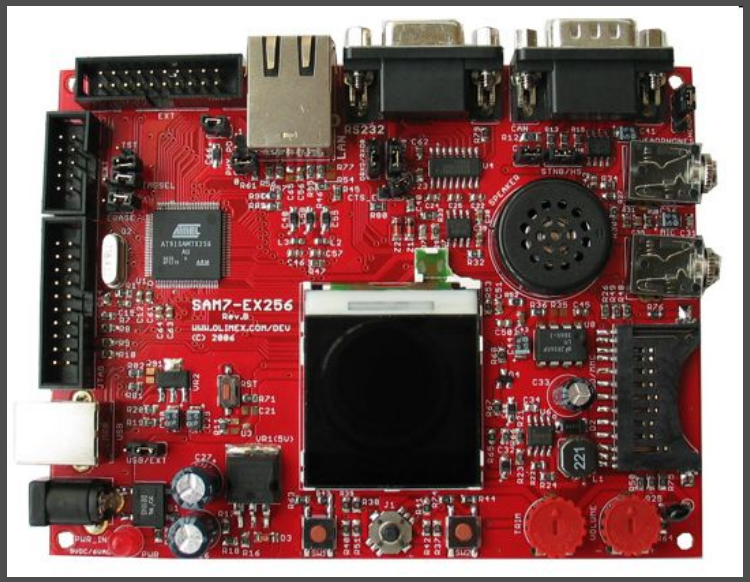
- Algumas funções da biblioteca podem disparar excessões.
- As funções podem ser sobreescritas, sendo substituidas por versões DIET

```
void *operator new(size_t size) throw()
{  return malloc(size); }
void operator delete(void *p) throw()
{     free(p); }
extern "C" int __aeabi_atexit(void *object,   void (*destructor)(void *),   void *dso_handle)
{  return 0; }
```

Estudo de Caso :
Adaptando um Código para C++ ARM7 AT91SAM7X256

- Temporizador

- Acesso Direto ao Hardware
- I2C
- Aplicação



O Código

- Aplicação com Interface Gráfica utilizando Display 132x132 pixels
- Interface com Manipulador de Interrupção do Temporizador
- Analisando o código em C (no editor)

Manipuladores de Interrupção em C++

- Manipuladores de Interrupção podem ser desenvolvidos em C++.
- Simplesmente remova a declaração do manipulador em C e declare o manipulador como uma função-membro estática na seção “protected” da classe que vai tratar o evento.

Portando para C++

- C++ se aplica ao meu projeto (heranca, polimorfismo, encapsulamento)?
- Adaptando o código para C++
- Crie Classes, “reuse” e aloque os objetos
- Converta as funções para métodos (exceto manipuladores de interrupção)
- Converta os manipuladores de interrupção.
- Converta as variáveis globais em membros protected
- Cuidado com múltiplas instâncias
- Faça o mais simples possível
- Analisando o código portado para C++ (no editor)

C++ com Núcleo de Tempo Real

- Não há grandes restrições quanto à utilização de C++ em tarefas (Tasks) em um ambiente com núcleo de tempo Real desde que as pilhas locais das tarefas sejam dimensionadas corretamente.
- Deve ser dada atenção especial à alocação e desalocação de heap, de forma a evitar fragmentação.
- Exemplos de núcleos: FreeRTOS, MicroDigital Kernel C++, uC/OS II

C++ com Linux Embarcado

- O ambiente Linux Embarcado permite que sejam utilizados praticamente todos os recursos do C++, pois os microcontroladores compatíveis são dotados de MMU e possuem maior capacidade de memória.
- Para reduzir o código e a ocupação de flash, podem ser eliminadas funcionalidades desnecessárias ou não utilizadas como rtti e excessões.
- Deve ser tomado cuidado com a alocação desnecessária de múltiplas instâncias de objetos pois os recursos não são “ilimitados” como em uma plataforma IBM/PC.

Abstração Gera Ineficiência. Mito ou Verdade?

Boa parte da abstração provida pela orientação a objeto do C++ não é tratada em “run-time” e, sim, resolvida pelo compilador.

Por exemplo: abstrações do tipo protected, private, public e const são totalmente resolvidas em “compile time”.

Binários Gerados pelo C++ são maiores. Mito ou Verdade?

O código em C++ gerado por um compilador corretamente customizado não tende a ser muito maior do que em C.

A orientação a objeto pode trazer maior estabilidade e testabilidade à aplicação.

ISRs não podem ser feitos em C++ Mito ou Verdade?

Manipuladores de interrupção podem sim ser devenvolvidos em C++.

No nosso exemplo, isto foi feito utilizando métodos estáticos protected.

C++ é mais lento do que C
Mito ou Verdade?

Algumas vezes um código em C++ pode ser mais lento do que em C.

O “dereferencing” de variáveis e funções pode ocupar um pouco de tempo e RAM.

Funções Virtuais são mais lentas? Mito ou Verdade?

As funções virtuais pagam o preço pelo polimorfismo por necessitarem da consulta em tempo de execução a tabelas anteriormente à sua execução.

C++ não é ROMable
Mito ou Verdade?

Sistemas embarcados de pequeno porte normalmente possuem maior quantidade de ROM do que de RAM.

Para um objeto ser ROMable, ele deve ser capaz de ser inicializado estaticamente (como uma struct).

Os objetos que não forem ROMable serão inicializados por construtores estáticos, ocupando mais ROM e RAM.

Uma classe poderá ser inicializada estaticamente se:

- Não descender de uma classe base
- Não possuir construtor
- Não possuir funções virtuais
- Qualquer classe interna deve seguir as mesmas regras.

Objetos são grandes Mito ou Verdade?

O tamanho dos objetos está diretamente ligado ao tamanho das classes que ele contiver.

O tamanho das classes deve ser calculado apenas considerando os métodos que estão sendo utilizados.

Os link editores modernos para C++ são capazes de remover código não utilizado.

Obrigado

Luiz de Barros Oliveira Neto

E-mail: Lboneto@gmail.com