Introdução aos Sistemas Embebidos

João M. P. Cardoso
Email: jmcardo@ualg.pt
URL: http://w3.ualg.pt/~jmcardo

2004/2005

Sistemas Embebidos

- O que são?

"*Embedded systems are computers camouflaged as non-computers. Examples include electronic devices around you, in your car, in your office, in your pocket, and many other places, that perform something intelligent.*" Tony Givargis, University of California, Irvine, USA

Sistemas Embebidos

Objectivos

- Saber distinguir entre um sistema embebido e um sistema computacional não embebido
- Ter a noção das diferenças e cuidados a ter no desenvolvimento
- Aprender algumas das características de sistemas embebidos reais
- Desenvolver software para um sistema embebido

Conteúdo Programático

- Introdução aos sistemas embebidos.
- Caracterização dos vários tipos de sistemas embebidos.
- Exemplos de sistemas embebidos.
- Mobilidade nos sistemas embebidos.
- Hardware para sistemas embebidos.
- Software para sistemas embebidos.
- Compiladores para sistemas embebidos.
- Especificação de sistemas embebidos.
- Simulação de sistemas embebidos.
- Depuração de software e de hardware.
- Co-projecto hardware/software
- Análise do desempenho

Aulas Práticas

- Estudo do microcontrolador PIC:
  - Arquitectura, conjunto de instruções
  - Ambiente de desenvolvimento e depuração
- Desenvolvimento de sistemas embebidos utilizando microcontroladores;
- Aspectos práticos relacionados com o desenvolvimento de um sistema embebido;
- Projectos utilizando microcontroladores PIC;

Aulas Práticas

- Equipamento e Informação
  - Ambiente de desenvolvimento para o microcontrolador PIC: Microchip MPLAB® IDE v6.60,
  - PICkit Flash Starter Kit v1.2, Microchip Technology Inc.,
  - PICkit 1 FLASH Starter Kit User's Guide, Microchip Technology Inc., “PIC12F629/675 Data Sheet 8-Pin FLASH-Based 8-Bit CMOS Microcontrollers”, Microchip Technology Inc., 2003,

Bibliografia

- Wayne Wolf, Computers as Components: Principles of Embedded Computing Systems Design, Morgan Kaufmann Publishers, 2000.
- Arnold S. Berger, Embedded Systems Design: An Introduction to Processes, Tools and Techniques, CMP Books, 2001
- Michael Barr, Programming Embedded Systems in C and C++, O'Reilly & Associates, Inc., 1999.
- F. Vahid, T. Givargis, Embedded System Design: A Unified Hardware/Software Introduction, John Wiley and Sons, October 2001.

Suporte à Disciplina

- Página web da disciplina:
  - http://w3.ualg.pt/~jmcardo/ensino/ise2004
  - Slides PowerPoint das aulas
  - Notícias e Avisos
  - Links para a documentação
- Mailing-list
  - Fct-ise@ualg.pt
- Horário de atendimento
  - Quartas: --, Quintas: --

Avaliação

- Trabalho Prático: 40%
  - Efectuado nas últimas 4 aulas práticas
  - Apresentado pelo grupo de alunos ao professor da prática e discutido na última aula prática do semestre
  - Requer relatório
- Mini-teste: 20%
  - Efectuado a meio do semestre nas aulas práticas
- Exame: 40%
  - Época Normal ou de Recurso
- Aprovação: >= 10 valores
  - Mas: Nota do trabalho prático >= 10 valores e
  - Nota do exame de época normal ou de recurso >= 6,5 valores
- Nota que é tida em conta nos exames é a nota do último exame realizado pelo aluno

Introdução aos Sistemas Embebidos

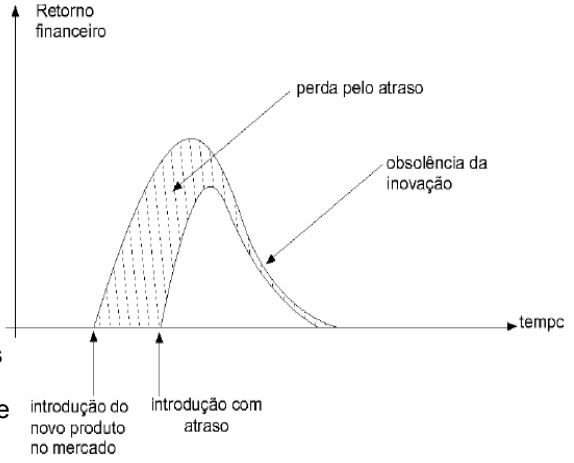
Introdução

Sistemas Embebidos

- Sistemas computacionais projectados para desempenhar tarefas específicas.
- Qualquer dispositivo que realize computação mas que não é um computador.
- Satisfaz requisitos que são por vezes muito diferentes dos requisitos para computadores de propósito geral
- O projecto destes sistemas tira partido da especificidade
- Sistemas Embebidos? http://www.wikipedia.org/wiki/Embedded_system

Factores Importantes

- Contínua evolução tecnológica
- Pressão do mercado por novos produtos
- Cada vez menos tempo para desenvolver um produto
- Novos produtos têm uma vida cada vez mais curta
- Retorno financeiro do projecto deve ser obtido também em poucos meses
- Atrasos de poucas semanas no lançamento de um produto pode comprometer seriamente os ganhos esperados



Origem: Luigi Carro, Flávio R. Wagner, "Sistemas Computacionais Embarcados", JAI'03 – XXII Jornadas de Atualização em Informática, Campinas, Brasil, Agosto 2003.

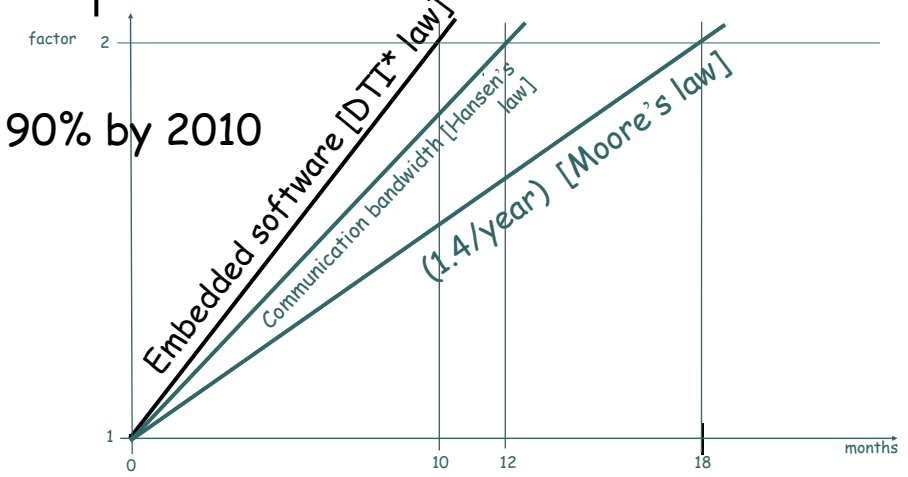
Factores importantes

- Custo
  - Deve ser baixo para que o sistema possa ser competitivo
    - Não vamos utilizar um Pentium 4 para o sistema electrónico a embeber numa varinha mágica!
  - A selecção da arquitectura, e da organização computacional (incluindo periféricos) depende dos requisitos
- Baixo custo, mas deve permitir projectar um sistema que satisfaça os requisitos

Sistemas Embebidos

- Variáveis a considerar no projecto
  - Desempenho (restrições temporais)
  - Consumo de potência
- Outros requisitos
  - Tamanho físico do sistema, fiabilidade, etc.

Tendências



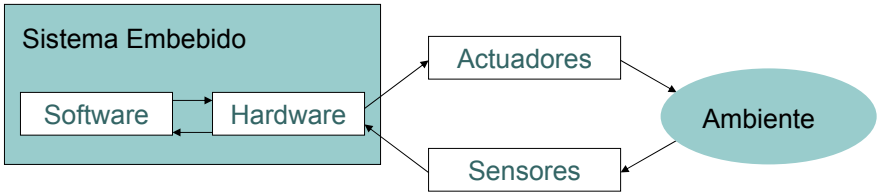
90% by 2010

*) Department of Trade and Industry, London

Origem do slide: Prof. Hartenstein

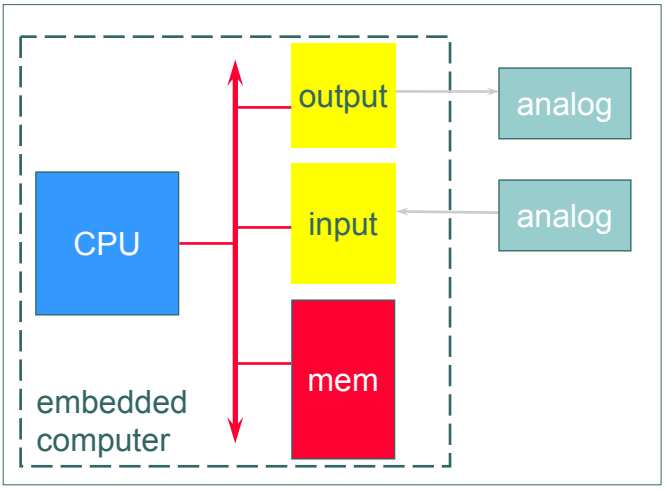
Sistemas Embebidos

- Formados por componentes hardware e componentes software, com interface a actuadores e sensores



Sistemas Embebidos

- Muitas arquitecturas, muitas soluções (não estamos sob domínio do "PC")



Sistemas Embebidos

- Os mais simples não têm disco rígido, não têm sistema operativo, etc.
- Muitos deles não necessitam de alto-desempenho (o relógio pode funcionar a frequências muito baixas)
- Em muitos deles os dados podem ser representados por palavras pequenas (4, 8, 16, 32 bits)

CPU

- Microcontrolador
  - Inclui interfaces com o exterior, memória, temporizadores, cães-de-guarda, etc.
  - Normalmente pouco poderoso em termos de cálculo
  - 4-bits, 8-bits, 16-bits
- Microprocessador
  - Sistemas computacionais mais exigentes
- DSP (*Digital Signal Processor*)
  - Específicos para sistemas embebidos com processamento digital de sinais e que requerem alto-desempenho

Periféricos

- Interface I/O
  - Ethernet, USB, ADC, DAC, etc.
- Temporizadores
- Interfaces são normalmente mais específicos do que os interfaces a que nos habituamos nos PCs (teclado, rato, ecrã, etc.)

Links

- Revistas Online
  - http://www.circuitcellar.com/
  - http://www.embedded.com
  - http://www.ddjembedded.com/
- Empresa com o Microcontrolador (PIC) que vamos utilizar
  - http://www.microchip.com

Introdução aos Sistemas Embebidos

Introdução e Exemplos

Exemplo de Sistemas Embebidos

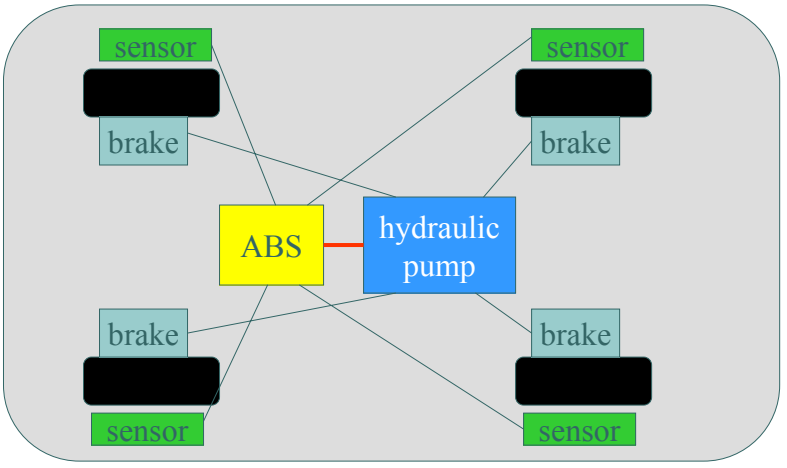
- Sistemas electrónicos em vários dispositivos:
  - Impressoras, leitor de MP3, leitor de DVDs, PDA (*Personal Digital Assistant*), telemóvel, aparelhagem áudio, máquina fotográfica/de filmar digital, consola de jogos, televisões, micro-ondas, etc.
  - Placa gráfica de um PC é um exemplo de um sistema embebido (neste caso embebido num computador)
  - Existe um sistema embebido no teclado dos PCs

Exemplos: Automóvel

- Automóveis de gama alta podem ter cerca de 100 microprocessadores:
  - Microcontroladores de 4-bit para o cinto de segurança;
  - Microcontroladores para os dispositivos no tabelier;
  - Microprocessador de 16/32-bits para controlar o motor
  - Para o ABS (*antilock brake system*), ...
  - ...

Exemplos: Automóvel

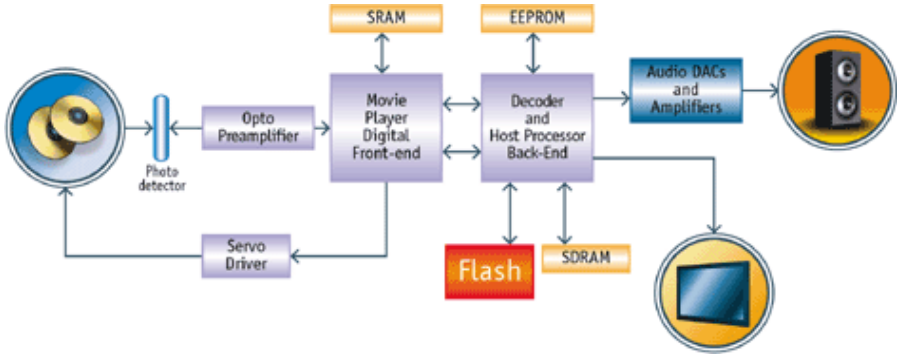
- ABS



Fonte: transparências do livro *Computers as Components*, Morgan Kaufman, 2000

Exemplos: Leitor de DVDs

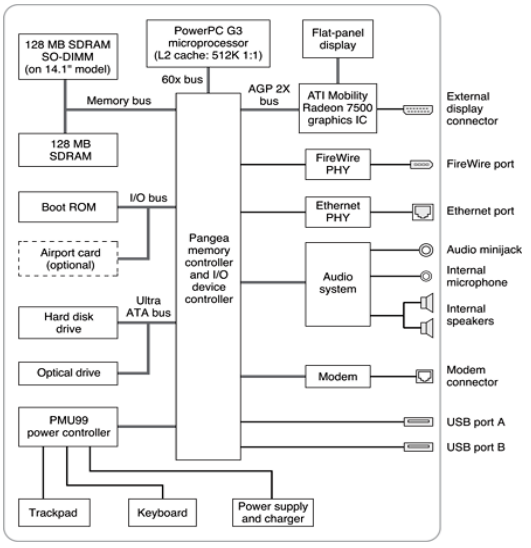
- Leitor de DVDs com memória flash
  - Possibilidade de actualização de software



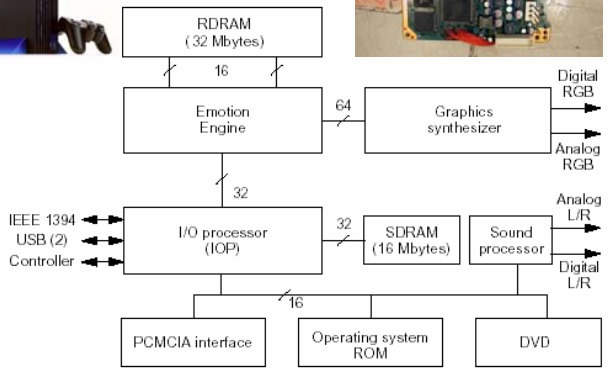
Fonte: http://us.st.com/stonline/prodpres/memory/flash/fl_dvd.htm

São as diferenças suficientemente visíveis?

- O que distingue uma arquitectura para sistemas embebidos e uma arquitectura para um computador?
- Diagrama de blocos (organização computacional do iBook, um computador): é uma arquitectura de um sistemas embebido ou não?



Exemplo: Sony PlayStation2



- CPU de 128-bits ("Emotion Engine")
- GS (Graphics Synthesizer)
- SPU2 (Dynamic Sound Processor)
- Processador de I/O
- Sistema de DVD/CD-ROM
- 32 MB de memória Rambus

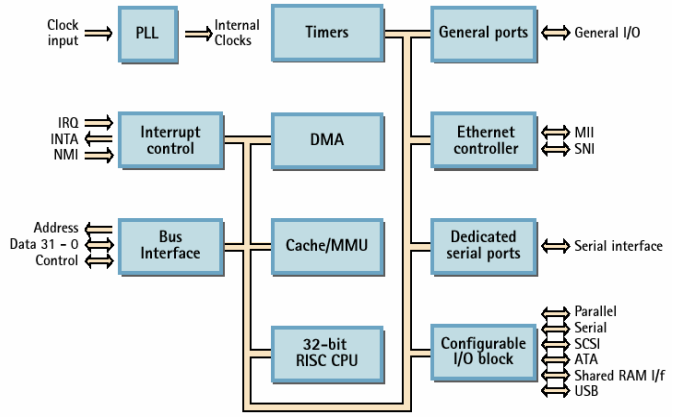
Fonte: http://www.computer.org/micro/mi1999/pdf/m6020.pdf

Exemplo de Arquitecturas

- *System-On-a-Chip* (SOC)
  - Toda a organização computacional (processador, periféricos, memória, barramento de sistema, etc.) no mesmo chip!
  - Similar ao *computer-on-a-chip*

System-On-a-Chip (SOC)

- Exemplo: ETRAX100Lx (32-bits, 100 MHz)
  - Periféricos embebidos



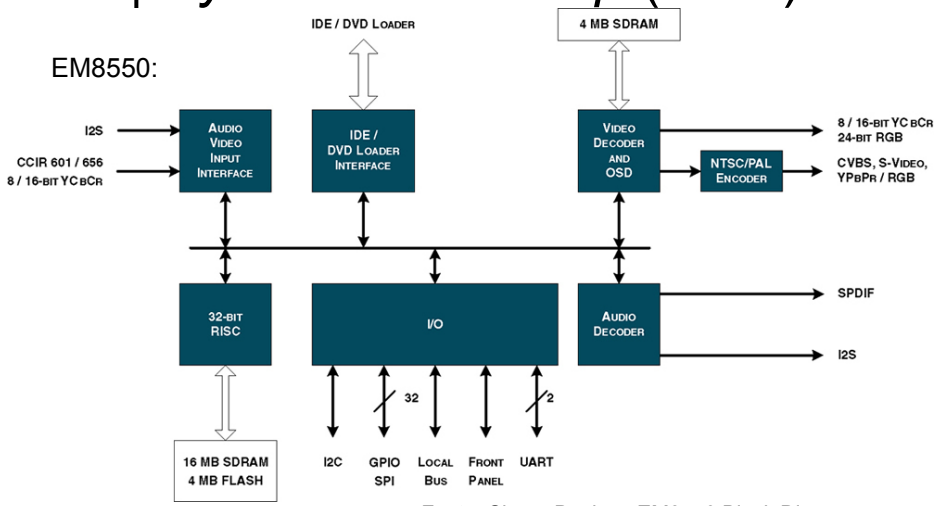
System-On-a-Chip (SOC)

- EM8550: *Digital Media Processor for SDTV Consumer Appliances*
  - Vários suportes:
    - Descodificação MPEG-4, MPEG-2 e DVD
    - Suporte a MPEG-2 e MPEG-4 sobre IP
  - Utilizado em leitores portáteis de DVD, de media
  - Processador RISC embebido para executar o software de sistema e aplicações
- Muitas outros suportes...

Fonte: Sigma Designs EM8550 Block Diagram, http://www.sigmadesigns.com/products/em8550.htm

System-On-a-Chip (SOC)

EM8550:



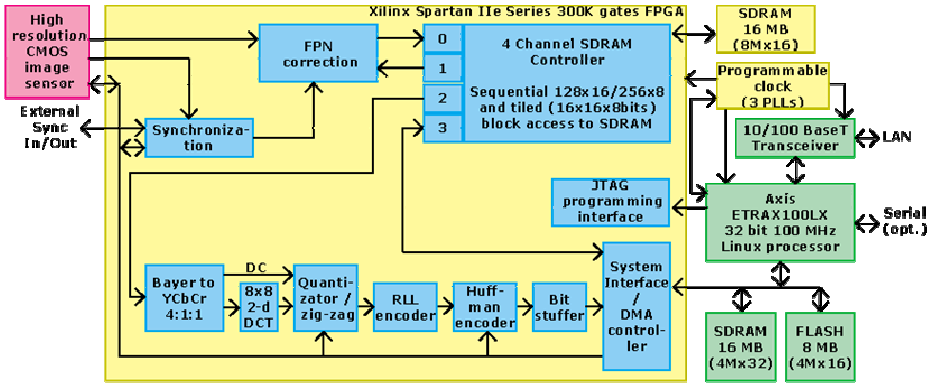
Fonte: Sigma Designs EM8550 Block Diagram, http://www.sigmadesigns.com/products/em8550.htm

Exemplo: camera em rede

- Modelo 303 da câmera (http://www.elphel.com/)
  - Alta-resolução
  - Lento: o ETRAX100LX requer cerca de 5 segundos para compressão JPEG de uma trama a cores de 1280x1024 pixels
- Aceleração da compressão JPEG:
  - Chip de compressão JPEG-2000 da Analog Devices
  - Utilização de uma FPGA (*Field-Programmable Gate Array*) para executar a compressão
- Modelo 313: Com uma FPGA: 15 fps @1280x1024

Exemplo: camera em rede

- Modelo 313 (http://www.elphel.com/)



Lições a aprender

- Projecto pode implicar:
  - Necessidade de utilizar certos dispositivos de hardware específicos para, por exemplo, satisfazer restrições temporais
- Pode implicar a migração de certas tarefas do software para o hardware

Sistemas Embebidos

- Alguns sistemas são chamados de sistemas embebidos de tempo-real (*real-time embedded systems*)
  - Têm *deadlines* para realizar as operações
- *Hard real time*: falha do *deadline* causa falha do sistema
- *Soft real time*: falha do *deadline* causa degradação do desempenho

Links

- Museu da Intel: http://www.intel.com/intel/intelis/museum/INDEX.htm
  - História dos Microprocessadores: http://www.intel.com/intel/intelis/museum/exhibits/hist_micro/index.htm
  - Aprender acerca de Tecnolgia: http://www.intel.com/education/sections/section6/index.htm

Introdução aos Sistemas Embebidos

Panorâmica Geral

Sistemas Embebidos

- História de um dos primeiros sistemas embebidos:
  - 1969: Busicom, uma empresa Japonesa, pediu à Intel para desenhar um conjunto de CIs um para cada novo modelo das suas calculadoras
  - A Intel desenvolveu o microprocessador 4004, que poderia ser utilizado em todos os modelos da calculadora
- A utilização de microprocessadores reveste-se de várias vantagens: uma delas é que o mesmo modelo de microprocessador pode ser utilizado por todos os modelos de uma mesma linha de produtos

Sistemas Embebidos

- Composição
  - Microprocessador e Software
  - Memória volátil ou não-volátil, regravável ou não-regravável
  - Dispositivos de I/O
  - Outros periféricos…

Sistemas Embebidos

- A utilização de um microprocessador de 8-bits
  - Espaço de endereçamento bastante pequeno, permite a integração da memória de programa no mesmo chip do µP
  - Quando a arquitectura inclui bastantes registos internos, não existe necessidade de memória para dados

Sistemas Embebidos

- São programáveis
  - Software tem papel fulcral no desenvolvimento
- Linguagens de programação
  - Maioritariamente em C (algumas vezes são embebidos pedaços de assembly)
  - Java tem-se tornado uma séria opção (veja-se o caso da proliferação dos jogos Java)
  - Ada, C++ em alguns casos
  - Assembly continua a ser utilizado

Porquê C?

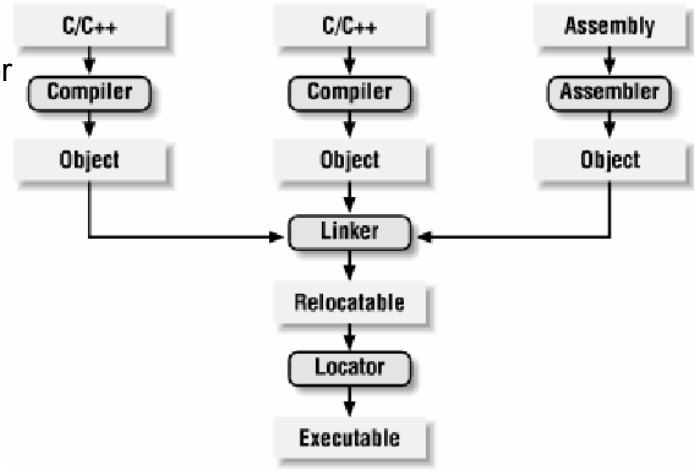
- Linguagem de alto-nível
  - Abstrai o programador dos detalhes da arquitectura
- C, permite utilizações de baixo-nível
  - Permitem aceder aos recursos de hardware
    - Facilidade na interacção com o hardware
  - Possibilidade de embeber pedaços de assembly no código C (instrução: asm{})

Desenvolvimento de Software

- Abordagem idêntica ao desenvolvimento de software para PCs com algumas nuances…
- Ressalvas:
  - Desenvolvimento é realizado num computador de propósito geral (ex.: PC)
  - Com a ajuda de simulador, depurador, etc.
  - Testes finais são depois realizados no próprio sistemas embebido
  - …

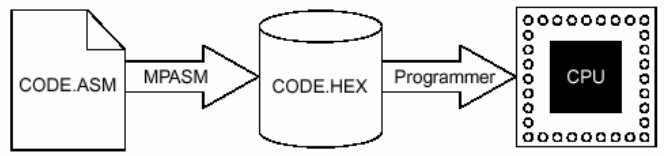
Desenvolvimento de Software

- Compilador
- Assemblador
- Linker
- Locator



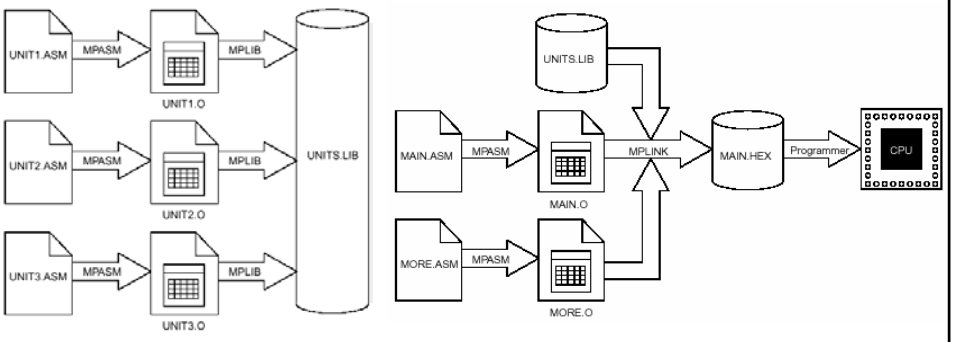
Desenvolvimento de Software: µC PIC

- Gerar código para programar directamente o PIC



Desenvolvimento de Software: µC PIC

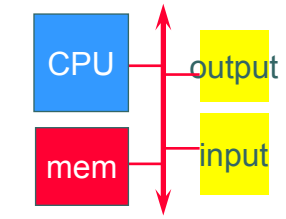
- Gerar código para ser utilizado por outros programas



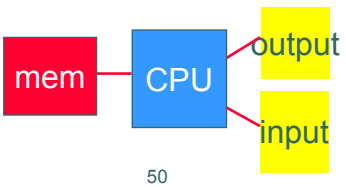
Introdução aos Sistemas Embebidos

Organização Computacional

Organização Computacional



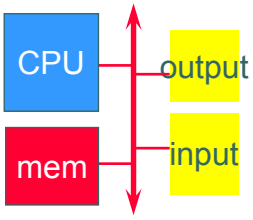
- Organização computacional baseada em barramento



- Organização computacional baseada em ligações ponto-a-ponto

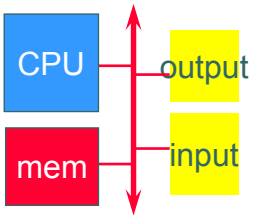
Organização Computacional

- Baseada na partilha de um ou mais barramentos
  - Facilidade de adicionar novos dispositivos
  - Menor desempenho
  - Menores custos
- Normalmente são utilizados dois barramentos:
  - De dados
  - De endereços



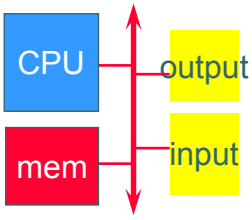
Transações no barramento

- Leitura
  - CPU coloca endereço no barramento
  - CPU sinaliza no sinal de controlo que vai ler
  - Memória coloca palavra no endereço e sinaliza os sinais de controlo apropriados



Transacções no barramento

- Escrita
  - CPU coloca endereço no barramento
  - CPU coloca palavra no barramento
  - CPU sinaliza no sinal de controlo que vai escrever
  - Memória guarda palavra no endereço dado



Transacções no Barramento

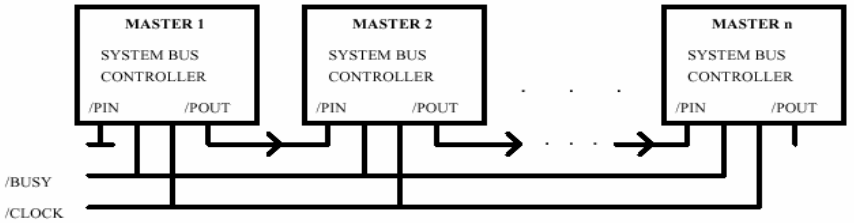
- Normalmente o CPU é o único mestre do barramento (controlo centralizado)
  - Único dispositivo que pode iniciar transacções no barramento
- Pode haver múltiplos mestres (controlo distribuído)
  - Quando existem vários CPUs
  - Quando os dispositivos de I/O podem iniciar transacções
  - Requer esquema de arbitragem

Transacções no Barramento

- Vários pedidos de transacções no mesmo barramento (oriundos de diferentes dispositivos)?
  - Árbitro
  - Arbitragem baseada em prioridades fixas (*daisy-chain*), selecção aleatória, ...

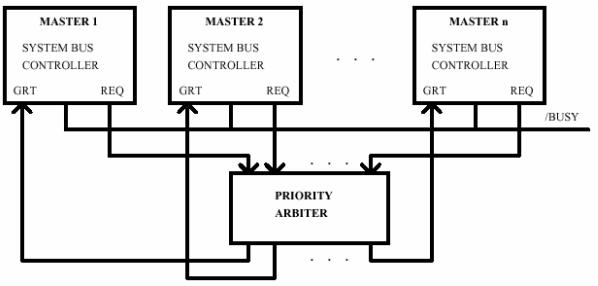
Esquema de arbitragem

- *daisy-chain*
  - Prioridades diminuem da esquerda para a direita
  - /PIN = 0? (na transição ascendente do /CLOCK)
    - IF NOT: espera
    - IF YES: /BUSY = 1?
      - IF NOT: espera
      - IF YES: /BUSY := 0 (e pode utilizar o barramento)



Esquema de arbitragem

- Arbitragem em paralelo
  - Esquema de prioridades pode ser aleatório, fixo, *round-robin*, etc.
  - GRT: Grant
  - REQ: Request

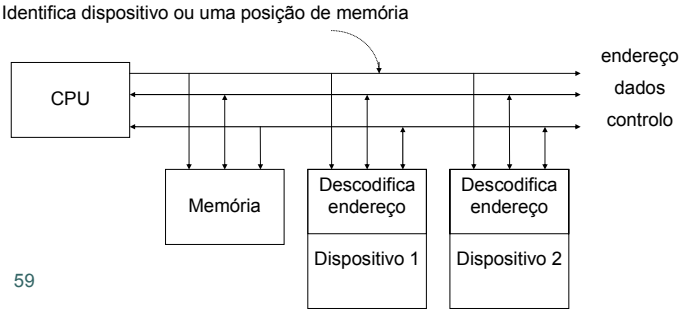


Comunicação CPU-Dispositivos

- Como é que o CPU indica que quer escrever ou ler de um determinado dispositivo presente no barramento?
  - I/O mapeado em memória
  - Sinal a indicar que a transacção é com os dispositivos de I/O (implica instruções no CPU)

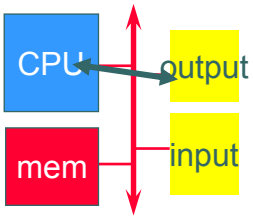
Comunicação CPU-Dispositivos

- I/O mapeado em memória
  - Porções do espaço de endereçamento são dedicadas aos dispositivos
  - Ex: a escrita num determinado endereço indica o envio de um comando para um determinado dispositivo



Comunicação CPU-Dispositivos

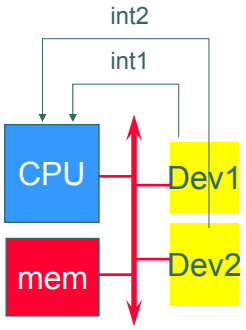
- Como é que o CPU sabe que pode ler/escrever de/em um dispositivo?
  - *Polling*: testando repetidamente cada dispositivo (normalmente verificando um registo de controlo)
    - Implica que o CPU esteja sempre a verificar em vez de fazer outras coisas!
    - Exemplo:




```
// polling a bit in the status register of the IODevice1
for(;;) {
      if(IODevice1.getStatus() == 0x1)
          // do actions
}
```

Comunicação CPU-Dispositivos

- Como é que o CPU sabe que pode ler/escrever de/em um dispositivo?
  - Interrupções: dispositivos sinalizam o CPU via linhas especiais
    - O CPU pode executar outras tarefas...

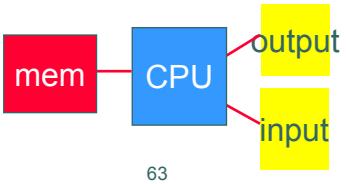


Melhoria na transferência de dados

- Hardware para DMA (*Direct Memory Access*)
  - Transfere blocos de dados da memória para um dispositivo de I/O e vice-versa sem intervenção contínua do CPU
    - CPU pode ocupar-se de outras coisas!
    - CPU indica o endereço base e o número de bytes e depois a transferência é concretizada sem a sua intervenção
  - Quando a transferência termina o controlador sinaliza o CPU
  - O controlador é um dos mestres do barramento
  - Integrado na maioria dos controladores dos dispositivos de I/O

Organização Computational

- Organização computacional baseada em ligações ponto-a-ponto
  - Requer que o CPU tenha um número variável de portos de I/O
  - Requer instruções para lidar com esses portos

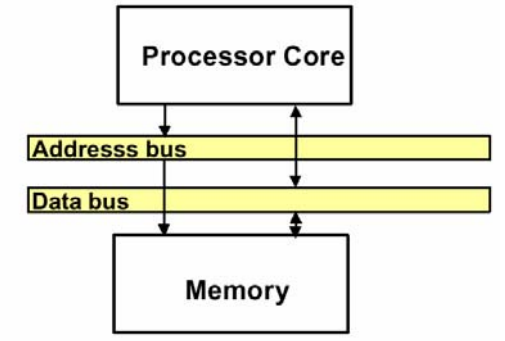


Organização Computacional

- Em alguns sistemas embebidos a organização computacional é baseada nos dois esquemas
- Alguns dispositivos de I/O são ligados a portos de I/O do microprocessador

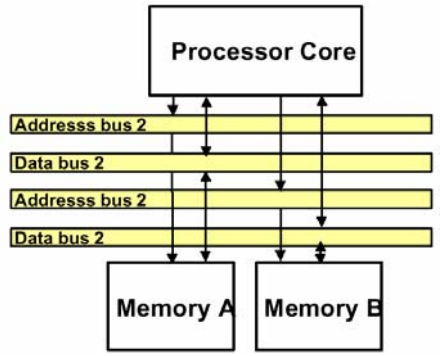
Organização CPU-Memória

- Von Neumann: dados e programa na mesma memória



Organização CPU-Memória

- Harvard: dados e programa em memórias separadas



Introdução aos Sistemas Embebidos

Representações Numéricas

Representações Numéricas

- representações numéricas muitas das vezes diferentes das utilizadas nos PCs
- Por vezes porque o CPU não tem suporte primitivo a alguns dos tipos de dados (ex., vírgula flutuante)
- Por vezes por que o CPU tem uma palavra de comprimento reduzido (ex. 8 bits)

Representações Numéricas

- Representações numéricas
  - Complemento para dois (utilizada nos PCs para representar números inteiros)
  - Vírgula flutuante (utilizada nos PCs para representar números reais)
    - formato standard IEEE 754
  - Vírgula fixa
    - Utilizados N bits à esquerda da vírgula e M bits à direita da vírgula
    - Os valores de N e de M dependem da precisão necessária para que os erros sejam desprezáveis (requer simulações e estudos prévios)

Vírgula Flutuante

- IEEE 754: 32 bits (precisão simples)
  - 1bit para identificar o sinal (1 indica '-')
  - 8 para o expoente (e)
  - 23 para a fracção (f)
  - Valor = (+/-) $1,f \times 2^{e-127}$
  - 1 10000001 01000000000000000000000
  - $-1,25 \times 2^2 = -5$
- Precisão dupla (64 bits)

Vírgula Fixa

- Por que motivo se utiliza a vírgula fixa?
  - CPU sem suporte a operações de vírgula flutuante (implementação em software é demasiado complexa e requer muitos ciclos de relógio)
  - Em alguns sistemas os erros de precisão oriundos da utilização de vírgula fixa são desprezáveis
  - Apenas requer operadores inteiros (algumas das vezes alinhamento: deslocamento de bits)
    - Operações mais rápidas, menos dispendiosas

Vírgula Fixa

- Multiplicação: representações com 4 bits à esquerda e 4 bits à direita da vírgula
  - 2,5 => 0010,1000
    - Supondo registos de 8 bits: 00101000 (40)
  - 1,25 => 0001,0100
    - Supondo registos de 8 bits: 00010100 (20)
  - 2,5 x 1,25 (3,125) = (00101000 x 00010100) >> 4 = 01100100000 >> 4 (800>>4) = 00110010 (50)
  - 0011,0010 (3,125)
- Requer alinhamento no final

Vírgula Fixa

- Adição: representações com 4 bits à esquerda e 4 bits à direita da vírgula
  - 2,5 => 0010,1000
    - Supondo registos de 8 bits: 00101000 (40)
  - 1,25 => 0001,0100
    - Supondo registos de 8 bits: 00010100 (20)
  - 2,5 + 1,25 (3,75) = 00101000 + 00010100 = 00111100 (28)
  - 0011,1100 (3,75)
- Não requer alinhamento no final

Introdução aos Sistemas Embebidos

Representações binárias

Rever operações e representações:

http://w3.ualg.pt/~jmcardo/ensino/sistdigitais01/AulasTeoricas/aula4.pdf

Representações binárias utilizadas

- N-bits sem sinal
- N-bits em complemento para dois
- Vírgula fixa (M,N)
- Vírgula flutuante (sinal: 1 bits, expoente: N bits, mantissa: M bits)

Overflow

- quando o valor da amplitude requer um número de bits que ultrapassa o nº de bits disponíveis na representação

A flag de Carry

- Indica um bit de transporte depois do último bit da representação
- Exemplo com 8-bits (utilizando complemento para dois)
  - FFh + 1h = 00h (carry = 1)
  - 02h – 03h = FFh (carry = 0)
  - 03h - 02h = 01h (carry = 1)

A flag de overflow

- Indica se um número com ou sem sinal ultrapassou a gama de valores possíveis
  - Para 8-bits, com sinal, a gama é de -128 a +127
- Exemplos com 8-bits
  - 7Fh + 1h = 80h, queríamos 127 + 1 = 128 (obtivemos –128) (carry = 0; overflow = 1)
  - 80h – 1h = 7F, queríamos -128 -1 = -129 (obtivemos +127) (carry = 0; overflow = 1)
  - FFh + FFh = FEh {(-1) + (-1)} = -2; (carry = 1; overflow = 0)

Subtracção de números inteiros sem sinal

- A-B
- Determinar se o resultado é negativo (se A<B)?
- Bit de carry determina o sinal do resultado:
  - Carry=1: resultado é positivo sem sinal
  - Carry=0: resultado é negativo com sinal

Overflow: adição

- Números sem sinal
  - Ocorre quando carry = 1
- Números com sinal
  - Pode ocorrer quando operandos têm o mesmo sinal
    - Adição de 2 números positivos deve dar um positivo
    - Adição de 2 números negativos deve dar um negativo
  - Quando o bit de carry que entra no último bit é diferente do bit de carry que sai do último bit

Overflow: subtracção

- Números sem sinal
  - Não há overflow
- Números com sinal
  - Quando os dois números têm sinais diferentes e carry=1
  - Quando bit de carry que entra no último bit é diferente do bit de carry que sai do último bit

Flags possíveis

- Z: indica se o resultado é zero
- C: indica o valor do bit de transporte
- S: indica o sinal do resultado
- O: indica overflow no resultado

Comparação de dois números utilizando subtracção: A-B

- Operandos sem sinal
- Z: igualdade/desigualdade
- C: A<B (C=0)
  A>B (C=1)
- S: sem significado
- O: sem significado

- Operandos com sinal
- Z: igualdade/desigualdade
- C: sem significado
- S e O:

If(S xor O == 1) A<B
Else A >=B

Comparação de dois números utilizando subtracção

- Caso 1: S = 1 e O = 0
  - A-B parece negativo, e
  - Não houve overflow, então o resultado está na gama de valores permitida e é válido
  - Então A tem de ser menor do que B
- Exemplo com 4 bits:
  - A = 0010 (2)
  - B = 0100 (4)
  - A-B = 1110 (-2)

Comparação de dois números utilizando subtracção

- Caso 2: S = 0 e O = 1
  - A-B parece positivo, mas
  - Deu overflow, então o resultado está fora da gama permitida e o sinal não está correcto
  - A é menor do que B
- Exemplo (8-bits): (-128) - (1) = (+127)
  - Resultado (+127) com overflow. A resposta deveria ter sido -129.
  - Resultado parece positivo, mas ocorreu overflow
  - Assim (-128) é menor do que (1)

Comparação de dois números utilizando subtracção

- Caso 3: S = 0 e O = 0
  - A-B é positivo, e
  - Não houve overflow, então o bit de sinal está correcto
  - Então A é maior do que B (podem ser iguais!)
- Caso 4: S = 1 e O = 1
  - A-B parece negativo, mas
  - Houve overflow, então o sinal está incorrecto
  - A é maior do que B

Introdução aos Sistemas Embebidos

Implementação de funções aritméticas em software

Implementação de funções aritméticas em software

- Muitas das vezes o CPU não suporta certas operações aritméticas que são necessárias na aplicação
- Exemplo:
  - Multiplicação de inteiros quando não existe instrução de multiplicação (veja-se o PIC)
  - Divisão de inteiros quando não existe instrução de divisão (veja-se o PIC)
  - Raiz quadrada de um número inteiro...
  - ... E muitos outros casos

Implementação de funções aritméticas em software

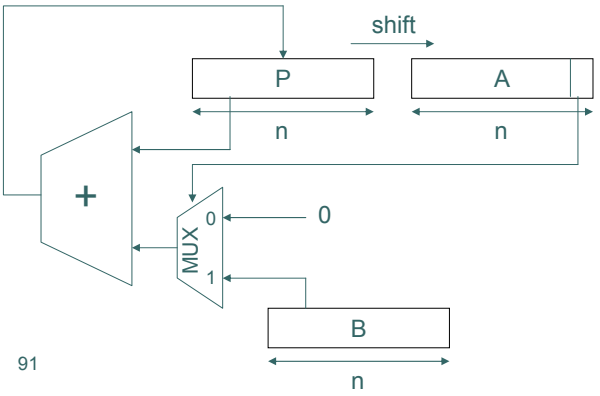
- Normalmente tem de se implementar a função aritmética com base nas primitivas existentes (ISA do CPU)
- Pode-se também utilizar uma tabela de valores (chamada de *lookup table*)
- Um dos métodos utilizados para funções mais complexas é o método iterativo de Newton-Raphson para encontrar o zero de uma função
- http://numericalmethods.eng.usf.edu/mcd/gen/03nle/mcd_gen_nle_lea_newton.htm

Multiplicação sem sinal

- Vamos supor que o CPU não tem unidade de multiplicação e queremos determinar A x B
- $A = a_{n-1}a_{n-2}...a_0$; $B = b_{n-1}b_{n-2}...b_0$; $P = 0$
- Se o bit menos significativo de A é 1, então soma B a P; senão soma 0 a P
  - O resultado é colocado em P
- Os registos P e A são deslocados para a direita de um bit, com o bit menos significativo de P a ser colocado no bit mais significativo de A (o bit menos significativo de A é deitado fora)
- Deve ser atribuído ao bit mais significativo de P o valor do bit de transporte (carry-out) da adição B+P efectuada
- Depois de n passos o resultado está em P e A, em que A tem os bits menos significativos da multiplicação

Multiplicação sem sinal

- Diagrama de Blocos



Divisão sem sinal

- Vamos supor que o CPU não tem unidade de divisão e queremos determinar A / B
- A = $a_{n-1}a_{n-2}...a_0$; B = $b_{n-1}b_{n-2}...b_0$; P = 0
- Deslocar um bit para a esquerda o par (P, A)
- Subtrair o conteúdo de B do registo P;
- Se o resultado da subtracção dá um número negativo, colocar o bit menos significativo de A a 1, senão colocá-lo a 0
- Se o resultado da subtracção dá um número negativo, colocar o valor anterior de P (adicionar o conteúdo de B a P)
- Depois de n passos o registo A contém o quociente e o registo P contém o resto

Divisão sem sinal

- Diagrama de Blocos

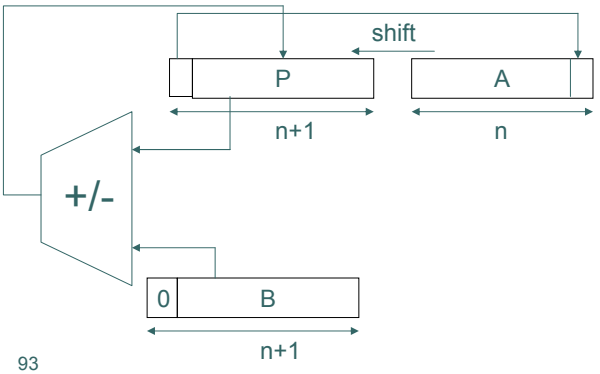
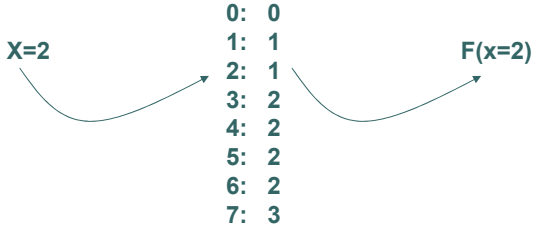


Tabela de valores

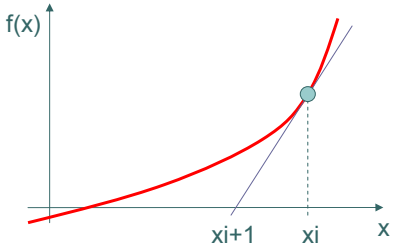
- Pretende-se implementar: f(x)=sqrt(x)
- Vamos supor que x é um número inteiro positivo representado por 3 bits:



- Entrada com 3 bits requer tabela de 8 valores: n bits de entrada => tabela de 2^n valores!

Newton-Raphson

- Método iterativo para encontrar o zero de uma função
  - Se x_i é uma estimativa do zero de uma função f(x) então x_{i+1} é uma estimativa melhor
  - Para determinar x_{i+1} é preciso encontrar a intersecção da tangente de f(x) no ponto x_i com o eixo x
  - Se iterarmos obtemos cada vez melhores estimativas
  - O método auto-corrige-se no sentido de que um erro em x_0 não implica insucesso!



Newton-Raphson

- Encontra um zero para a função f(x)=0 iterativamente utilizando a fórmula (a dedução desta fórmula é um bom exercício):

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Necessita de valor inicial x_0
  - Pode ser obtido utilizando uma tabela de valores
- Precisão aumenta com o número de iterações

Divisão com Newton-Raphson

- A divisão A/D = A × (1 / D)
- 1º é necessário escolher uma função f(x) que seja zero quando f(1/D)

$$f(x) = D - \frac{1}{x}$$

$$f'(x) = \frac{1}{x^2}$$

Divisão com Newton-Raphson

- Logo:

$$x_{n+1} = x_n \times (2 - D \times x_n) \qquad (1)$$

- Assim o quociente será:
  - $Q = x_i \times A$
- Caso seja necessário o resto:
  - $R = A - Q \times D$
- Etapas do método:
  - Escolher x0 de forma a que $0 < x0 < 2/D$
  - Iterar a equação (1) um número de vezes
- O número de bits correcto duplica em cada iteração
- O número de iterações depende da estimativa inicial

Divisão com Newton-Raphson

- Como determinar sqrt(x) utilizando o método?

Introdução aos Sistemas Embebidos

Desenvolvimento do Sistema Embebido

Desenvolvimento do Sistema Embebido

- Ciclo de vida:
  - Especificação do produto
  - Fraccionamento do Projecto nos seus componentes hardware e software
  - Iteração e refinamento do fraccionamento
  - Tarefas de projecto independentes do hardware e do software
  - Teste do produto e entrega
  - Manutenção e actualização

Selecção do CPU

- É o CPU capaz de desempenho suficiente?
- Satisfaz o CPU os consumos de potência requeridos?
- É o CPU suportado pelo Sistema Operativo apropriado?
- É o CPU suportado por ferramentas de software adequadas?
- É o CPU suportado por uma boa placa de desenvolvimento?
- É o CPU disponibilizado numa implementação adequada?

Selecção do CPU

- Outros factores
  - *Time-to-market* (tempo de introdução no mercado) pode implicar a escolha do CPU com o qual a equipa de desenvolvimento está habituada
  - Relação com a empresa do CPU
  - ...

uP versus uC

- uP
  - Chip sem periféricos ou dispositivos de suporte adicionais
  - Apenas contém CPU
- Sistema baseado num uP => Chips separados
- uC
  - Chip contém CPU e um conjunto mínimo de dispositivos adicionais
- Sistema baseado num uC => um único chip

Vantagens dos uCs

- Menor custo do sistema
  - Um único chip a substituir vários chips
- Maior fiabilidade
  - Menos chips, menos interligações na placa
- Melhor performance
  - Componentes internos optimizados para o sistema (e.g., ligação ao CPU)
- Mais rápido
  - Sinais viajam no chip

Desempenho

- Como verificar se o uP é capaz do desempenho pretendido?
- Utilizar benchmarks com conjuntos de programas reais

Benchmarks (exemplo)

- EDN *Embedded Microprocessor Benchmark Consortium* (EEMBC)
  - Automóvel/Indústria
  - Consumidor
  - Redes
  - Automação no escritório
  - Telecomunicações
- Preço: cerca de $10K

Desempenho

- Uma empresa precisava de melhorar a performance de um sistema em 2X e para isso decidiu mudar o uP do sistema para uma nova versão do uP a operar com o dobro da frequência de relógio. No entanto, após alteração do sistema só conseguiu 20% de melhoria no desempenho.
  - Porque acha que isto aconteceu?

Desempenho

$$Speedup = \frac{Tempo\_de\_execução\_da\_tarefa\_sem\_melhorias}{Tempo\_de\_execução\_da\_tarefa\_com\_melhorias}$$

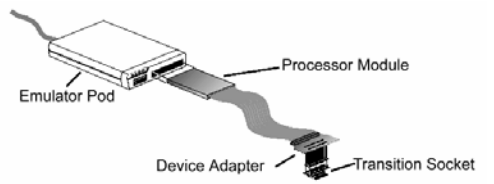
- Lei de Amdahl: a melhoria do desempenho obtida utilizando um modo mais rápido é limitada pela fracção de tempo que o modo rápido pode ser utilizado

Desempenho: Exemplo da Lei de Amdahl

- Para viajar entre dois pontos é necessário passar por dois percursos
  - Um percurso de 200Km com auto-estrada e com possibilidade de viajar sem limite de velocidade
  - Um percurso de 100Km com estrada em terra em que o limite foi fixado em 10Km/h
- Há duas hipóteses:
  - Utilizar um Ferrari com velocidade máxima de 200Km/h
  - Utilizar uma bicicleta que consegue fazer uma média de 40Km/h no primeiro percurso
  - A melhoria no desempenho (speedup) de utilizar o Ferrari em vez da bicicleta é: SpeedUp = 15H/11H ≅ 1,36
  - E se o Ferrari fizesse o primeiro percurso em 0 segundos?

Ferramentas

- Cross-compiler (compilador cruzado)
- Suporte a depuração
- Emuladores
- Simuladores, etc.



Introdução aos Sistemas Embebidos

Co-Projecto Hardware/Software

Fraccionamento hardware/software

- 80386 era um uP sem suporte hardware a operações de virgula flutuante
- 80387 foi um co-processador numérico de virgula flutuante
  - Operações aritméticas, trigonométricas, logarítmicas, com expoente
- Uma aplicação ficava 10x mais rápida com a utilização do co-processador
- O 80486 foi o primeiro uP Intel a incluir a unidade de virgula flutuante no mesmo chip

Fraccionamento hardware/software

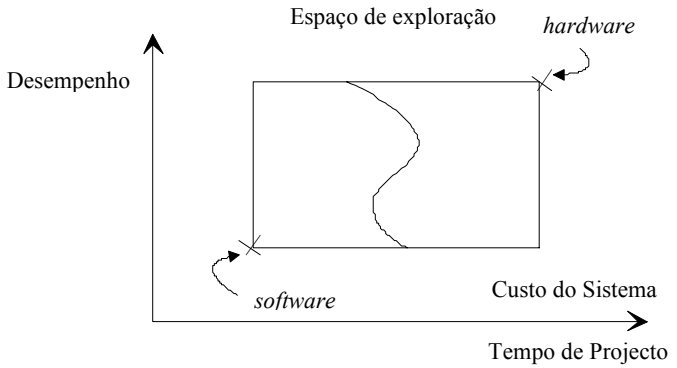
- Aceleração de processamento de imagens em tempo-real
  - Placas gráficas
  - Por exemplo: *scene rendering* feito em hardware

Relação Hardware/Software

- + Hardware
  - + velocidade (+ performance)
  - + custo
  - + risco
  - + tempo de projecto
  - - flexibilidade
- + Software
  - + software a desenvolver
  - + instruções de programa (> memória de programa)
  - + flexibilidade
  - - gastos no desenvolvimento
  - Pode implicar um uP diferente

Relação Hardware/Software

- Espaço de exploração do co-projecto



Relação Hardware/Software

- Hardware tem um ciclo de desenvolvimento mais lento
- Muitas das vezes o software está pronto para teste mas o hardware ainda não
  - Equipa do software tem de simular a funcionalidade do hardware para poder fazer os testes intermédios (utilização de *stubs*)

Hardware

- Se fosse fácil não lhe tinham chamado hard...!
- Desenvolvimento tecnológico permite colocá-lo quase lado a lado com o software
  - ASICs e FPGAs são cada vez mais utilizados nos projectos de sistemas embebidos
- Avanços no projecto e síntese de circuitos integrados

Hardware: projecto

- Compiladores de silício (anos 80)
  - Síntese de circuitos integrados
- Linguagens próximas do alto-nível das linguagens de software
  - Verilog, VHDL, etc.
  - Designam-se por HDL: *hardware description languages*

Hardware: projecto

- Uma classe em Java que implementa a funcionalidade de uma and

```
Class andgate {
  public boolean a, b, c;

  public void do() {
     c = a && b;
  }
}
```

- Um módulo em VHDL para implementar uma porta and

```
entity andgate is
   Port ( a : in bit;
          b : in bit;
          c : out bit);
end andgate;

architecture Behavioral of andgate is
begin
    c <= a AND b;
end Behavioral;
```

Hardware: projecto

- Java: Utilização da classe andgate

```
Class circuit {
  boolean result;

  Void define() {
     andgate and1 = new andgate();

    And1.a = true;
    And1.b = false;
    And1.do();
    this.result = and1.c;
  }
  ...
}
```

- VHDL: Utilização do componente andgate

```
…
architecture struct of circuit is
  component andgate
     Port ( a : in bit;
              b : in bit;
              c : out bit);
  end component;
  signal a, b, result: bit;
begin
    a <= '1';
    b <= '0';
    and1: andgate port map(a => a, b => b,
     result => c);
end struct;
```


Projecto de Hardware

- Começa a parecer-se com o projecto de software utilizando um compilador diferente...
- Utilização de bibliotecas fornecidas por empresas
  - Os API do software têm um equivalente no hardware
  - Cores de IP (*Intelectual Property*): código numa HDL que descreve um ou mais módulos

Co-Projecto Hardware/Software

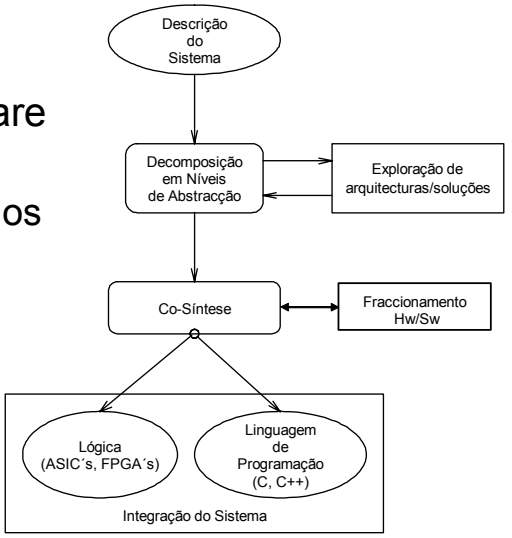
- Desenvolvimento do hardware e do software em conjunto (concorrentemente)
  - ao longo de todas as etapas o projecto mantém-se cooperativo
- A escolha para uma dada funcionalidade depende
  - dos requisitos do sistema global
    - desempenho, área de silício, flexibilidade, consumo de potência, e custo
- Metodologias anteriores
  - o fraccionamento era elaborado na fase inicial do projecto
  - projecto ramificado em dois sub-projectos distintos

Co-Projecto Hardware/Software

- O co-projecto é um projecto integrado com uma metodologia concorrente e cooperativa entre as análise de software e hardware, cujas principais vantagens são:
  - Implementações mais eficientes, e melhoramento do custo-eficiência;
  - Aumento do desempenho do sistema;
  - Melhoramento da fiabilidade;
  - Resposta mais adaptada aos requisitos.

Co-Projecto Hardware/Software

- Co-Síntese Hardware/Software
  - Projecto automatizado dos componente hardware e software



Co-Projecto Hardware/Software

- Co-Verificação Hardware/Software
  - Foca as correcções do interface hardware/software
- Co-Simulação Hardware/Software
  - Foca a simulação dos dois componentes em conjunto

Co-Projecto Hardware/Software

- Desvantagens do projecto do hardware sem cooperação com o projecto do software:
  - Ciclos de projecto independentes, com pouca interacção até à integração de todo o sistema:
    - hardware elaborado sem a apreciação completa dos requisitos do software
    - restrição da possibilidade de opção de resposta a requisitos de modo comum (hardware/software)
    - restrições na modificação do interface hardware/software
  - O software não influencia o projecto do hardware,
    - que não é modificado ao longo das opções necessárias no desenvolvimento do software;
  - O interface do sistema torna muitas vezes necessário modificar software e/ou hardware
    - custos mais elevados e perdas de desempenho
  - Selecção prematura do hardware, com posteriores correcções feitas em software

Co-Projecto Hardware/Software

- A maior parte das funcionalidades em Software, porquê?
  - Compiladores optimizados, com melhores resultados do que a programação directamente em assembler
  - Fiabilidade dos processadores,
    - por terem passado por fastidiosos processos de teste e de controlo de qualidade que asseguram a funcionalidade correcta, ao invés de integrados de aplicação específica
  - A depuração do software é extremamente simples, ao contrário da tarefa de depuração do hardware
  - O custo do sistema é menor
  - Facilidades de verificação e simulação
  - Maior flexibilidade no caso de modificação

Introdução aos Sistemas Embebidos

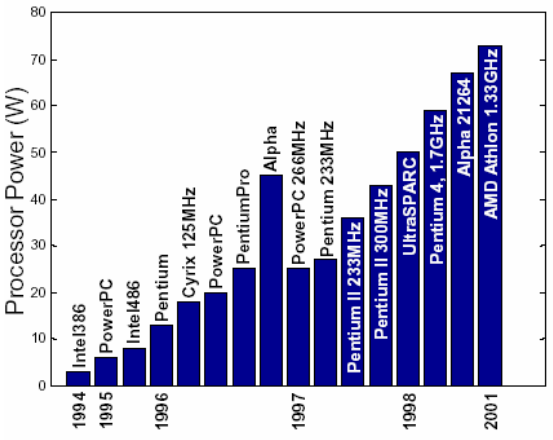
Consumo de Potência

CPU

- Performance
  - Tempo do ciclo de relógio, pipelining, sistema de memória (tópicos vistos em Arquitectura de Computadores)
- Consumo de potência
  - Depende do quê?

Consumo de Potência

- A maioria dos CPUs de hoje em dia são projectados tendo em mente o consumo de potência
- Potência vs. Energia:
  - Calor depende do consumo de potência
  - Tempo de descarregamento da bateria depende do consumo de energia



Consumo de Potência

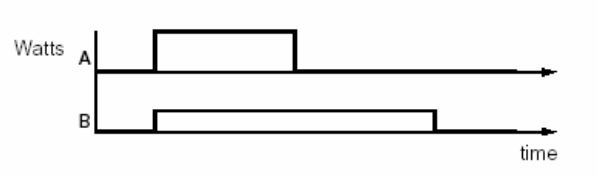
- Tecnologia dominante no fabrico de chips: CMOS

$$P_{tot} \approx P_{dynamic} + P_{static} + P_{short-circuit} \approx$$

$$\alpha C_L V_{dd}{}^2 f + V_{dd} I_{leak} + V_{dd} I_{sc}$$

Consumo de Potência

- Suponha-se dois CPUs (A e B):
  - em B uma tarefa demora mais tempo do que em A
  - Para a mesma tarefa o valor do pico da potência em A é maior do que em B
  - Qual dos CPUs é melhor em termos de consumo de energia?



Energia

- Integral da potência ao longo do tempo (produto da potência média consumida pelo tempo de computação)
  - A redução da potência só economiza energia se o tempo requerido para completar a tarefa não aumentar
  - Um CPU que consome mais potência que outro CPU pode ou não consumir mais energia para executar um determinado programa

Consumo de Potência: CMOS

- Redução da tensão de alimentação: consumo de potência proporcional a V^2
- *Toggling*: mais actividade implica mais consumo de potência.
- *Leakage*: características básicas do circuito; pode ser eliminada desligando o circuito da fonte de energia.

Estratégias para a redução do consumo de potência

- Redução da tensão de alimentação
- Execução a frequências de relógio mais baixas
- Desactivação de unidades funcionais quando não estão a ser utilizadas
- Desconexão de partes do chip da alimentação quando não estão a ser utilizadas

Estilos de gestão do consumo de potência

- Gestão estática de potência: não depende da actividade do CPU.
  - Exemplo: modo de *power-down* activado pelo utilizador
- Gestão dinâmica de potência: baseada na actividade do CPU
  - Exemplo: desactivação de unidades funcionais durante a execução de uma aplicação

Redução do consumo de potência

- Potência é poupada com modos de funcionamento controlados por software:
  - Por exemplo: modo sleep do PIC
- Os microprocessadores mais avançados têm vários modos cuja utilização pode depender da rapidez com que se quer voltar ao modo normal
- Existem técnicas avançadas para projecto de hardware de modo a reduzir o consumo de potência (disciplina de projecto de hardware)

Desactivação de unidades funcionais

- Benchmarks demonstram que, durante a execução de uma aplicação, muitas das unidades são utilizadas durante um curto período de tempo
- Percentagem do tempo que as unidades do PowerPC 603 estão **idle** para as benchmarks SPEC (fp e int):

unit	Specint92	Specfp92
D cache	29%	28%
I cache	29%	17%
load/store	35%	17%
fixed-point	38%	76%
floating-point	99%	30%
system register	89%	97%

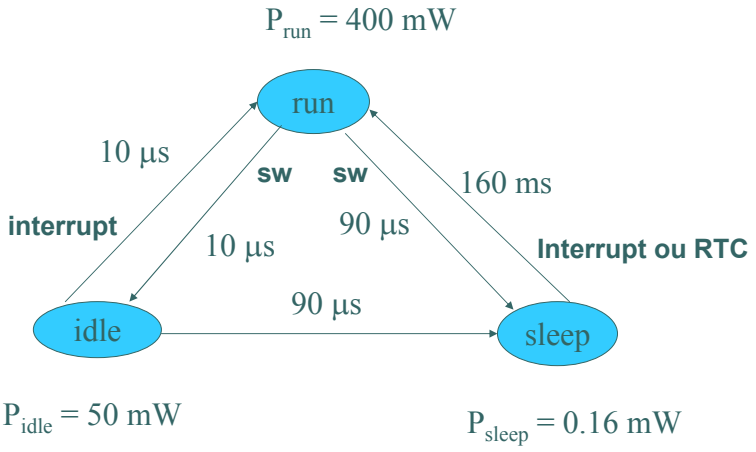
Power-Down: custos

- Passagem para o modo de *power-down* custa :
  - tempo
  - energia
- É necessário determinar se a passagem para este modo se justifica
- Podemos modelar os estados (modos) de potência de um CPU com uma máquina sequencial de estados finitos

Exemplo: Intel StrongARM SA-1100

- CPU trabalha a 1,5 V:
  - Têm sido realizados trabalhos que mostram gestão dinâmica da tensão de alimentação entre 0,8 V a 2 V
- Frequência pode ser modificada em tempo-real de 59 MHz a 206 MHz (cada modificação requer 150 us)
- Três modos de economia de potência:
  - Run: operação normal
  - Idle: pára o relógio do CPU, lógica continua alimentada
  - Sleep: desliga a maioria da actividade do chip; 3 passos, cada passo leva 30 us; o acordar demora > 10 ms.

Modos de economia de potência no CPU SA-1100

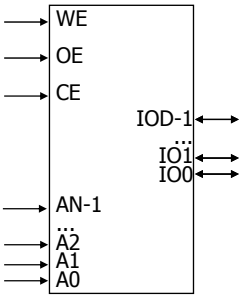


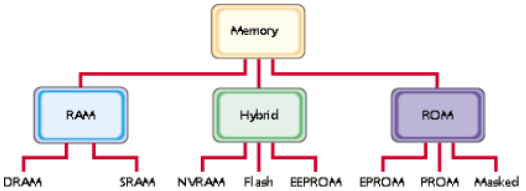
Introdução aos Sistemas Embebidos

Tecnologia de suporte ao CPU

Memórias

- RAMs (*random-access memory*)
  - DRAM, SRAM, DDRAM, etc.
- ROMs (*Read-Only Memory*)
  - EPROM, PROM, *Masked*
- Híbridas
  - EEPROM, Flash, NVRAM



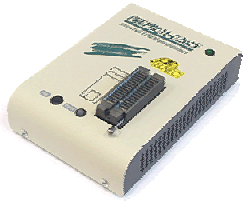


RAMs

- SRAM
  - *static random access memory*
- DRAM
  - *dynamic random access memory*
- DDRAM
  - *Double Data Rate-Synchronous* DRAM

ROMs

- PROM
  - *programmable* ROM
  - Só podem ser programadas uma vez (usa-se um dispositivo de programação de PROMS)
  - Também conhecidas por dispositivos *one-time programmable* (OTP)
- EPROM
  - *erasable-and-programmable* ROM
  - Podem ser programadas muitas vezes (exposição do silício à luz ultravioleta)



Híbridas

- EEPROM
  - *electrically-erasable-and-programmable* ROMs
  - escrita muito lenta
  - *Erasable* Byte a byte
- Flash
  - Grande densidade, custo baixo, não-volátil, mais rápida (na leitura, mas não na escrita), e
  - *electrically reprogrammable*
  - *Erasable* Sector a sector
  - Tamanho dos sectores típico: de 256 bytes a 16 kilobytes.
- Flash está a tornar-se a principal tecnologia de suporte em sistemas embebidos

Híbridas

- EEPROM versus Flash
  - Da perspectiva do software, as tecnologias Flash e EEPROM são similares
  - A maior diferença é que os dispositivos de Flash só podem ser escritos um sector de cada vez,
  - Tamanho dos sectores típico: de 256 bytes a 16 kilobytes
  - Aparte esta desvantagem, Flash é muito mais popular do que EEPROM e
  - substitui cada vez mais os dispositivos ROM nos sistemas embebidos

Híbridas

- NVRAM: *nonvolatile* RAM
  - tecnologia muito diferente de Flash e EEPROM
  - é uma SRAM com um *backup* de bateria
  - Quando a alimentação está ligada a NVRAM opera como uma SRAM
  - Quando a alimentação está desligada a NVRAM utiliza a energia da bateria necessária para manter o conteúdo
  - Tecnologia mais cara (mais do que SRAMs)
  - Tipicamente limitadas a armazenar algumas centenas de bytes

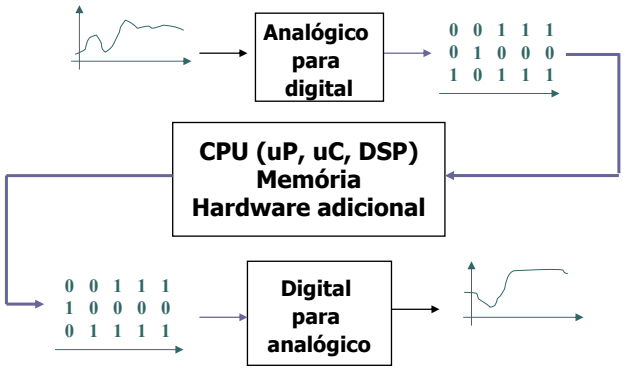
Memórias

- Características

Memory Type	Volatile?	Writeable?	Erase Size	Erase Cycles	Relative Cost	Relative Speed
SRAM	yes	yes	byte	unlimited	expensive	fast
DRAM	yes	yes	byte	unlimited	moderate	moderate
Masked ROM	no	no	n/a	n/a	inexpensive	fast
PROM	no	once, with programmer	n/a	n/a	moderate	fast
EPROM	no	yes, with programmer	entire chip	limited (see specs)	moderate	fast
EEPROM	no	yes	byte	limited (see specs)	expensive	fast to read, slow to write
Flash	no	yes	sector	limited (see specs)	moderate	fast to read, slow to write
NVRAM	no	yes	byte	none	expensive	fast

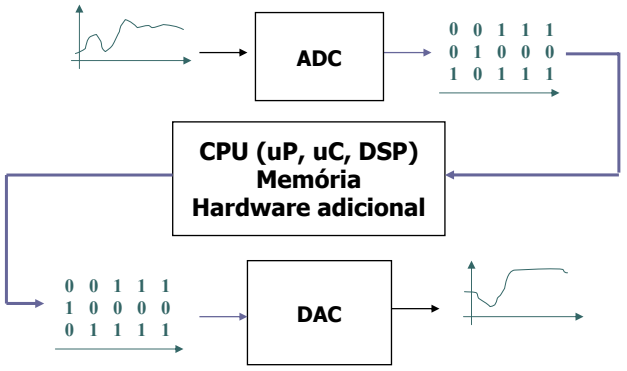
Sistema Embebidos

- Analógico-Digitais



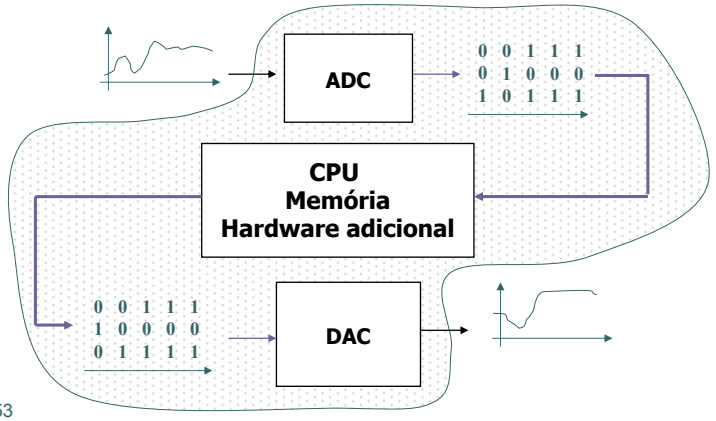
Sistema Embebidos

- Analógico-Digitais



Sistema Embebidos

- Analógico-Digitais



Sistemas Embebidos

- Interfaces Analógico-Digitais
  - ADC: *Analog to Digital Converter*
    - Requer amostragem da entrada analógica antes da conversão para digital
    - Um sinal de controlo indica ao dispositivo para digitalizar amostra
    - Nos dispositivos em que o tempo da digitalização depende da amostra, existe um sinal que indica o fim da digitalização
  - DAC: *Digital to Analog Converter*
    - Converte continuamente a palavra digital na entrada em forma analógica

Hardware Específico

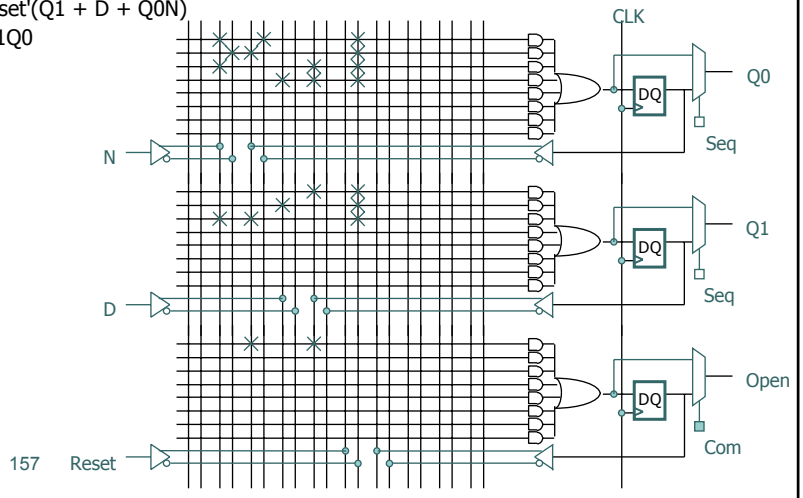
- Hardware não-programável
  - Algumas funções podem ser implementadas numa ROM
  - ASIC (*Application Specific Integrated Circuit*)
  - Chip construído especialmente para realizar uma determinada tarefa do sistema
- Hardware programável
  - PLAs, PALs, PLDs
  - FPGAs

Hardware Programável

- PLA: *Programmable Logic Array*
- PAL: *Programmable Array Logic*
  - Mais económicas e mais rápidas do que as PLAs
- PLD: *programmable logic device*
- FPGA: *Field-Programmable Gate Array*
  - Maior potencialidade do que os PLD

PLDs

D0 = reset'(Q0'N + Q0N' + Q1N + Q1D)
D1 = reset'(Q1 + D + Q0N)
OPEN = Q1Q0



Dispositivos de Hardware Programável

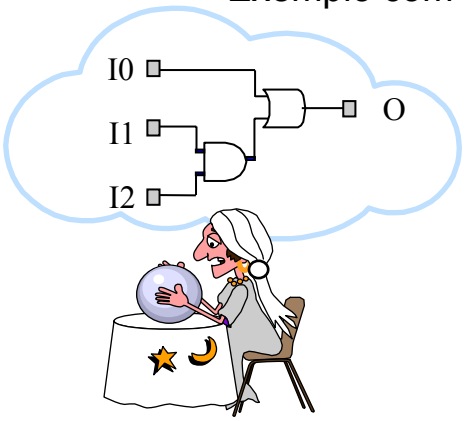
- Elemento de computação (célula)
  - Função lógica de N entradas (FPGA)
  - Operações aritméticas ou lógicas (dispositivos de granulosidade maior)
- Interligações
  - Fios que permitem ligações entre saídas/entradas de células ou de pinos de I/O (FPGA)
  - Barramentos (dispositivos de granulosidade maior)

FPGAs

- Matrizes de células e recursos para interligações
  - E muito mais...
- Cada célula pode ser programada para desempenhar uma função lógica de N entradas (complexidade da função depende da complexidade da célula)
- Ligações entre interligações são também programados
- A programação é normalmente feita por escrita em células de SRAM

FPGAs

- Como realizar a função lógica?
  - Exemplo com 3 entradas e uma saída

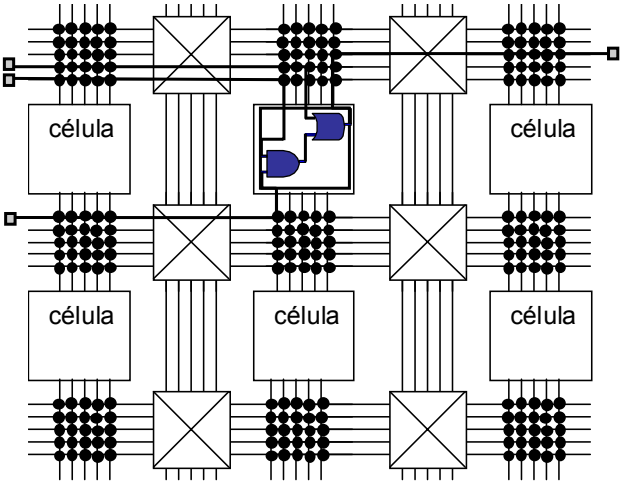


I2	I1	I0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3 entradas: 256 funções possíveis

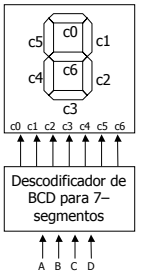
2^{2^N}

FPGAs



Outros dispositivos

- LEDs: *Light-emitting diodes*
  - Utilizados para sinalizar (ascende/apaga): eventos, valores lógicos, etc.
  - Podem ser usados na depuração
  - Podem formar *displays*
- *LCD: liquid crystal display*
  - Utilizados para mostrar símbolos (e.g., *display* de 7 segmentos para algarismos)
- Ecrãs sensíveis ao toque
- Teclados



Links

- Michael Barr's Embedded Systems Glossary
  - http://www.netrino.com/Publications/Glossary/index.html

Introdução aos Sistemas Embebidos

Teste dos dispositivos de Memória do Sistema

Do livro:

Michael Barr, Programming Embedded Systems in C and C++, O'Reilly & Associates, Inc., 1999.

Teste de Memórias

- Objectivo: confirmar que cada endereço de memória está a funcionar
- Se armazenarmos o número 50 num determinado endereço esperamos que o número esteja lá até que haja outra escrita no mesmo endereço
- A ideia básica no teste de memórias é escrever um determinado conjunto de valores em cada endereço e verificar os valores armazenados lendo-os de volta
- Se os valores lidos são os mesmos que os escritos então a posição de memória passa no teste
- A selecção do valor a escrever é importante...

Falhas nas Memórias

- Erros no chip
  - Muito difíceis de ocorrer - o fabricante realizou testes bastante completos e é muito improvável que o chip novo tenha defeitos
- Danos devido a catástrofe (eléctrica ou física)
  - Torna defeituosas zonas grandes do chip e por isso bastante fácil de descobrir pelo programa de teste
- Defeitos relacionados com a Placa
  - Mais prováveis de acontecer

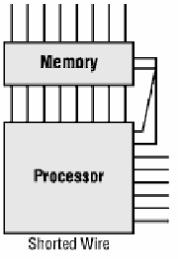
Defeitos relacionados com a Placa

- Problemas eléctricos nas ligações (fios) entre a memória e os dispositivos
- Dispositivo em falta (não se encontra presente)
- Má conexão do dispositivo no *socket*

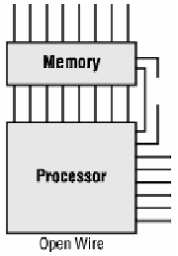
- Um bom programa de teste deve conseguir detectar estes problemas

Problemas eléctricos nas ligações (fios) entre a memória e os dispositivos

- Problema num fio pode ser causado por um erro no projecto ou na produção da placa ou resultante de um acidente depois da manufactura
- Cada fio que liga o processador à memória é de um dos 3 tipos:
  - Linha de endereços
  - Linha de dados
  - Linha de controlo (indicam leitura ou escrita)
- Um ou mais fios podem ser ligados incorrectamente ou pode ter sido danificado de um modo que:
  - Curto-circuito (ligado a outro fio da placa)
  - Em aberto (desligado)



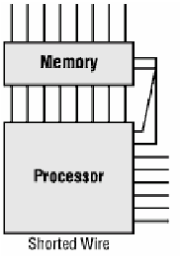
Shorted Wire

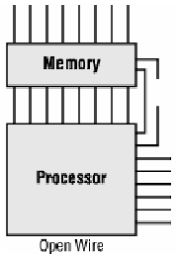


Open Wire

Problemas eléctricos nas ligações (fios) entre a memória e os dispositivos

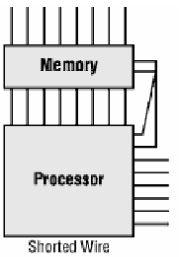
- Causam funcionamento incorrecto da memória:
  - Dados podem ser armazenados incorrectamente
  - Guardados no endereço incorrecto
  - Ou não serem guardados
- Problema com linhas de dados
  - Vários bits podem estar ligados, dois ou mais bits contêm sempre o mesmo valor, independentemente do valor transmitido
  - Uma linha pode estar sempre ligada a “1” ou ligada a “0”
  - Estes problemas podem ser detectados escrevendo uma sequência de dados que testam se a cada pino pode ser atribuído o valor zero e o valor um, independentemente do valor dos outros bits

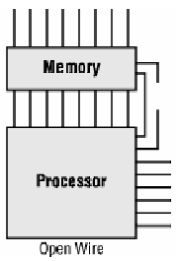




Problemas eléctricos nas ligações (fios) entre a memória e os dispositivos

- Problemas nas linhas de endereços
  - O conteúdo de duas posições de memória podem parecer sobrepostas (dados escritos num endereço alteram o valor de outro endereço)
- Linha de controlo curto-circuitada ou em aberto
  - Normalmente os sinais de controlo são específicos ao tipo de memória utilizada
  - Se existir um problema é normal que este seja detectado com os testes a linhas de endereços e dados





Dispositivo em falta

- Um dispositivo ausente deve ser detectado
- Por causa da natureza capacitiva dos fios desligados deve-se ter certos cuidados
  - Não se deve escrever e em seguida ler pois o valor pode não ser da posição de memória mas da tensão remanescente (acontece quando o processo de escrita-leitura é muito rápido)
- Normalmente é feito um teste de escritas de dados em posições consecutivas seguido de leituras consecutivas
- Por exemplo, escrever o valor 1 na primeira posição, 2 na segunda, 3 na terceira, e em seguida verificar o conteúdo da 1ª posição, 2ª posição, etc.
- Se os dados são únicos (como no exemplo), a ausência de um dispositivo será detectada

Má conexão do dispositivo no *socket*

- Se uma memória está presente mas inapropriadamente ligada ao *socket*, o sistema comporta-se normalmente como se a memória estivesse ausente ou como se houvesse más ligações
- Se o teste de ligações e de chips ausentes for feito a má conexão será detectada

Teste de Memórias

- Estratégia de teste
  - A escolha criteriosa dos dados para teste e a ordem pela qual os endereços são testados, é possível detectar todos os problemas referidos
- É normalmente aconselhável dividir o teste de memórias em partes
  - Ajuda a melhorar a eficiência do teste global
  - Torna mais legível o código
  - Podem fornecer informação mais detalhada acerca da origem do problema que tiver sido detectado
- Por exemplo, três testes individuais pela ordem indicada:
  - Teste do barramento de dados (fios e chips ligados impropriamente)
  - Teste do barramento de endereços (fios e chips ligados impropriamente)
  - Teste do dispositivo (chips ausentes e falhas catastróficas)

Teste de Memórias

- Como consequência, o teste do dispositivo também detectará problemas nas linhas de controlo, embora não se possa extrair a origem do problema
- A ordem de execução destes testes é importante:
  - por que o teste do barramento de endereços assume um barramento de dados a funcionar, e o teste do dispositivo só faz sentido se os barramentos estiverem a funcionar correctamente

Teste do Barramento de Dados

- Confirmar que o valor colocado no barramento de dados é recebido correctamente pela memória
- O meio mais óbvio é escrever todos os valores possíveis e verificar se a memória armazena cada um deles com sucesso
- Um método mais rápido será testar o barramento um bit de cada vez
  - O barramento passa o teste se cada bit puder ser colocado a zero e a um independentemente dos outros bits
  - Cada bit pode ser testado independentemente com um teste chamado de "walking 1's test"

Teste do Barramento de Dados

- Padrões de dados para este teste supondo um barramento de 8 bits
  - 00000001
  - 00000010
  - 00000100
  - 00001000
  - 00010000
  - 00100000
  - 01000000
  - 10000000
- O número de valores a testar é o mesmo do que o comprimento do barramento
- Reduz o número de padrões de teste de 2^N para N (N é o número de bits do barramento)

Teste do Barramento de Dados

- Como estamos a testar o barramento de dados, todos os valores podem ser escritos no mesmo endereço
- Caso o barramento de dados seja ligado a várias memórias o teste dever ser feito para cada uma delas (repetido com o endereço específico a cada chip)
- Para realizar o teste
  - escreve-se um padrão e
  - de seguida lê-se e verifica-se se o valor lido é o mesmo

Teste do Barramento de Dados

- A função *memTestDataBus* mostra como implementar o “walking 1's test” em C
  - Assume que quem invoca a função selecciona o endereço de teste e testa todo o conjunto de dados utilizando esse endereço
- Se o barramento estiver a funcionar correctamente a função retorna 0, caso contrário retorna o primeiro valor para o qual o teste falhou
- O bit a um no valor devolvido corresponde à primeira linha com falta

Teste do Barramento de Dados

```
typedef unsigned char datum; /* Set the data bus width to 8 bits. */

/* memTestDataBus() */
Datum memTestDataBus(volatile datum * address) {
    datum pattern;
    /* Perform a walking 1's test at the given address. */
    for (pattern = 1; pattern != 0; pattern <<= 1) {
        /*  Write the test pattern. */
        *address = pattern;
        /* Read it back (immediately is okay for this test). */
        if (*address != pattern) {
            return (pattern);
        }
    }
    return (0);
} /* memTestDataBus() */
```


Introdução aos Sistemas Embebidos

Teste dos dispositivos de Memória do Sistema (continuação)

Do livro:

Michael Barr, Programming Embedded Systems in C and C++, O'Reilly & Associates, Inc., 1999.

Teste do Barramento de Endereços

- Problemas no barramento de endereços originam posições de memória sobrepostas
- Verificar se cada pino de endereço pode ser colocado a um e a zero sem afectar os outros
- Pode ser utilizado um esquema similar ao “walking 1's test”
  - 00000h, 00001h, 00002h, 00004h, 00008h, 00010h, 00020h, etc.
- Como pode haver posições de memória sobrepostas,
  - depois da escrita num dos endereços tem de se verificar que nenhum dos outros endereços foi escrito

Teste do Barramento de Endereços

- Nem todas as linhas de endereços podem ser testadas desta forma!
  - Parte do endereço (bits mais à esquerda) seleccionam o chip de memória
  - A outra parte (bits mais à direita) podem não ser todos significativos
  - Os bits que não são significativos permanecem constantes durante o teste e por isso permitem reduzir o número de endereços a testar
- Por exemplo,
  - se o processador tem 20 bits de endereço pode endereçar 1 MB de memória
  - Teste de um bloco de memória de 128 KB => que os 3 bits mais significativos permaneçam constantes (neste caso apenas os 17 bits mais à direita são testados)

Teste do Barramento de Endereços

- Para confirmar que duas posições de memória não estão sobrepostas
  - Devemos escrever um valor em cada offset potência de dois
  - Depois escrevemos um novo valor (uma cópia invertida do valor anterior é uma boa escolha) no primeiro offset do teste
  - Em seguida verificamos os outros offsets potência de dois e se encontramos uma posição, que não aquela em que escrevemos o novo valor, com esse novo valor então existe um problema nessa linha de endereçamento
  - Caso nenhuma sobreposição tenha sido encontrada repetimos o procedimento para cada um dos outros offsets

Teste do Barramento de Endereços

- A função *memTestAddressBus indica* como o teste pode ser feito
  - Aceita dois parâmetros: o endereço base do bloco de memória, o tamanho do bloco em bytes
- O tamanho do bloco é necessário para determinar quais os bits do barramento de endereços devem ser testados
- Se o teste falhar, é retornado o endereço onde foi encontrado o primeiro erro
- Caso contrário a função retorna NULL (indica sucesso)

Teste do Barramento de Endereços

```
datum * memTestAddressBus(volatile datum * baseAddress, unsigned
    long nBytes) {
      unsigned long addressMask = (nBytes - 1);
      unsigned long offset, testOffset;
      datum pattern = (datum) 0xAAAAAAAA;
      datum antipattern = (datum) 0x55555555;
      /* Write the default pattern at each of the power-of-two offsets.. */
      for (offset = sizeof(datum); (offset & addressMask) != 0; offset <<= 1)
            baseAddress[offset] = pattern;

      Código do slide seguinte

      return (NULL);
} /* memTestAddressBus() */
```


Teste do Barramento de Endereços

```
/* Check for address bits stuck high. */
testOffset = 0;
baseAddress[testOffset] = antipattern;
for (offset = sizeof(datum); (offset & addressMask) != 0; offset <<= 1)
      if (baseAddress[offset] != pattern)
            return ((datum *) &baseAddress[offset]);

baseAddress[testOffset] = pattern;

/* Check for address bits stuck low or shorted. */
for (testOffset = sizeof(datum); (testOffset & addressMask) != 0; testOffset <<= 1) {
    baseAddress[testOffset] = antipattern;
    for (offset = sizeof(datum); (offset & addressMask) != 0; offset <<= 1)
          if ((baseAddress[offset] != pattern) && (offset != testOffset))
                return ((datum *) &baseAddress[testOffset]);
    baseAddress[testOffset] = pattern;
}
```

Teste do Dispositivo

- Cada bit do dispositivo é capaz de armazenar zero e também um
  - Computacionalmente, muito mais intensivo do que os testes anteriores
  - Temos de visitar (escrever, ler) cada posição de memória duas vezes
- É preferível utilizar valores diferentes para cada endereço
- É normalmente utilizado um teste incremental

Teste do Dispositivo

- O padrão de teste incremental é fácil de implementar

Memory Offset	Binary Value	Inverted Value
000h	00000001	11111110
001h	00000010	11111101
002h	00000011	11111100
003h	00000100	11111011
... ...		
0FEh	11111111	00000000
0FFh	00000000	11111111

Teste do Dispositivo

- A função *memTestDevice* implementa as duas passagens do teste de incremento/decremento
- Aceita dois parâmetros
  - Endereço de início
  - Número de bytes a testar
- A função retorna NULL em caso de sucesso
- Caso contrário, retorna o primeiro endereço que contém um valor incorrecto

Teste do Dispositivo

```
datum * memTestDevice(volatile datum * baseAddress, unsigned
    long nBytes) {
      unsigned long offset;
      unsigned long nWords = nBytes / sizeof(datum);
      datum pattern;
      datum antipattern;

      Código do slide seguinte

      return (NULL);
} /* memTestDevice() */
```

Teste do Dispositivo

```
/* Fill memory with a known pattern. */
for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++) {
      baseAddress[offset] = pattern;
      /* Check each location and invert it for the second pass. */
      for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++) {
         if (baseAddress[offset] != pattern)
                return ((datum *) &baseAddress[offset]);
         antipattern = ~pattern;
         baseAddress[offset] = antipattern;
      }
}
/* Check each location for the inverted pattern and zero it. */
for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++) {
      antipattern = ~pattern;
         if (baseAddress[offset] != antipattern)
                return ((datum *) &baseAddress[offset]);
      baseAddress[offset] = 0;
}
```


Estratégia de Teste

- Necessidade de teste de memórias
  - Perceptível durante o desenvolvimento do produto
  - Como as memórias são um dos recursos críticos de um sistema embebido, pode ser importante incluir o teste na versão final do software
- Neste caso, o teste a memórias (e todos os outros testes a efectuar) deve ser realizado de cada vez que o sistema é ligado ou de cada vez que é feito o reset

Validação do Conteúdo de Memórias

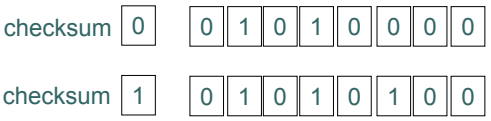
- Alguns dos testes ilustrados não fazem sentido com memórias ROM ou híbridas
  - As ROMs não podem ser escritas!
  - Os dispositivos híbridos contêm dados que não podem ser apagados!
- Contudo, o mesmo tipo de erros pode ocorrer com estes dispositivos!
- Tem de haver um método de determinar se há algum problema com estes dispositivos
- Vamos falar do esquema mais simples:
  - Checksums
- Um esquema mais complexo e do qual não falaremos é o:
  - Cyclic redundancy code (CRC)

Checksums

- Como determinar se os dados ou se o programa armazenados numa memória não-volátil são válidos?
  - O método mais fácil é determinar o checksum dos dados quando se conhece o valor correcto (e.g., antes de programar a ROM)
- Cada vez que se quer confirmar a validade dos dados só é necessário recalcular o checksum e comparar o resultado com o valor do checksum correcto
- Se os dois forem iguais os dados são válidos
- Se escolhermos um bom algoritmo de checksum, a probabilidade de detecção de dados inválidos é maior

Checksums

- Um dos algoritmos mais simples de checksum é somarmos todos os bits de uma palavra não considerando os bits de transporte (o checksum indica se o número de 1's é ímpar)
- Um dos defeitos deste algoritmo é que se por acidente todos os bits de uma palavra forem colocados a zero (incluindo o checksum) não se detecta qualquer erro!
- O método mais simples de resolver esta fraqueza é adicionarmos um passo final ao algoritmo: inverter o checksum



Checksums

- Infelizmente, este esquema não consegue detectar muitos dos erros que podem acontecer
  - Se apenas um bit de uma palavra for corrompido (de 0 para 1 ou de 1 para 0) o esquema detecta
  - Mas se dois bits forem corrompidos e mudarem de valor o erro é detectado?

Checksums

- Depois de calcularmos o checksum de cada palavra onde devemos armazená-lo?
  - Na rotina que verifica a validade dos dados?
  - Na memória?
- A escolha da rotina tem a desvantagem de que sempre que é modificado o conteúdo da memória também tem de se modificar a rotina
- Um esquema vai vantajoso é a colocação dos checksum na mesma memória onde estão os dados a validar ou noutra memória não-volátil do sistema

Introdução aos Sistemas Embebidos

Modelação de Sistemas

Modelo

- Sistema formal constituído por objectos e regras de composição, que é usado para descrever as características de um sistema
- Propósito: forma abstracta de representar um sistema
- Uma linguagem pode capturar muitos modelos diferentes, e um modelo pode ser capturado por muitas linguagens diferentes

Modelo

- Formal, de modo a não ter ambiguidades
- Completo, de modo a descrever todo o sistema
- Compreensível, de modo a que os desenvolvedores o percebam
- Fácil de modificar
- Suficientemente natural para que seja fácil modelar o sistema

Modelo: exemplo do elevador

- Vistas conceptuais do controlador do elevador possíveis
  - Funcionalidade pretendida em linguagem natural
  - Modelo algorítmico
  - Modelo de máquina de estados

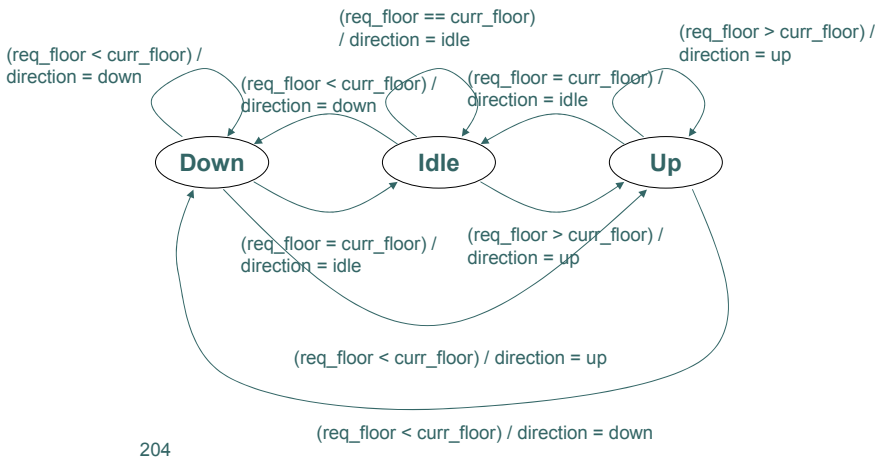
Elevador: linguagem natural

- Se o elevador estiver parado e o piso chamado for o piso actual do elevador, então o elevador mantém-se no mesmo piso
- Se o elevador estiver parado e o piso chamado for menor do que o piso actual do elevador, então o elevador desce para o piso chamado
- Se o elevador estiver parado e o piso chamado for maior do que o piso actual do elevador, então o elevador sobe para o piso chamado

Elevador: algoritmo

```
While(true) {
    If(req_floor == curr_floor)
        Direction = idle;
    Else if(req_floor < curr_floor)
        Direction = down;
    Else if(req_floor > curr_floor)
        Direction = up;
}
```


Elevador: máquina de estados



Modelos

- Vários modelos do sistema
  - => diferentes perspectivas do sistema (modelação de diferentes características)
  - => diferentes vistas conceptuais
- Vantagens da utilização de vários modelos
  - Modelo de máquina de estados: comportamento temporal do sistema (modos e a transição de modos causada por eventos internos ou externos)
  - Modelo algorítmico: Vista procedimental do sistema (relação entre as entradas e as saídas do sistema)

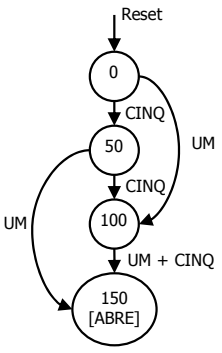
Modelos

- O tipo de modelos a utilizar depende
  - do tipo de sistema a implementar
  - do nível de abstracção

Modelos

- Orientados para os estados
  - FSM: máquina de estados finitos
  - FSMD: máquina de estados finitos com caminho de dados
  - E muitos outros:
    - HCFSM: máquina de estados finitos concorrentes e hierárquicas
    - Statecharts
    - Redes de Petri
    - ...

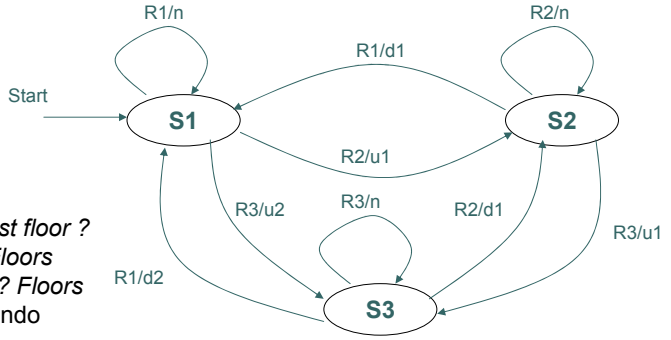
FSM: máquinas de estados finitos



- Conjunto de estados representados por círculos
- Conjunto de transições entre estados
- Transições dependem de eventos
- Saídas dependem apenas do estado actual, ou do estado actual e das entradas
- Estado inicial (onde é iniciada a execução da máquina)
- Representadas graficamente por diagramas de transições de estados (STG - *State Transition Graphs*)

FSM: máquinas de estados finitos

- Controlador do elevador para 3 pisos



R? Indica: *request floor ?*
u? Indica: *up ? Floors*
d? Indica: *down ? Floors*
n: nenhum comando

FSM: máquinas de estados finitos

- De Mealy
  - As saídas dependem das entradas e do estado actual
- De Moore
  - As saídas dependem do estado actual
- A máquina anterior é de Moore ou de Mealy?
  - Desenhe uma máquina de tipo diferente ao da máquina apresentada mas que seja equivalente

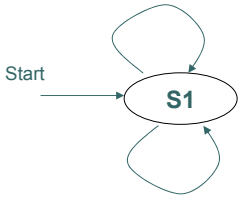
FSM: máquinas de estados finitos

- Explosão do número de estados!
  - Tente modelar um somador de duas palavras representadas por 16 bits utilizando uma FSM!
  - Tente modelar um contador de 0 a 255 utilizando uma FSM!
- Forma de resolução da explosão do número de estados:
  - Utilização de variáveis e expressões

Máquinas de estados finitos

- FSMD: *FSM with datapath*
- É uma FSM com enunciados de atribuição em cada estado ou em cada transição

(curr_floor != req_floor) / output = req_floor – curr_floor; curr_floor = req_floor;



(curr_floor == req_floor) / output = 0;

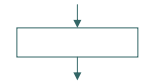
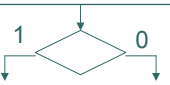
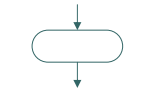
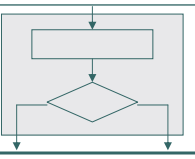
Máquinas de estados finitos

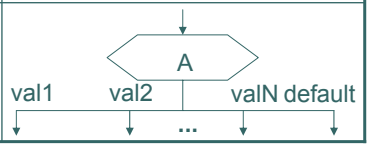
- FSM: adequada para modelar sistemas dominantemente de controlo
- FSMD: pode ser adequada para sistemas dominantemente de controlo e de dados
- Nenhum é adequado para sistemas complexos:
  - Não suportam concorrência
  - Não suportam hierarquia
  - Quando atingem centenas de estados ou de arcos tornam-se incompreensíveis para os humanos!

Máquinas de estados finitos

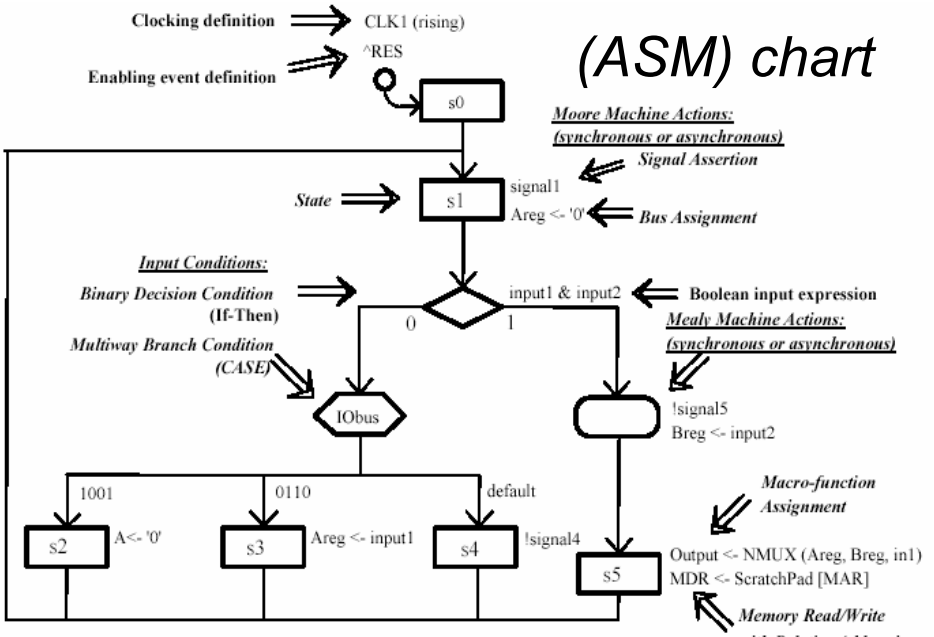
- *Algorithmic-state machine (ASM) chart*
  - representação gráfica utilizada para representar FSMDs

Algorithmic-state machine (ASM) chart

Nome	Definição
Caixa de estado (incondicional)	
Caixa de decisão	
Caixa condicional (atribuição condicional)	
Bloco ASM	

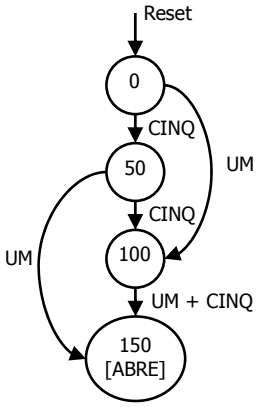
Nome	Definição
Caixa de decisão tipo switch (case)	

(ASM) chart



Fonte: http://www.cse.sc.edu/~jimdavis/Courses/CSCE-611/CSCE611(Fall02)-Lectures8&9.pdf

Máquinas de estados finitos



- Desenhe uma FSMD para a máquina de estados que controla a máquina de chocolates apresentada nas aulas práticas
  - Utilizando um *ASM chart*

Introdução aos Sistemas Embebidos

Modelação de Sistemas (continuação)

Modelos

- Orientados para a actividade
  - Grafo de fluxo de dados (*dataflow graphs*)
    - DFG
  - Grafos de fluxo de dados e de controlo
    - CDFGs (*control/data flow graphs*)
    - Parecido com os fluxogramas

DFG

- Modelo sem condicionais
- Nós circulares representam actividade (operações, por exemplo)
- Nós rectangulares representam entrada/saída de dados (variáveis de entrada ou variáveis intermédias, por exemplo)
- Laços direccionados representam fluxo de dados entre actividades
- São por natureza acíclicos
- Podem ter hierarquia (um nó de actividade pode ser representado por outro DFG)

DFG

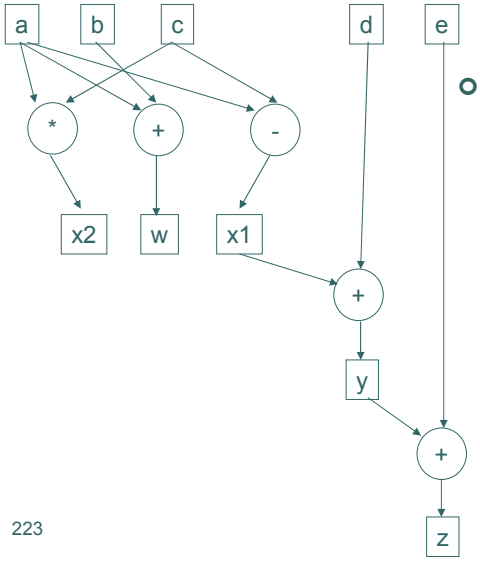
- Muito útil e pode ser utilizado em diferentes domínios
  - Basta utilizar associações diferentes aos laços e aos nós do DFG
  - Por exemplo, na programação pode ser utilizado para representar enunciados de instruções sem saltos (nós circulares representam operações, nós rectangulares representam variáveis, e os laços dependências de dados)

DFG

- Atribuição única: uma variável aparece uma só vez do lado esquerdo da atribuição
- Bloco básico em C:
- W=a+b;
- X=a-c;
- Y=x+d;
- X=a+c;
- Z=y+e;

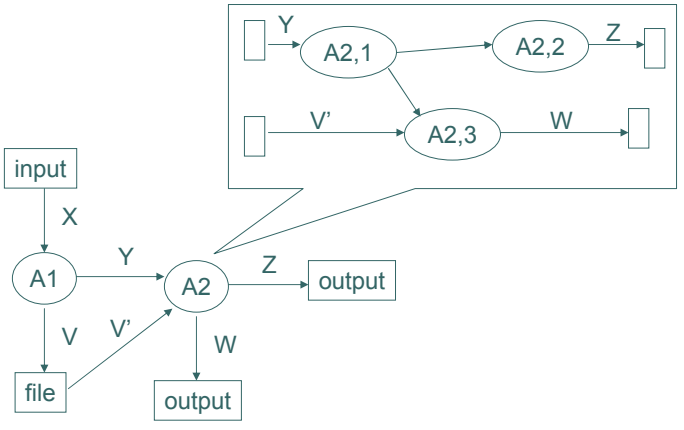
- Bloco básico na forma de atribuição única
- W=a+b;
- X1=a-c;
- Y=x1+d;
- X2=a+c;
- Z=y+e;

DFG



- Muitas das vezes os rectângulos não são usados e usam-se rótulos a indicar o nome das variáveis

DFG



DFG

- Na implementação software correspondente ao DFG, as operações têm de ser ordenadas
- No DFG as operações são apenas ordenadas pelas dependências de dados
- O DFG é útil para realizar algumas optimizações e para se ordenar eficientemente as operações (tendo em conta o *pipelining* do uP, por exemplo)

DFG

- É um modelo muito utilizado durante a fase de especificação
- Mas como não modela acções de controlo e comportamento temporal…
- É pouco utilizado para modelar sistemas embebidos

CDFG

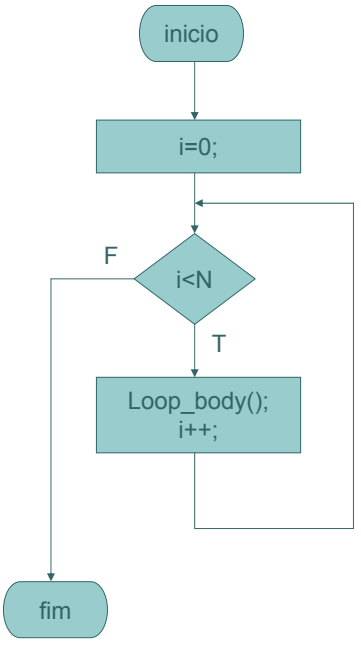
- Inclui construções para modelar
  - operações sobre dados (aritméticas e outras)
  - Operações de controlo (condicionais)
- Tipos de nós
  - Nó de início e nó de fim
  - Nós de decisão
  - Nós de fluxo de dados: podem encapsular um DFG (neste caso o CDFG é hierárquico)

CDFG

```
For(i=0; i<N; i++)
   loop_body();
```

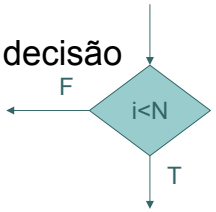
- Equivale a:

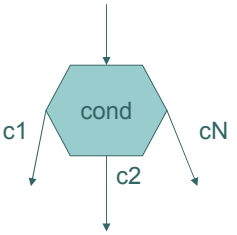
```
i=0;
While(i < N) {
   loop_body();
   i++;
}
```



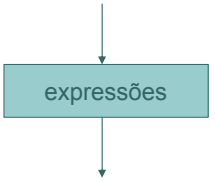
CDFG

- Nós de decisão





- Nós de fluxo de dados



CDFG

- Desenhar o CDFG para os dois exemplos seguintes:

Exemplo 1:

```
While(a < b) {
    a= proc1(a, b);
    b=proc2(a, b);
}
```

Exemplo 2:

```
If(cond1)
  basic_block_1();
Else
  basic_block_2();
Switch(test1) {
  case c1:
   basic_block_4(); break;
  case c2:
   basic_block_5(); break;
  case c3:
   basic_block_6(); break;
}
```

CDFG

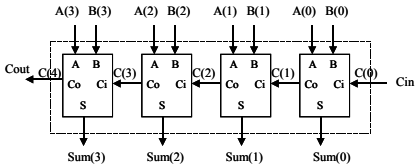
- O CDFG modela uma FSMD?
  - Também dá enfâse aos aspectos relacionados com o controlo do sistema, mas…
  - A diferença reside em como as transições entre nós são despoletadas
    - No CDFG: as transições são efectuadas assim que uma dada actividade esteja concluída
    - Na FSMD: as transições são efectuadas por ocorrência de eventos externos

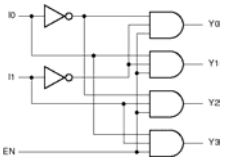
CDFG

- Uma representação gráfica similar é o fluxograma (*flowcharts*)
  - Muito utilizado, embora os puristas de modelação de sistemas digam “cobras e lagartos” do modelo
  - Nos fluxogramas os rectângulos representam acções (computações), mas existem outros tipos de nós para representar acessos a ficheiros, por exemplo

Modelos

- Orientados para a estrutura
  - Diagramas de conectividade entre componentes (CCD: *component-connectivity diagrams*)
  - Representam a estrutura de um sistema: podem ser utilizados em vários níveis de abstracção (portas lógicas, unidades funcionais, dispositivos da placa, etc.)





+ Modelos (não abordaremos)

- Existem muitos mais modelos, alguns para determinados domínios, outros que permitem aplicações mais vastas
- Orientados para os dados
  - Diagramas entidade-associação
  - Diagramas de Jackson
  - …
- Heterogéneos
  - Modelo orientado a objectos
  - PSMs (*Program-state machines*)
  - …

Introdução aos Sistemas Embebidos

Implementações em Software

Implementação

- Dado um modelo segue-se a implementação
  - Em software
    - Caso se utilize uma linguagem de alto-nível, a tradução é quase directa
    - Caso se utilize assembly, a tradução é mais fastidiosa, mas podem conseguir-se melhores resultados (tamanho de código, tempo de execução, etc.)
  - Em hardware (fora do âmbito desta disciplina)

Programa principal

- A maioria das vezes é constituído por um ciclo infinito:

```
Void main(void){
    // inicializar
    while (1) {
            /* Change the state of the LED.*/
           toggleLed(LED_GREEN);
            /* Pause for 500 milliseconds.*/
           delay(500);
           /* Clear WDT */
           clear_wdt();
    }
} /* main() */
```


Tempos de espera

- Rotina sintonizada por medições ou simulações

```
Void delay(unsigned int nMilliseconds) {
    /* Number of decrement-and-test cycles.*/
    #define CYCLES_PER_MS 260
    unsigned long nCycles = nMilliseconds *
    CYCLES_PER_MS;
    while (nCycles--);
} /* delay() */
```

Temporizadores

- Utilização de uma biblioteca para o microprocessador ou microcontrolador
- Compilador reconhece enunciados como sendo utilização do temporizador

```
Void main(void) {
    Timer timer; // Timer is defined in library
    timer.start(500, Periodic); // Start a periodic 500 ms Timer.
    while (1) {
        toggleLed(LED_GREEN); // Toggle the green LED.

        // Do other useful work here.

        timer.waitfor(); // Wait for the timer to expire.
    }
} /* main() */
```


Watch Dog Timers (WDTs)

- Temporizador especial que quando termina a contagem faz com que o uP ou uC salte para o vector de reset
- Em termos de software é muitas das vezes necessário reiniciar periodicamente a contagem destes temporizadores
- Caso contrário, o tempo de término do WDT pode ocorrer durante a execução de uma rotina!

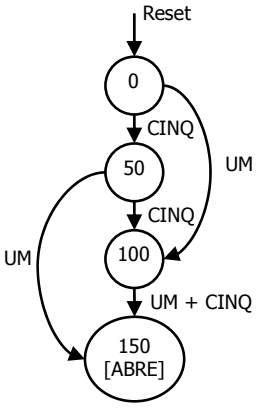
```
while (1) {
        // do useful work here!
        /* Clear WDT */
        clear_wdt();
}
```

Implementação em Software de FSMs e FSMDs

- Uso da construção alto-nível Switch

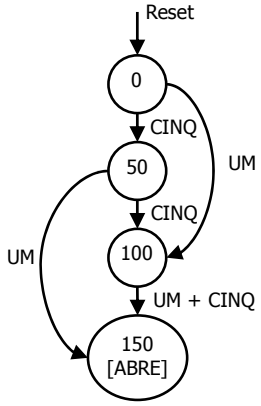
```
While(1) {
    Switch(State) {
        case State0 :
            //State = next state;
            break;
        case State1 :
            //State = next state;
            break;
        …
        default:
            State = State0; // just in case!
            break;
    }
    // clear_wdt();
}
```


Implementação em Software de FSMs



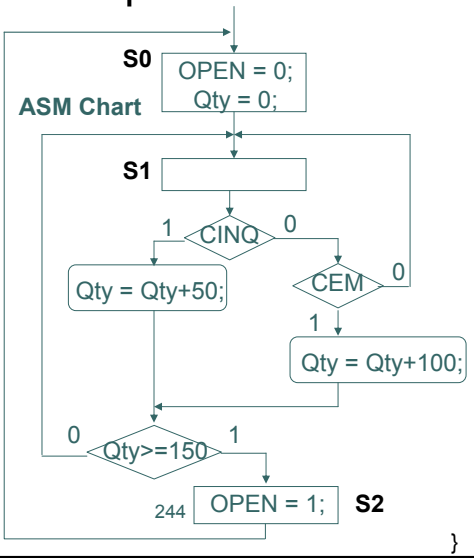

```
While(1) {
    Switch(State) {
        case State0 :
          if(CINQ == '1') State = State50;
          else if(UM == '1') State = State100;
          else State = State0;
          break;
        case State50 :
          //State = next state;
          break;
        …
        default:
          State = State0; // just in case!
          break;
    }
}
```

Implementação em Software de FSMs



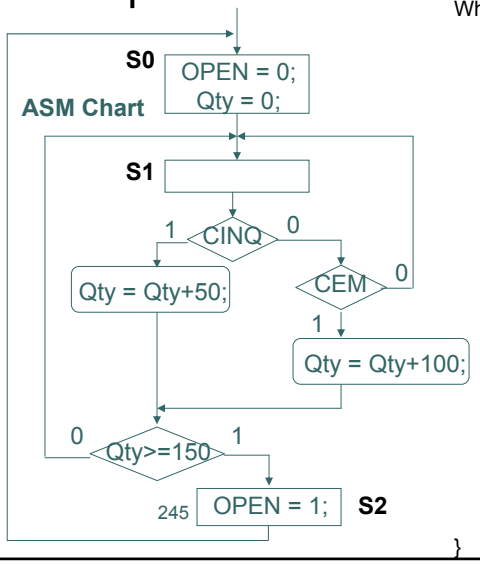

```
While(1) {
    Switch(State) {
        case State0 :
          if(transUp(CINQ)) State = State50;
          else if(transUp(UM)) State = State100;
          else State = State0;
          break;
        case State50 :
          //State = next state;
          break;
        …
        default:
          State = State0; // just in case!
          break;
    }
}
```


Implementação em Software de FSMDs



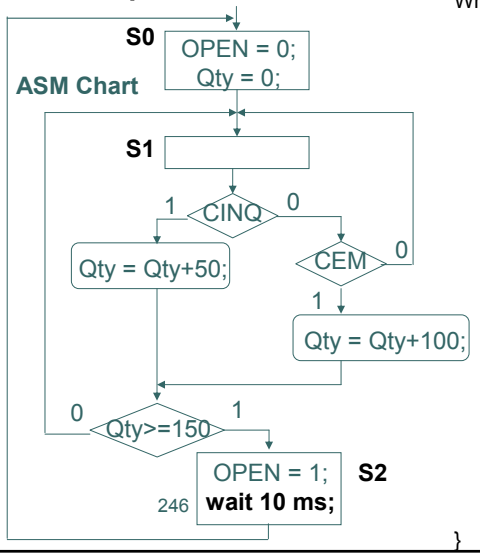

```
While(1) {
    Switch(State) {
        case S0 :
          OPEN = 0; Qty = 0;
          State = S1;
          break;
        case S1 :
          if(CINQ == 1) Qty = Qty + 50;
          else if(CEM == 1) Qty = Qty + 100;
          else { State = S1; break;}
          if(Qty >= 150) State = S2;
          else State = S1;
          break;
        case S2:
          OPEN = 1; State = S0;
          break;
        default:
          State = S0; // just in case!
          break;
    }
}
```

Implementação em Software de FSMDs



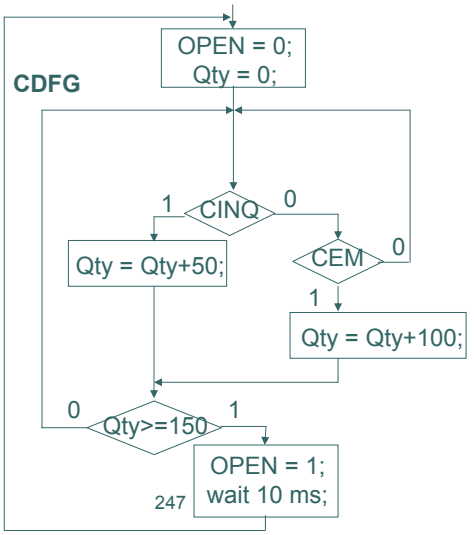

```
While(1) {
    Switch(State) {
        case S0 :
          OPEN = 0; Qty = 0;
          State = S1;
          break;
        case S1 :
          if(CINQ == 1) Qty = Qty + 50;
          else if(CEM == 1) Qty = Qty + 100;
          if(Qty >= 150) State = S2;
          else State = S1;
          break;
        case S2:
          OPEN = 1; State = S0;
          break;
        default:
          State = S0; // just in case!
          break;
    }
}
```

Implementação em Software de FSMDs




```
While(1) {
    Switch(State) {
        case S0 :
          OPEN = 0; Qty = 0;
          State = S1;
          break;
        case S1 :
          if(CINQ == 1) Qty = Qty + 50;
          else if(CEM == 1) Qty = Qty + 100;
          if(Qty >= 150) State = S2;
          else State = S1;
          break;
        case S2:
          OPEN = 1; wait(10); State = S0;
          break;
        default:
          State = S0; // just in case!
          break;
    }
}
```

Implementação em Software de CDFGs




```
While(1) {
    OPEN = 0;
    Qty = 0;
    do {
      if(CINQ == 1) Qty = Qty + 50;
      else if(CEM == 1) Qty = Qty + 100;
    } while(Qty < 150);
    OPEN = 1;
    wait(10);
}
```

Introdução aos Sistemas Embebidos

Implementações em Software (continuação)

Aceder a portos e registos

- Numa linguagem de alto-nível (e.g.,C, Java, etc.)
  - Utiliza-se assembly expandido (keyword: asm)
    - asm(“instruções assembly”)
    - Asm{“instruções assembly”}
  - Compilador extrai informação baseando-se no nome da variável
- Exemplo:
  - char PORTA; // para um compilador de C para o PIC poderia representar o registo relativo ao porto PORTA
  - Assim: PORTA = 0x0F; //Indicaria a escrita de 0x0F no PORTA
  - Char value = PORTA; //Indicaria a leitura de PORTA e armazenamento em value

Acesso a dispositivos mapeados na memória

- Baseado no linker
  - Extern volatile device_register;
  - Diz ao compilador que existe um recurso designado por device_register num local conhecido pelo linker
  - Após este declaração pode-se escrever ou ler no dispositivo como se estivesse a ler ou escrever numa variável global
  - Um comando do linker informa este da associação da variável com um endereço
    - Por exemplo, PUBLIC _device_register = $40000000

Acesso a dispositivos mapeados na memória

- Com ponteiros
  - Volatile unsigned short *io_regs;
  - Io_regs = (unsigned short *) 0x40000000; // faz com que o ponteiro io_regs aponte para o endereço de memória 0x40000000

Operações sobre bits

- No contexto de acesso a dispositivos são necessárias para escrever/ler num determinado bit
  - const char mask = 0x04;

  ou #define mask 0x04
  - extern volatile char dev_reg;

Operações sobre bits

- Coloca o terceiro bit menos significativo a ‘1’
  - Dev_reg = dev_reg | mask;
- Coloca o terceiro bit menos significativo a ‘0’
  - Dev_reg = dev_reg & ~(mask);
- Muda o valor do terçeiro bit menos significativo
  - Dev_reg = dev_reg ^ mask;

Volatile?

- Normalmente um compilador rege-se considerando o seguinte
  - um valor guardado numa variável não muda até à próxima instrução que o modifica
  - se não existir um uso de uma variável entre duas atribuições, só a última faz sentido

Volatile?

```
Int a, b;
A = 10;
// Instruções que não
   usam A
A = 20;
B = A * 100;
```

```
Int a, b;
B = A;
// Instruções que não
   modificam A
B = A;
A = B * 100;
```


Volatile?

- São estas optimizações sempre desejadas?
- Não. Por isso se utiliza a palavra **volatile** na declaração de uma variável
  - Indica ao compilador que esta variável pode ser modificada por outros
  - O compilador não faz optimizações para variáveis do tipo volatile
  - Por exemplo: não utiliza um registo do microprocessador para a variável...

Volatile?

- Vamos supor que queríamos pôr a piscar um LED que se encontra mapeado no 3º bit menos significativo da posição de memória 0x40000000

```
Main() {
    volatile unsigned char* io_regs = (unsigned char *)
    0x40000000;
    While(1) {
            *io_regs = 0x04; // acende
             delay(500); // delay 500 ms
            *io_regs = 0x00; // apaga
            delay(500); // delay 500 ms
    }
}
```


Volatile?

- Caso não utilizássemos volatile, o compilador poderia fazer a optimização seguinte:

```
#define mask 0x04
Main() {
    unsigned char* io_regs = (unsigned char *) 0x40000000;
    While(1) {
            delay(500); // delay 500 ms
            *io_regs = 0x00; // apaga
            delay(500); // delay 500 ms
    }
}
```

- Poderia até atribuir ao ponteiro io_regs um registo do microprocessador, por achar que com essa atribuição melhora o desempenho!

Exemplo

```
/* The green LED is controlled by bit 6. */
#define LED_GREEN    0x40
/* The green LED is controlled by bit 5. */
#define LED_YELLOW  0x20
/* The green LED is controlled by bit 4. */
#define LED_REG        0x10

/* setLed: ascende o LED especificado pela máscara do argumento */
Void setLed(unsigned char ledMask) {
     volatile unsigned char* io_regs = (unsigned char *) 0x40000000;
     *io_regs = *io_regs | ledMask;
} /* setLed */

/* toggleLed: apaga/acende o LED especificado pela máscara do argumento */
Void toggleLed(unsigned char ledMask) {
     volatile unsigned char* io_regs = (unsigned char *) 0x40000000;
     *io_regs = *io_regs ^ ledMask;
} /* toggleLed */
```


Exemplo de Polling a um dispositivo mapeado em memória

```
Void main() {
   volatile int *p_status = (int *) 0x4001;
   volatile int *p_data = (int *) 0x4000;
   while((*p_status & 0x01) != 0); // wait
   //*p_data = escreve valor;
   ...
}
```

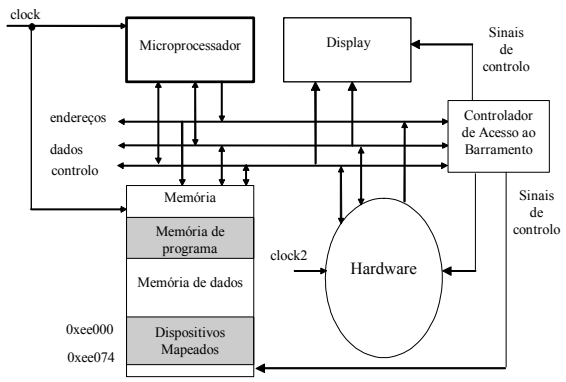
Introdução aos Sistemas Embebidos

Aplicação de Conhecimentos

Exemplo 1

- Pretende-se implementar parte de um sistema dedicado a processamento gráfico.
- Duas das funções do sistema estão relacionadas com:
  - determinação de um conjunto de pontos de um segmento de recta que une dois pontos dados
  - determinação de um conjunto de pontos do circulo, dado o centro e o raio
- Cada ponto é representado por 16 bits
- o nº máximo de pontos a determinar (100) é representado por 8 bits
- Para obedecer aos requisitos (tempo de execução), foi necessário utilizar hardware específico para realizar estas funções

Exemplo 1



- O dispositivo encarregue de realizar as duas funções encontra-se mapeado em memória
- O display encontra-se também mapeado em memória
- Existe um controlador que gere os acessos ao barramento

Exemplo 1

- O byte no endereço 0xee000 identifica o registo de status
  - O bit 2 identifica a função a realizar pelo dispositivo (‘1’ indica linha, ‘0’ indica circulo)
  - O bit 1 a ‘1’ indica se o dispositivo já terminou a execução
  - O bit 0 a ‘1’ indica ao dispositivo para começar a execução
- Restantes endereços:
  - O byte no endereço 0xee001 indica ao dispositivo o número de pontos a determinar
  - 0xee004 indica ao dispositivo o valor de X1 e Y1 (coordenadas do raio ou do primeiro ponto do segmento de recta)
  - 0xee008 indica ao dispositivo o valor de X2 e Y2, ou do raio
  - 0xee00C a 0xee06F armazenam os valores X, Y calculados pelo dispositivo

Exemplo 1

- O endereço 0xee074 é o endereço onde se encontra mapeado o display:
  - O conjunto de pontos a representar no ecrã deve ser enviado sequencialmente para o endereço anterior
  - Para cada ponto é enviado o triple: X, Y, Valor (em que valor é uma representação de 16 bits da cor de cada ponto/pixel) pela ordem indicada
  - Deve ser assumido que o os pixels do circulo ou da linha são identificados por Valor=0xffff (ponto do circulo) ou 0x0000 (não pertence ao circulo)

Exemplo 1

- Desenvolva, em linguagem C, o software seguinte
  - para as rotinas *line* e *circle* utilizando o dispositivo de hardware específico
    - void line(unsigned char N, unsigned short X1, unsigned short Y1, unsigned short X2, unsigned short Y2)
    - void circle(unsigned char N, unsigned short X1, unsigned short Y1, unsigned short radius)
  - Para a função *wait* que espera que a execução do dispositivo de hardware específico cesse
  - para a função *draw* que envia para o display o conjunto de pontos calculados
    - void draw(unsigned short N)

Exemplo 2

- o filtro FIR (*finite impulse response*) é muito usado no processamento de sinais
- No slide seguinte são apresentados dois modelos algorítmicos para o mesmo filtro:
  - No da esquerda as amostras são lidas de um porto de entrada e supõe-se que estas vão estando disponíveis em cadeia
  - No da direita supõe-se a existência de N amostras do sinal de entrada armazenadas em memória

Exemplo 2

```
#include “ports.h"

#define c0 2
#define c1 4
#define c2 4
#define c3 2

#define PORTA  0x1
#define PORTB  0x2

main() {
  int x, xd, xdd, xddd, y;
  x=0;
  xd=0;
  xdd=0;
  while(1) {
    xddd=xdd;
    xdd=xd;
    xd=x;
    receive(PORTA, &x);
    y = c0*x + c1*xd + c2*xdd + c3*xddd;
    send(PORTB, y);
  }
}
```

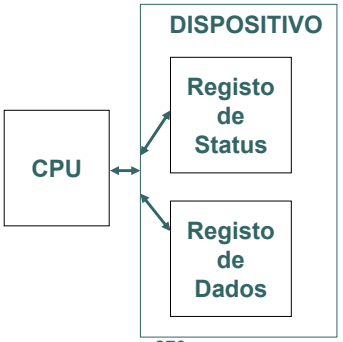
```
#include “ports.h"


#define PORTA  0x1
#define PORTB  0x2

#define N 100
#define NUM_TAPS  4

Int  c[] ={2, 4, 4, 2};

main() {
  int x, xd, xdd, xddd, y;

  for(int j=0; j<N-3; j++) {
     int f=0;
     for(int i=0; i<NUM_TAPS; i++) {
        f=f+c[i]*x[j+i];
     }
     y[j+3] = f;
  }
}
```

Exemplo 2

- Desenhe um CDFG para cada uma das versões do algoritmo do FIR
- Se o sistema fosse ligado a um canal de entrada e os resultados comunicados a um canal de saída, qual das duas versões escolheria? Porquê?

Exemplo 3

- Pretende-se escrever uma sequência de caracteres num dispositivo de saída
- O dispositivo tem dois registos
  - Um para o caracter a ser escrito
  - Um registo de status
    - Igual a '1' indica que o dispositivo está ocupado a escrever
    - Igual a '0' indica que a escrita já foi realizada



Exemplo 3

- Existem duas funções
  - Peek: lê de uma posição de memória

```
Int Peek(char *location) {
   return *location;
}
```

  - Poke: escreve numa posição de memória

```
void poke(char *location, char newval) {
   (*location) = newval;
}
```


Exemplo 3

- Definição do dispositivo
  - #define DEV1 0x1000
- Ler do registo do dispositivo
  - Dev_status = peek(DEV1);
- escreve 8 num registo do dispositivo
  - Poke(DEV1, 8);

Exemplo 3

- Char *mystring=“Hello, world”;
- Pretende-se escrever a string anterior no dispositivo
- Um caracter só deve ser escrito caso o dispositivo tenha cessado o tratamento do caracter anterior
- Mapeamento em memória do dispositivo
  - #define OUT_CHAR 0x1000 //registo para o caracter a ser escrito
  - #define OUT_STATUS 0x1001 //registo de status

Exemplo 3

```
Char *mystring=“Hello, world”;
Char *current_char;
Current_char=mystring;
While(*current_char != ‘\0’) {
   poke(OUT_CHAR, *current_char);
   while(peek(OUT_STATUS) != 0); // polling
   current_char++;
}
```

Exemplo 4

- Pretende-se implementar um controlador da buzina do cinto de segurança de um carro
- A buzina deve tocar enquanto a pessoa estiver sentada e não tiver o cinto de segurança colocado
- Antes da buzina tocar, deve haver um tempo de espera (fornecido por um temporizador externo) entre o instante em que a pessoa se sentou e a não colocação do cinto
- 3 entradas
  - Do sensor no banco que indica se alguém está sentado (SENTADO)
  - Do sensor do cinto que indica se o cinto está apertado (APERTADO)
  - Um sinal do temporizador que indica se um determinado tempo de espera esgotou (ACABOU)
- 1 saída: para a buzina (BUZINA = 1 (ON), 0 (OFF))
- 1 sinal que permite controlar por software a inicialização do Temporizador (INICIA)

Exemplo 4

- Desenhe um modelo gráfico do sistema a implementar
- Escreva o programa em C considerando que os sensores são ligados aos 3 bits menos significativos de PORTA, a buzina ao bit 3, e o bit 4 é utilizado para inicializar o temporizador (escrita de 0 inicializa)
- Existem duas funções em C para escrever/ler dos/nos portos de I/O do microprocessador
  - Int read(int PORT)
  - Void write(int PORT, int val)
- PORTA: identificador = 1

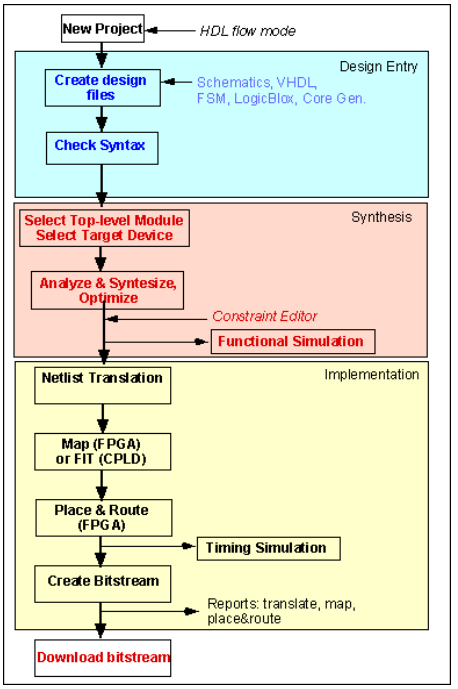
Introdução aos Sistemas Embebidos

Apresentação do fluxo de projecto de hardware digital em FPGAs utilizando síntese lógica

Fluxo de projecto de hardware digital em FPGAs utilizando síntese lógica

- Hardware a implementar é descrito numa linguagem de descrição de hardware (ex.: VHDL)
- A ferramenta de síntese lógica encarrega-se de criar a estrutura do circuito que implementa a nossa especificação
- A ferramenta de Place & Route coloca e mapeia o circuito no FPGA utilizado
- Por último é gerado o código binário que programa o hardware

Fluxo de projecto de hardware digital em FPGAs utilizando síntese lógica



Introdução aos Sistemas Embebidos

Comentários finais.

Comentários Finais

- Conteúdo programático das aulas teóricas foi cumprido
- Faltou
  - Exercício prático com interrupções
  - Trabalho final utilizando o kit de desenvolvimento com ligação a actuadores/sensores

Comentários Finais

- Os sistemas embebidos estão em todo o lado!
  - Omnipresentes!
  - Invisíveis!

Alguns artigos recentes

- IEEE Spectrum
  - Go Reconfigurable
    - http://www.spectrum.ieee.org/WEBONLY/publicfeature/dec03/1203reco.html
  - Machine Chameleon
    - http://www.spectrum.ieee.org/WEBONLY/publicfeature/dec03/1203pda.html
  - Talk to the Machine