

# GUIA DOS EMBARCADOS - 2020.2

IFCE – Eng.Computação



Cadeira - Sistemas Embarcados

PROF – Elias Teodoro da Silva Junior

## **SISTEMAS COMPUTACIONAIS EMBARCADOS (O COMPUTADOR EMBARCADO)**

- O QUE É.
- COMO SE COMPORTA.
- ONDE SE ENCONTRA.
- SE TEMOS RESTRIÇÕES EM SEU PROJETO.

### **DEFINIÇÕES:**

1 – “É QUALQUER DISPOSITIVO QUE INCLUI UM COMPUTADOR PROGRAMÁVEL MAS QUE NÃO É UM COMPUTADOR DE PROPÓSITO GERAL”

///(OU SEJA ELE CONTÉM UM COMPUTADOR PROGRAMÁVEL, POR ISSO DE CERTO MODO É UM COMPUTADOR).

2- “EMBUTIDO”(DEFINIÇÃO MAIS PRECISA): DIZER QUE UM COMPUTADOR ESTÁ EMBARCADO, SE REFERE A QUANDO O MESMO ESTÁ EM ALGUMA COISA QUE SE MOVE, DIZER QUE UM COMPUTADOR ESTÁ EMBARCADO EM ALGO QUE ESTÁ PARADO É ESTRANHO, MELHOR REFERIR-SE AO MESMO COMO EMBUTIDO. ([DEFINIÇÃO CLÁSSICA](#))

3-“SÃO SISTEMAS DE PROCESSAMENTO DE INFORMAÇÃO(COMPUTADORES) INCORPORADOS EM PRODUTOS FECHADOS(NO SENTIDO DE QUE SUA APLICAÇÃO É BEM CARACTERIZADA E DENTRO DISSO HÁ UM COMPUTADOR SERVINDO HÁ UMA FUNÇÃO BEM DEFINIDA)”(DEFINIÇÃO MAIS ABRANGENTE). ([DEFINIÇÃO CLÁSSICA](#))

4-HOJE EM DIA CADA VEZ MAIS TEM-SE QUERIDO AUMENTAR A FLEXIBILIDADE DOS SEMBS, PERMITR QUE ELES ASSUMAM APLICAÇÕES/FUNÇÕES DIFERENTES(ISSO TORNA O PRODUTO MAIS DIFÍCIL DE CARACTERIZAR EM ALGUMAS DAS DEFINIÇÕES BÁSICAS DOS SEMBS). ([HOJE EM DIA](#))

### **CARACTERÍSTICAS(DA MAIORIA DAS APLICAÇÕES EMBARCADAS):**

- O COMPUTADOR ESTÁ ALI DENTRO DE UM CERTO EQUIPAMENTO EMBARCADO PRA FAZER ELE FUNCIONAR(DE MANEIRAS EFICIENTES, MAIS BARATAS, DE MANEIRAS MAIS FLEXÍVEIS). ESSE COMPUTADOR NÃO VAI SERVIR A NENHUMA OUTRA UTILIDADE ALÉM DAQUELA QUE É A DO EQUIPAMENTO QUE ELE ESTÁ EMBARCADO.
- AO SE COMPRAR PRODUTOS CLASSIFICADOS COM SEMBS, MUITAS VEZES NEM PERCEBEMOS QUE TAIS EQUIPAMENTOS SÃO DE FATO SEMBS! OU SEJA QUE TEM UM COMPUTADOR EMBUTIDO NELES.
- A IDEIA DE UM SEMB É EXATAMENTE ESSA, QUE VC USE SEU EQUIPAMENTO E O COMPUTADOR ESTEJA ALI PARA AJUDÁ-LO A UTILIZAR ESSE EQUIPAMENTO, TORNANDO ESSE USO MAIS EFICIENTE, BARATO, TUDO ISSO PARA QUE VC NÃO PRECISE ESTAR CONSCIENTE DE QUE DENTRO DE TAL PRODUTO HAJA UM COMPUTADOR.O COMPUTADOR É EFICIENTE, ECONÔMICO, RESISTENTE DENTRO DO CONTEXTO(OBJETIVO DO DISPOSITIVO ELETRÔNICO, UM CELULAR, CARRO, RELÓGIO E ETC) NO QUAL ELE ESTÁ INSERIDO!

### **EXEMPLOS:**

[““QUASE TUDO TUDO HOJE EM DIA!””](#)



"ESSES PRODUTOS TEM EM COMUM NÃO SÓ O FATO DE SEREM SEMBS, MAS DE SEUS USUÁRIOS PERCEBEREM NELES MUITAS CARACTERÍSTICAS MENOS AS DE UM COMPUTADOR EM SI, PORQUE ESSSE É O PROPÓSITO DE UMSEMB, OMITIR ESSA NECESSIDADE DE SE ATENTAR A ESSE COMPUTADOR" – ISSO NÃO ESTÁ NAS PREOCUPAÇÕES DO USUÁRIO, VC DESCREVE DIFERENCIAR ESSES OBJETOS EM TUDO MENOS USANDO DAS CARACTERÍSTICAS DE SEUS COMPUTADORES EMBUTIDOS

#### CARACTERÍSTICAS-PRINCIPAIS(DA MAIORIA DAS APLICAÇÕES EMBARCADAS):

- UM PROJETO DE UM SISTEMA EMBARCADO SEMPRE PROCURA TIRAR VANTAGENS DAS PROPRIEDADES QUE ESTÃO DISPONÍVEIS NAQUELA APLICAÇÃO EM ÁRTICULAR.
- O ESPAÇO DE MANOBRA QUE O PROJETISTA DE UM SEMB TEM, QUE É MAIS LIMITADO PARA O ESCOPO DO QUE O OBJETO DEVE FAZER, VAI SER MUITO IMPORTANTE PARA CONSEGUIR ENTREGAR A ALTA EFICIÊNCIA QUE A APLICAÇÃO DE TAL OBJETO PEDE! SE DE UM LADO ELE PRECISA GASTAR POUCA BATERIA COM A APLICAÇÃO POR OUTRO LADO ELE NÃO PRECISA SER CAPAZ DE FAZER QUALQUER TIPO DE CONTA E ELAS NEM PRECISAM AS VEZES TÃO RÁPIDAS!



**"EXPLORAÇÃO DO ESPAÇO DE PROJETO"**

**(QUANDO UM ENGENHEIRO ESTIVER PROJETANDO UM HARDWARE E UM SOFTWARE DE UM SEMB VAI PODER TRABALHAR PRA ENTREGAR O MELHOR RESULTADO DESSE PROJETO A SEU PÚBLICO ALVO)**

- A COMPUTAÇÃO QUE TÁ ALI NAQUELE PRODUTO EMBARCADO É UM MEIO PARA AQUILO QUE O USUÁRIO PERCEBE, COMO UM RELÓDIO SMART, O USUÁRIO SÓ QUER VER OS RESULTADOS NA INTERFACE GRÁFICA, ELE NÃO QUER SABER SE É UM COMPUTADOR QUE OS ESTÁ PROCESSANDO E MOSTRANDO, ISSO PORQUE ANOS ATRÁS TAIS RESULTADOS TAMBÉM ERAM MOSTRADOS POR MEIO DE OUTRAS TECNOLOGIAS, HOJE SÃO COMPUTADORES.

#### ESPAÇOS DAS APLICAÇÕES EMBARCADAS HOJE EM DIA(QUASE TUDO HOJE EM DIA TEM UMA):

- ELETRÔNICA DE CONSUMO: câmeras, fornos microondas, tocadore de CD/DVD, receptor de TV.  
(É O MAIOR MECADO DE APLICAÇÕES EMBARCADAS E ONDE HÁ A MAIOR QUANTIDADE DE PROCESSADORES EMBARCADOS)
- TELECOMUNICAÇÕES: telefones celulares, centrais, telefonicas(PABX).
- VEÍCULOS: controle de injeção de combustível, freio antitravamento (ABS), câmbio automático.
- CONTROLE INDUSTRIAL: robôs, esteiras de transporte, instrumentos de medição .  
(GRANDE NECESSIDADE DE COMPUTADORES PARA AUTOMAÇÃO DE PROCESSOS)

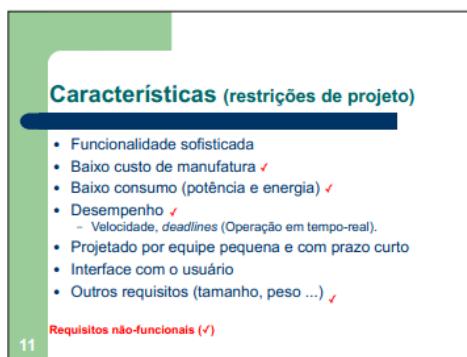
#### A CHEGADA DE UM PRODUTO EMBARCADO AO MERCADO É CRUCIAL:



### TIME-TO-MARKET – “O TEMPO PARA UM PRODUTO CHEGAR AO MERCADO”.

- Tem muito a ver com a engenharia que faz o hardware e quem faz o software!

- Se uma organização não conseguir entregar um produto tão rápido quanto o concorrente, esse concorrente vai ter mais oportunidade de retorno financeiro, e se esses atrasos forem recorrentes, essa organização vai acabar perdendo retorno financeiro e falindo/fechando.



//////////////////////////////

## SISTEMAS COMPUTACIONAIS EMBARCADOS – SISTEMAS DE ENTRADA E SAÍDA

### DEFINIÇÕES:

- AQUELES EQUIPAMENTOS QUE FAZEM INTERFACE DO NOSSO COMPUTADOR COM O MUNDO EXTERNO.

### CARACTERÍSTICAS(NA MAIORIA DAS APLICAÇÕES EMBARCADAS):

- COSTUMAM APRESENTAR MUITO CONTATO COM O MUNDO EXTERIOR.

- PRECISAM FAZER INTERFACES COM MEDIDAS EXTERNAS(TEMPERATURA, ACELERAÇÃO).POR ISSO NÃO É MUITO COMUM O USO DE TECLADOS E DISPLAYS.

### GRADEZAS EXTERNAS QUE O COMPUTADOR EMBARCADO PRECISA ACESSAR:

1- MUITAS DELAS SÃO ANALÓGICAS! DAÍ A EXISTÊNCIA DE SISTEMAS QUE FAZEM A CONVERSÃO DE ANALÓGICO PRA DIGITAL(ENTRADA COM INFORMAÇÃO DIGITAL NO COMPUTADOR) E DE DIGITAL PRA ANALÓGICO(SAIR COM INFORMAÇÃO ANALÓGICA DO COMPUTADOR).

### 2- (SISTEMAS)ANALÓGICAS vs DIGITAIS:

#### \*SINAL ANALÓGICO:



- ADMITE VÁRIOS VALORES (INFINITOS ATÉ).
- VARIA DE FORMA CONTÍNUA (VALORES CONTÍNUOS NA FAIXA TRABALHADA).
- QUALQUER VALOR É POSSÍVEL DENTRO DE UMA CERTA FAIXA
- MEU SINAL/QUANTIDADE ANALÓGICA PODE VARIA LIVREMENTE DENTRO DE UMA FAIXA DE VALORES.
- A **GRANDE DESVANTAGEM** DE SE TRABALHAR COM VALORES ANALÓGICOS PARA TRANSMITIR INFORMAÇÕES DE UM LUGAR PARA OUTRO:

- UM SISTEMA QUE RECEBE UM SINAL ANALÓGICO ADMITE QUALQUER VALOR COMO VÁLIDO ELE NÃO TEM COMO SABER SE MISTURADO COM ESSA SENÓIDE DA IMAGEM ACIMA POR EXEMPLO VEM ALGUM “RUÍDO” PORQUE UM RUÍDO TAMBÉM É UM VALOR ANALÓGICO VÁLIDO!

#### **\*SINAL DIGITAL:**



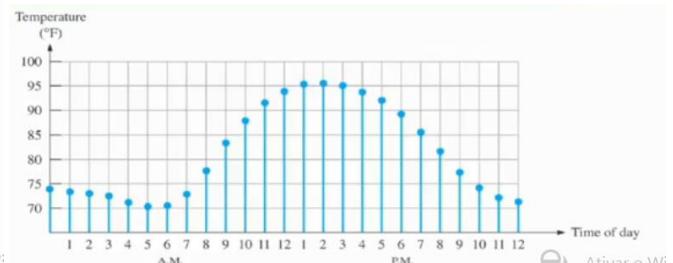
- DISCRETIZAÇÃO DE VALORES.
- EXISTEM VALORES PRÉ-ESTABELECIDOS.
- MEU SINAL/QUANTIDADE DIGITAL SÓ PODE ASSUMIR ALGUM DESTES VALORES PRÉ-ESTABELECIDOS QUALQUER VALOR QUE NÃO CAIA NESSES PRÉ-DEFINIDOS É CONSIDERADO INVÁLIDO. (**\*ISSO GERA UMA FACILIDADE NA HORA DE TRATAR ESSES SINAIS!**)
- A **GRANDE VANTAGEM** DE SE TRABALHAR COM VALORES DIGITAIS PARA TRANSMITIR INFORMAÇÕES DE UM LUGAR PARA OUTRO:

  - QUANDO SE TRABALHA COM VALORES DIGITAIS EM UM SINAL, VOCÊ JÁ PRÉ-ESTABELECEU ALGUNS VALORES COMO VÁLIDOS, ENTÃO SE HÁ UM RUÍDO E ELE CAIR EM VALORES QUE NÃO ESTÃO DENTRO DOS VALORES PREVISTOS PODEMOS FACILMENTE DESCARTÁ-LO COMO UMA INFORMAÇÃO NÃO-VÁLIDA!

- AO PEGAR VALORES ANALÓGICOS E COLOCÁ-LOS DENTRO DE UM COMPUTADOR TEM-SE QUE DEFINIR FORMAS DE “DISCRETIZAÇÃO” PARA ELES. NÃO SE PODERÁ TER QUAISQUER VALORES, TERÁ QUE SE ESTABELECER ALGUNS VALORES PRÉ-DEFINIDOS E QUAISQUER VALORES FORA DOS PRÉ-DEFINIDOS TERÁ QUE SER ARREDONDADO PARA ALGUM DESSES VALORES. VEJA AS DUAS IMAGENS ABAIXO DE UM GRÁFICO COMO SINAL ANALÓGICO E DEPOIS COMO DIGITAL:



\*(Quantidade Analógica)



\*(Quantidade Digital)

- A LIMITAÇÃO NA QUANTIDADE DE VALORES QUE PODE-SE CAPTURAR POR TEMPO É IMPORTANTE PARA DELIMITAR A QUANTIDADE DE MEMÓRIA QUE IRÁ PRECISAR PARA ARMAZENAR ESSAS LEITURAS, COMO NA TEMPERATURA NO EXEMPLO DOS GRÁFICOS ACIMA.

- AO FALAR DE COMPUTADORES EMBARCADOS, QUE PRECISAM SER ECONÔMICOS NO TANTO DE MEMÓRIA E OUTROS RECURSOS QUE GASTAM, PRECISAMOS DELIMITAR MUITO BEM A QUANTIDADE DE MEMÓRIA QUE VAMOS PRECISAR PARA NOSSA APLICAÇÃO. ENTÃO É PRECISO QUE AO SE PROJETAR UMA APLICAÇÃO COMO ESSA DOS GRÁFICOS DA TEMPERATURA POR EXEMPLO VC SE PREOCUPE EM CAPTURAR TEMPERATURAS NA “MEDIDA DA NECESSIDADE”.

### 3- QUANDO O COMPUTADOR PRECISA LIDAR COM VALORES ANALÓGICOS:

- PODE FAZER ISSO DE DUAS FORMAS:

1- QUANDO RECEBE UM VALOR ANALÓGICO E PRECISA CONVERTER DE ANALÓGICO PARA DIGITAL.

(ANALÓGICO PARA DIGITAL)

2- QUANDO PRECISA PRODUZIR UM VALOR ANALÓGICO PARA ENTREGAR PARA O MUNDO EXTERIOR. (DIGITAL PARA ANALÓGICO)

**>>>CONVERSOR D/A (DIGITAL PARA ANALÓGICO) – DISPOSITIVOS DE SAÍDA:**

- REGISTRADORES OU PORTAS DE SAÍDA DE UM MICROCONTROLADOR SERVEM COMO AS ENTRADAS QUE ENVIARAM OS SINAIS ANALÓGICOS(VALORES BINÁRIOS) PARA O CONVERSOR.

- **DAC (CONVERSOR D/A) R-2R:** A REDE R-2R( CIRUITOS QUE POSSUEM EM SUAS COMPOSIÇÕES SOMENTE RESISTORES DE VALOR R E O DOBRO DE TAL VALOR) É A QUE MAIS É ENCONTRADA DENTRO DOS CIRCUITOS INTEGRADOS QUE FAZEM A FUNÇÃO DE CONVERSORES D/A, ISSO PORQUE TAL REDE FACILITA MUITO A FABRICAÇÃO DESSE COMPONENTE ELETRÔNICO.

- OS CONVERSORES D/A DO MUNDO REAL TEM ESSA PEQUENA DIFERENÇA ENTRE UM NÚMERO BINÁRIO E OUTRO(EM SEUS VALORES). **SUAS LINEARIDES SÃO MAIS BAIIXAS QUE AS DOS TEÓRICOS!** POR EXEMPLO OS VALORES DOS RESISTORES TEÓRICOS SÃO SEMPRE OS MESMOS JÁ ESTABELECIDOS, MAS NO MUNDO REAL NÃO É ASSIM, HÁ MUITAS EXCEÇÕES E RESISTORES DE VALORES QUE ERAM PRA SER IGUAIS SÃO DE FORMA APROXIMADA MAS NÃO IGUAL. **FABRINCANTES** ENTÃO PROCURAM PRODUZIR CIRCUITOS O MAIS PRÓXIMO POSSÍVEL DOS CASOS TEÓRICOS COMO O R-2R PARA AUMENTAR AO MÁXIMO A LINEARIDADE DO CONVERSOR.

- CARACTERÍSTICAS DE DISCRETIZAÇÃO DO CONVERSOR D/A:

## 1- NO VALOR DE TENSÃO QUE ELE É CAPAZ DE PRODUZIR:

- Embora grandezas analógicas possam assumir infinitos valores dentro de uma certa faixa, num circuito D/A tais grandezas não poderão chegar nesse ideal teórico porque sempre haverá números finitos de números binários (os bits!), na imagem abaixo por exemplo, temos 4 bits:

**Conversor D/A - funcionamento**

A B C D Saída (V)

0	0	0	0	0,00
0	0	0	1	0,22
0	0	1	0	0,44
0	0	1	1	0,66
0	1	0	0	0,88
0	1	0	1	1,10
0	1	1	0	1,32
0	1	1	1	1,54
1	0	0	0	1,76
1	0	0	1	1,98
1	0	1	0	2,20
1	0	1	1	2,42
1	1	0	0	2,64
1	1	0	1	2,86
1	1	1	0	3,08
1	1	1	1	3,30

$$V_{Saida} = Digital * \frac{V_{max}}{2^{bits} - 1}$$

$$V_{Saida} = Digital * \frac{3,3}{15}$$

$$V_{Saida} = 1 * \frac{3,3}{15}$$

- Número finito de valores analógicos

(CONVERSOR D/A – RESOLUÇÃO 4 BITS)

- E SE NESSA APLICAÇÃO DA IMAGEM ACIMA NO CONVERSOR D/A EU QUISSE PRODUZIR UMA QUANTIDADE MAIOR DE VALORES ANALÓGICOS? DIGAMOS QUE QUERO PRODUZIR AGORA UM VALOR ENTRE 0 E 0,22, O QUE FARIA?

\*RESPOSTA: ACRESCENTAR UM BIT!

**Conversor D/A - funcionamento**

A B C D E Saída (V)

0	0	0	0	0	0,00
0	0	0	0	1	0,11
0	0	0	1	0	0,22
0	0	0	1	1	0,33
0	0	1	0	0	0,44
0	0	1	0	1	0,55
0	0	1	1	0	0,66
0	0	1	1	1	0,77
0	1	0	0	0	0,88
0	1	0	0	1	0,99
0	1	0	1	0	1,10
0	1	0	1	1	1,21
0	1	1	0	0	1,32
0	1	1	0	1	1,43
0	1	1	1	0	1,54
0	1	1	1	1	1,65

$$V_{Saida} = Digital * \frac{V_{max}}{2^{bits} - 1}$$

$$V_{Saida} = Digital * \frac{3,3}{31}$$

$$V_{Saida} = 1 * \frac{3,3}{31}$$

- Número finito de valores analógicos
- Como gerar uma tensão com valor maior que '0000' e menor que '0001'?

(CONVERSOR D/A – RESOLUÇÃO 5 BITS)

- PARA PRODUZIR UMA QUANTIDADE MAIOR DE COMBINAÇÕES VOU PRECISAR DE MAIS BITS! ASSIM, MAMTEMOS A MESMA FAIXA QUE ESTAMOS TRABALHANDO SÓ QUE AGORA TEMOS UMA QUANTIDADE MAIOR DE VALORES POSSÍVEIS, OU SEJA, AGORA TEMOS UMA **RESOLUÇÃO MAIOR!**

\*CARACTERÍSTICA DO CONVERSOR D/A: "RESOLUÇÃO"

- A RESOLUÇÃO DE UM CONVERSOR D/A ESTÁ DEFINIDA PELA QUANTIDADE DE BITS QUE ELE UTILIZA! PARA AUMENTAR A RESOLUÇÃO, BASTA INTRODUIR MAIS BITS A MEDIDA DA NECESSIDADE.

- EM TERMOS COMERCIAIS, É MUITO COMUM CONVERSORES DE 8, 10, 12 E 14 BITS. NORMALMENTE SÃO NÚMEROS PARES.

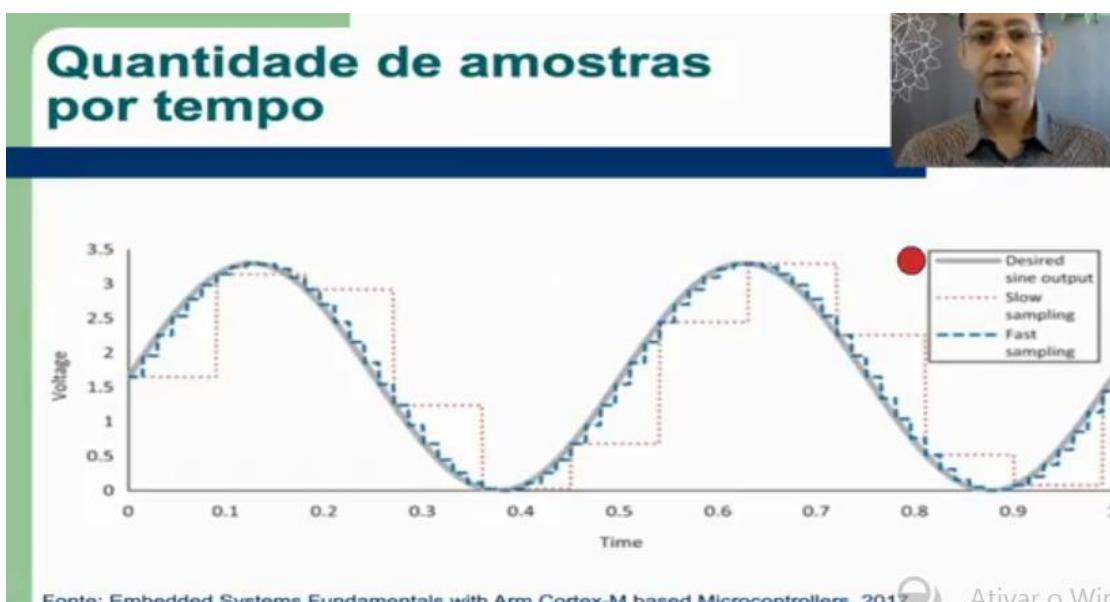
- UMA QUANTIDADE MAIOR DE BITS VAI IMPLICAR NUMA COMPLEXIDADE MAIOR DO CIRCUITO. MAS A ONDA ANALÓGICA PRODUZIDA NA SAÍDA DO CONVERSOR TENHA DEGRAUS MENORES, SE APROXIMANDO DA "RAMPA DENTE DE SERRA" IDEAL (VENDO UMA RAMPA PERFEITA ENTRE O VALOR ZERO E O VALOR MÁXIMO ANALÓGICO QUE EU POSSO PRODUZIR).

## 2- NO TEMPO, O INTERVALO ENTRE UMA CONVERSÃO E OUTRA:



- SE EU QUISER COM MEU CONVERSOR D/A PRODUZIR A ONDA CINZA DA IMAGEM ACIMA, TEREI QUE UTILIZAR DE ALGUM MÉTODO PARA CALCULAR ESSES VALORES.

- SE O PROCESSADOR ENTREGAR PARA O CONVERSOR D/A EM INTERVALOS MAIORES, TEREMOS UMA ONDA BEM MENOS PRÓXIMA DA DESEJADA NO SINAL DE SAÍDA COM INTERVALOS MUITO MAIORES E UM FORMATO MENOS PRÓXIMO AO DESEJADO. VEJA A ONDA VERMELHA NA IMAGEM ABAIXO:



- INTERVALOS MENORES DE TEMPO TE PERMITE FAZER UMA ONDA BEM MAIS PRÓXIMA DA DESEJADA, MESMO QUE NO SINAL ANALÓGICO A ONDA SAIA CERRILHADA, SE O PROCESSADOR CONSEGUIR FORNECER RESULTADOS EM INTERVALOS MENORES DE TEMPO JÁ É MELHOR. O NOME DE TAIS EVENTOS CHAMA-SE TAXA DE AMOSTRAGEM.

### \*CARACTERÍSTICA DO CONVERSOR D/A: "TAXA DE AMOSTRAGEM"

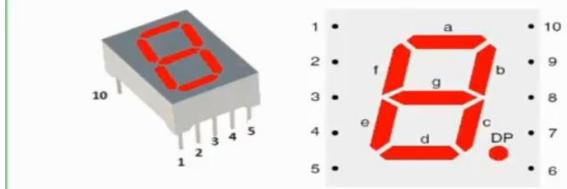
- CADA ENTREGA DE VALOR PRA SER CONVERTIDO É CONSIDERADO UMA AMOSTRA, ENTÃO SE UM PROCESADOR ENTREGA UM VALOR NUMA TAXA MAIS ALTA A ONDA ANALÓGICA NA SAÍDA SERÁ MAIS PRÓXIMA DA ESPERADA.

- A RESOLUÇÃO E A TAXA DE AMOSTRAGEM VÃO CONTRIBUIR PARA QUE O SINAL ENTREGUE AO MUNDO EXTERNO SEJA MAIS PARECIDO OU MENOS PARECIDO COM O DESEJÁVEL TEORICAMENTE!

### DISPLAYS(MOSTRADORES):

#### -LEDs(DISPLAY DE 7 SEGMENTOS):

## LED - 7 segmentos



-Uma das soluções mais baratas(baixo custo e pouca energia, solução ideal para aplicações mais simples, COM POUCA COMPLEXIDADE DE INFORMAÇÃO).

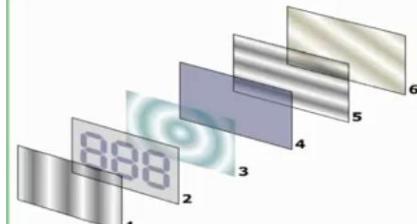
-Só nos permite produzir números ou letras, não há muita liberdade para se manipular as informações de saída no sistema.

### -LCD:

- Pode produzir imagens gráficas, de alta definição e colorida. Ou textos e números como num display de 7 segmentos.

- A vantagem do LCD em relação ao LED comum de 7 segmentos é seu baixo consumo de energia. Para se controlar um LCD utiliza-se somente CAMPO ELÉTRICO, praticamente não há consumo de corrente para fazer esse controle da luz polarizada. Tanto que o único gasto importante de energia nele é lâmpada que existe na camada 6 da figura ilustrativa de um LCD comum abaixo, mas caso não se use uma lâmpada e sim algo como um espelho para fornecer ao sistema energia iluminação o sistema teria um consumo praticamente 0.

## LCD - funcionamento



1. Filtro Polarizador
2. Substrato de vidro
3. Cristal Líquido
4. Substrato de vidro
5. Filtro Polarizador
6. Superfície refletora ou lâmpada

- Quando se quer preto no LCD, APENAS SE BLOQUEIA A PASSAGEM DA LUZ, MAS HÁ AINDA LUZ SENDO PRODUZIDA E ENERGIA GASTA.

Por que o monitor LCD gasta mais energia

... que o monitor de LED?

RESPOSTA:

1- O problema com as lâmpadas CCFL usadas na LCD é que elas são acesas todas de uma vez, não só consumindo mais energia como tornando o contraste mal definido em certos momentos, e os pontos escuros da imagem, não tão escuros.

2- Porque o display LCD precisa de energia para exibir todas as cores, enquanto o LED pode apagar quando exibir a cor preta.

### -OLED(DISPLAY DE LEDS):

- É uma alternativa melhor para não usar um LCD e ter tal lâmpada continuamente acesa na camada mais atrás(como visto na imagem acima na camada 6).

- Consome bem menos energia que um LCD E LED DE DISPLAY DE 7 SEGMENTOS.

- Substitui o display LCD também na produção de gráficos de alta definição.

- Tem LUZ PRÓPRIA.

- **PRETO VERDADEIRO:** Se a tela for preta, todos os LEDS do display saõ apagados!(REDUÇÃO EFETIVA NO CONSUMO). Dá contrastes melhores e percepção de qualidade de imagem melhor.

- Sem desperdício de Luz e Energia(ótima opção para se usar em equipamentos que se alimentem por baterias).

**-ATUADORES:**

- Equipamentos utilizados para atuar em algum processo.

- Podem ser sistemas mecânicos ou de outras grandezas físicas. Dispositivos mecânicos para movimentação ou controle.

- Exemplos: Relés, Motores, Pneumáticos, Hidráulicos, Piezoelétricos, Térmico.

- Ponte H(Driver p/ motor CC):

- Ponte H combinada com motor de corrente contínua(motor CC).

- Motores CC são utilizados para casos que se quer manter velocidades contantes e se quiser, aumentá-la ou diminuí-la um pouco(pode rodar um pouco mais rápido ou um pouco mais devagar).

- A ponte H é o circuito utilizado para comandar o motor.

- Motor de passo:

- Ele não fica rodando continuamente.

- Pode-se controlar quantos graus ele vai se mover numa direção ou na outra.

-Tipicamente é acionado por uma placa (MICROCONTROLADOR) onde se tem DUAS ENTRADAS DE CONTROLE:

- Uma que vai dar a direção do motor.

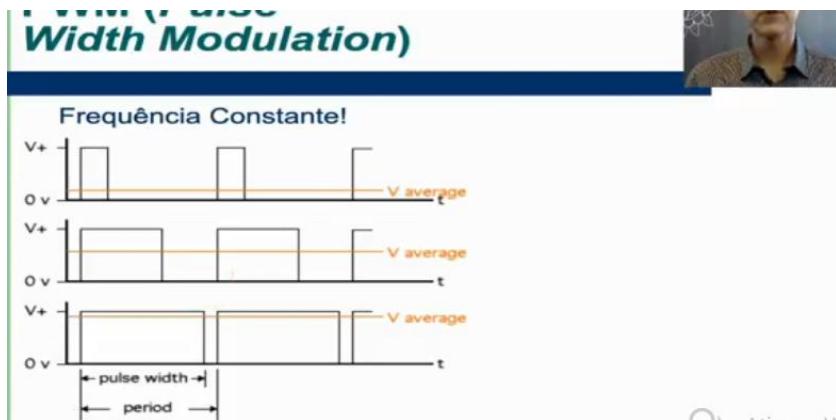
- Outra vai dar o passos do motor.

- Quando para de receber pulsos o motor PARA e trava na posição em que ficou. [Isso torna ele muito interessante para controle de direção.](#)

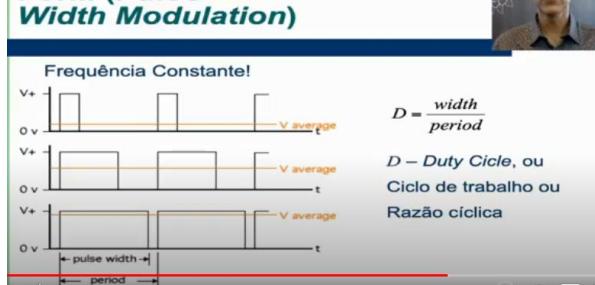
- **EXEMPLO:** Em carros de controle remoto por exemplo, quem fica responsável por fazer a tração do carro é o motor CC e o motor de passo é usado para dar a direção do movimento, está associado ao eixo dianteiro do carro pra fazer esse controle.

**- PWM (“PULSE WIDTH MODULATION”-MODULAÇÃO POR LARGURA DE PULSO):**

- VAI SER MUITO INTERESSANTE PARA QUE OS MICROCONTROLADORES POSSAM ACIONAR CARGAS.
- A FREQUÊNCIA É CONSTANTE! POSSO TER PULSOS MAIS CURTOS OU MAIS LONGOS, MAS ELES SEMPRE VÃO APARECER NA MESMA TAXA, COMO NA IMAGEM ABAIXO:



- Quando um pulso é mais curto ele vai resultar em uma quantidade menor de energia entregue ao sistema! O que vai produzir uma valor médio(na imagem acima: “V average”) bem mais baixo do que pulsos maiores.
- Essa técnica PWM é que é usada na Ponte H(Driver p/ motor CC) para mudar um pouca a velocidade do motor CC, tudo com base na energia dos pulsos enviados. Pulses mais longos dá ao motor CC velocidade mais alta e pulsos mais curtos uma velocidade mais baixa.



- A técnica PWM com frequência CONSTANTE utiliza o parâmetro D para definir a largura do pulso. E esse D fica independente da frequência que está sendo utilizada. O tamanho do pulso é dividido pelo período e isso dá um número que pode variar entre 0 e 1.

- O “D”(ciclo de trabalho ou razão cíclica) pode ser configurado no seu microcontrolador pra definir qual é essa taxa que se quer que seja entregue na saída do PWM.

**- DUTY CYCLE:** “o ciclo de trabalho, razão cíclica ou fator de ciclo de um sistema corresponde à fração de tempo em que este se encontra em estado ativo.” **Duty Cycle ou Ciclo de trabalho** é a proporção de tempo que uma carga ou circuito está LIGADO em comparação com o tempo em que a carga ou circuito está DESLIGADO. Ou seja, é o tempo em que determinado circuito fica ON comparado com OFF. Esse tempo do ciclo de trabalho é expresso como uma porcentagem do tempo de ligado. Por exemplo, se um círculo de trabalho é de 60%, quer dizer que o sinal está LIGADO 60% das vezes e desligado os outros 40%.

**- APLICAÇÃO DO PWM:** Essa técnica do PWM pode ser tanto usada num motor para controlar sua velocidade quanto numa série de outras aplicações como para controlar o brilho de um led por exemplo:

- Maior o valor do pulso ou seja maior o valor do D maior o brilho do led e menor o valor do D menor o brilho.

- Obs:

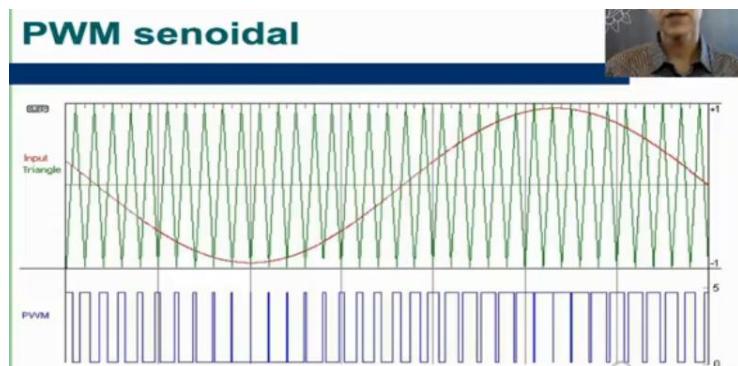
- O LED é um SISTEMA NÃO LINEAR, e por isso ele não permite que sejam aplicadas sobre ele TENSÕES DIFERENTES! Ele tem uma TENSÃO NOMINAL DE OPERAÇÃO e essa tensão típica é exatamente a tensão que tem de ser aplicada sobre ele, se aplicar uma tensão mais baixa do que essa o LED NÃO ASCENDE e se aplicar uma mais alta ele QUEIMA, ou seja não se tem muita liberdade para controlar o brilho controlando a tensão, para haver esse controle é que se usar do PWM. Com o PWM fica-se LIGANDO e DESLIGANDO o LED numa FREQUÊNCIA ACIMA daquela que o OLHO HUMANO CONSEGUE PERCEBER QUE O LED TÁ ASCENDENDO E APAGANDO. Nosso olho vai PERCEBER O BRILHO MÉDIO DO LED. Se o LED passar mais tempo aceso do que apagado vamos perceber o brilho mais alto do que se ele passar mais tempo LIGADO do que DESLIGADO.

- RESISTORES são sistemas lineares, se quiser aplicar um tensão diferente nele para mudar seu calor dissipado de acordo com essa tensão irá ter uma corrente proporcional e ele vai aquecer proporcionalmente.

- O conceito de PWM é muito importante para permitir acionar cargas NÃO-LINEARES ( COMO O LED) e CARGAS DE ALTA POTÊNCIA através de um microcontrolador.

- PWM PARA PRODUZIR UMA ONDA SENOIDAL:

- Ao variar a largura do pulso produzido pelo PWM gradualmente de maneira a obter valores mais baixos ou mais altos, isso seguindo a taxa de crescimento de UMA SENÓIDE. Isso para um sistema que precisa de um senóide em sua entrada vai mandar para o mesmo uma senóide como desejado misturada com uma ALTA FREQUÊNCIA que é frequência do PWM, que na imagem a baixo seria essa onda VERDE e TRIANGULAR(fácilmente filtrável para uma vez depois da filtragem o sistema receber somente a onda senoidal que precisava):



- ISSO É UTILIZADO PARA COMANDAR CERTOS TIPOS DE MOTORES QUE OPERAM COM ONDAS SENÓIDAIAS. Um sistema PWM que precisa produzir ondas senoidais deve ser capaz de controlar a AMPLITUDE E A FREQUÊNCIA da senóide.



>>>CONVERSOR A/D (ANALÓGICO PARA DIGITAL ou ADC) – DISPOSITIVOS DE ENTRADA:

- DISPOSITIVOS DE ENTRADA: UTILIZADOS PARA RECEBER INFORMAÇÕES PARA PROCESSAMENTO NO COMPUTADOR. COMPUTADORES EMBARCADOS SEMPRE VÃO

**PRECISAR MUITO DE ESTAR EM CONTATO COM GRANDEZAS FÍSICAS QUE VARIAM DE FORMA ANALÓGICA(POR ISSO O USO DE CONVERSORES).**

- Exemplos: Sensores, Conversores Analógico/Digital, S&H (Sample and Hold), Contadores/Temporizadores.

**-SENSORES:** Pensando nas grandezas FÍSICAS que podem ser utilizadas por uma aplicação embarcada virtualmente é possível que qualquer uma delas seja transformada em corrente elétrica pra que assim nós tenhamos um sensor que permite a entrada de dados em nosso computador. **(VIRTUALMENTE É POSSÍVEL CONSTRUIR SENSORES PARA TODAS AS GRANDEZAS FÍSICAS).**

- Tempo de resposta: Depende muito da grandeza física observada, as vezes um tempo para um resposta efetiva/confiável do sensor sob alguma medida precisa ser mais longo e as vezes para ser mais preciso precisa ser mais curto(tempo de respostas mais rápido).

-Exemplos: Acústicos(microfone),Ópticos(CCD),Elétricos e Magnéticos(efeito Hall),Químicos(pH,Oxigênio),Mecânicos(Acelerômetro),Pressão(Barômetro).

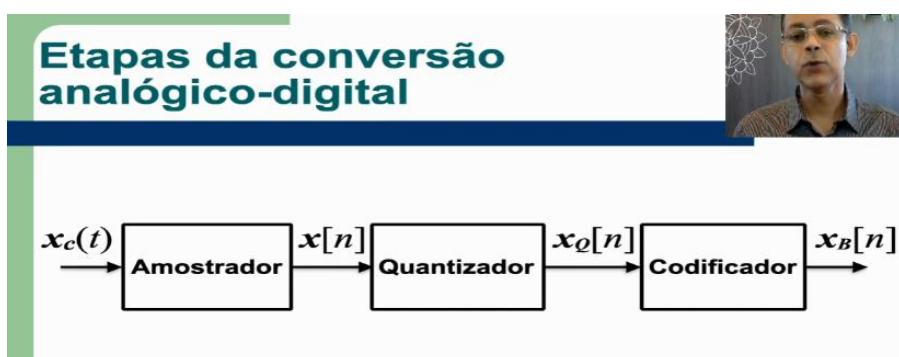
-Sonar: HC-SR04 (A TÉCNICA USADA POR SONARES COMO ESSE QUE SE UTILIZA DE ULTRASOM PARA MEDIR DISTÂNCIAS É BEM LIMITADA E TEM POUCA PRECISÃO, SENsores que utilizam da medição de distâncias através da emissão e recepção de um laser é bem mais precisa).

-Umidade: HIH-4000 Honeywell.

-Bússola: Honeywell HMC6352.

-MUITOS SENsores ENTREGAM SUA INFORMAÇÃO A SISTEMAS DE FORMA ANALÓGICA, assim é preciso que nosso microcontrolador receba tal informação analógica e transforme para digital para que o software possa utilizar, por isso é necessário o uso de CONVERSORES A/D ou ADC's:

- Etapas da conversão A/D:



- AMOSTRADOR:

- QUANTIZADOR: Faz o processo de DISCRETIZAÇÃO, ou seja, os valores analógicos podem estar dentro de uma certa faixa e ali eles podem assumir qualquer valor.

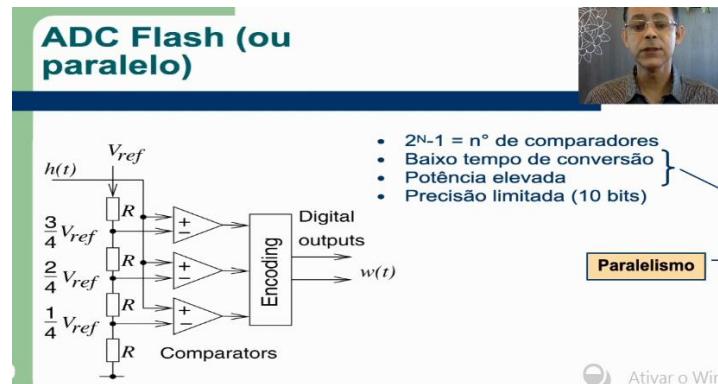
- APÓS O QUANTIZADOR IREMOS TER UM CONJUNTO DE VALORES PRÉ-DEFINIDOS PARA NOSSOS VALORES ANALÓGICOS.

- CODIFICADOR: Para cada valor analógico pré-estabelecido temos um CÓDIGO, ou seja, um NÚMERO BINÁRIO EQUIVALENTE.

- Na saída o ADC precisa entregar um número binário capaz de REPRESENTAR cada valor analógico passado na entrada!

- Existem muitas tecnologias variadas para se construir um conversor A/D ou ADC:

- ADC - FLASH(Ou PARALELO) – CONVERSÃO DIRETA:



Ativar o Wix

- Têm grande necessidade de utilizar de COMPARADORES DE TENSÃO para seu funcionamento.

- **PROBLEMA: O NÚMERO DE COMPARADORES CRESCE EXPLOSIVAMENTE A MEDIDA QUE SE ACRESCENTA UM BIT. COLOCA-SE UM BIT, DOBRA-SE O NÚMERO DE COMPARADORES PRATICAMENTE. ASSIM, TAL SISTEMA TORNA-SE INVÍAVEL PARA UM NÚMERO GRANDE DE BITS.**

- POTÊNCIA ELEVADA.

- SISTEMA POUCO ESCALÁVEL, por isso não é utilizado para quantidades de bits muito elevadas.

- Tem PRECISÃO LIMITADA(até 10 bits aproximadamente). PORQUE AO UTILIZAR-SE DE MUITOS COMPARADORES NO SISTEMA GERA UMA DIFICULDADE DE FABRICAÇÃO PORQUE TEORICAMENTE TODOS OS COMPARADORES DEVERIAM SER RIGOROSAMENTE IGUAIS PRA NÃO GERAR O PROBLEMA DE NÃO LINEARIDADE QUE É MOSTRADO ANTERIORMENTE NOS DEGRAUS DO CONVERSOR D/A, aquele mesmo fenômeno vai acontecer aqui provocado pelos diferentes comparadores de tensão que não vão ser completamente iguais e até os muitos resistores usados também que não serão da mesma forma fabricados de forma igual em tempo de fabricação.

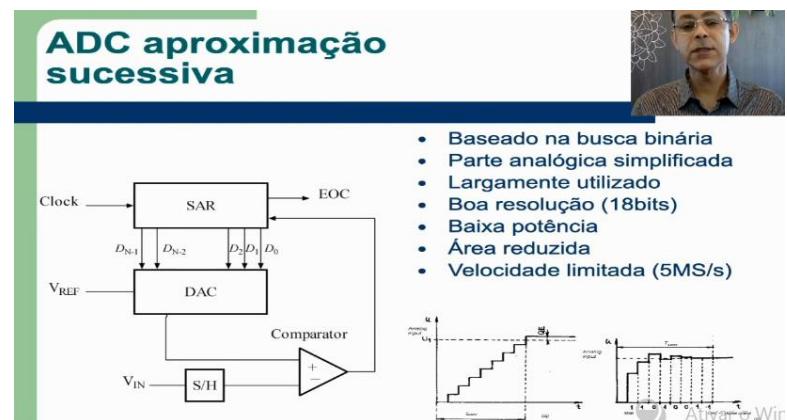
- **VANTAGENS: COMPARADO A OUTROS CONVERSORES A/D ESSE É MUITO RÁPIDO, SUA RESPOSTA É PRATICAMENTE INSTANTÂNEA.**

- **BAIXO TEMPO DE CONVERSÃO.** O conversor FLASH não possui nenhuma etapa para obter sua resposta (no ADC – FLASH o VALOR ANALÓGICO MUDAVA NA ENTRADA E NOSSO CONVERSOR JÁ TINHA TODOS OS SEUS COMPARADORES SENDO INFORMADOS DISSO JÁ ENTREGANDO O VALOR DIGITAL CORRESPONDENTE)

- Obs. **POTÊNCIA ELEVADA** e **BAIXO TEMPO DE CONVERSÃO/COMPUTAÇÃO** sempre acompanham estratégias de **SISTEMAS DE ALTO GRAU DE PARALELISMO**, característica

interessante para se ficar atento porque tais características ocorrem muito em outras áreas da computação.

### - ADC - APROXIMAÇÃO SUCESSIVA:



- Técnica MAIS UTILIZADA(POPULAR) NOS MICROCONTROLADORES HOJE EM DIA.

- Requer em seu sistema um CONVERSOR D/A (Ou DAC) para funcionar.
- SEU FUNCIONAMENTO BASEIA-SE FORTEMENTE NO MÉTODO DE BUSCA BINÁRIA.
- A estratégia de RESOLVER UM VALOR ANALÓGICO UTILIZANDO UM ALGORITMO TEM:

#### - VANTAGENS:

- Apesar de que QUANTO MAIS BITS, MAIS TEMPO SERÁ GASTO PARA FAZER A CONVERSÃO no ADC de aprox.sucessiva o resto do SEU SISTEMA SE MANTÉM CONSTANTE.

- SUA PARTE ANALÓGICA (QUE SÃO O SEU COMPARADO E SEU CONVERSOR D/A) É BEM MAIS SIMPLIFICADA DO QUE O CONVERSOR FLASH:

- No CONVERSOR FLASH pra cada bit acrescentado dobrava-se o número de comparadores, o sistema analógico de tal conversor AUMENTAVA BASTANTE DE COMPLEXIDADE.

- Já no conversor ADC de Aprox.Sucesiva ao acrescentar-se um bit não há a necessidade de mexer no número de comparadores(só precisda-se de apenas 1 comparador independente de quantos bits tiver o conversor). Isso torna a fabricação de tal conversor ADC MAIS SIMPLES, MAIS BARATA E MENOS SUJEITA A PROBLEMAS DE NÃO-LINEARIDADE COMO NO CASO DO conversor ADC-FLASH.

- Sua característica de SIMPLICIDADE DE CONSTRUÇÃO faz com que ele seja LARGAMENTE UTILIZADO.

- BAIXA POTÊNCIA (EXATAMENTE PELA SUA SIMPLICIDADE DECONSTRUÇÃO).

- ÁREA REDUZIDA DO SISTEMA (EXATAMENTE PELA SUA SIMPLICIDADE DECONSTRUÇÃO).

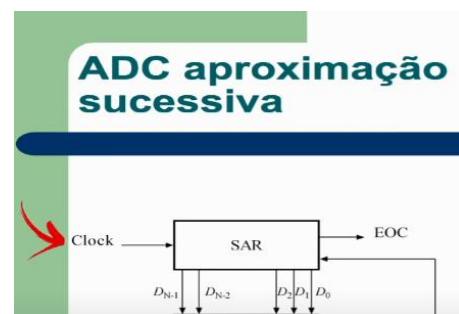
- VELOCIDADE LIMITADA (5MS/s)!

- CONSEGUE TAXAS DE AMOSTRAGEM BEM ELEVADAS(5MS/s - 5 MILHÕES DE VALORES POR SEGUNDO)!

- **DESAVANTAGENS:**

- Leva-se um certo tempo para resolver tal valor analógico, pois no ADC de APROX.SUCESSIVA há ETAPAS a se seguir para se encontrar a resposta esperada OU SEJA O SINAL DIGITAL, diferente do conversor FLASH que não possuia nenhuma etapa para obter sua resposta (no ADC – FLASH o VALOR ANALÓGICO MUDAVA NA ENTRADA E NOSSO CONVERSOR JÁ TINHA TODOS OS SEUS COMPARADORES SENDO INFORMADOS DISSO JÁ ENTREGANDO O VALOR DIGITAL CORRESPONDENTE).

- No ADC de APROX.SUCESSIVA há ETAPAS, ou seja, se escolhermos construir um conversor A/D de 8-bits teremos que fazer 1 PERGUNTA PARA CADA BIT pra descobrir se cada um deles é 1 ou 0 e dessa forma iremos precisar de 8 ETAPAS para resolver a nossa CONVERSÃO A/D. Cada ETAPA dessas é impulsionada por um CLOCK, na SAR, CADA CLOCK DESSE REGISTRADOR É O TEMPO PARA RESOLVER UM DOS BITS DO CONVERSOR:



“Então, se esse CLOCK demorar 1 milisegundo e nosso conversor ADC for de 8-bits iremos precisar de 1 milisegundo para resolver cada bit precisando de 8 miliegrandos para concluir a conversão do ADC de aprox.sucessiva ”

- QUANTO MAIS RÁPIDO O CLOCK MAIS RÁPIDA É A CONVERSÃO.

- SE O CLOCK FOR MANTIDO FIXO, A CADA BIT ACRESCENTADO IREMOS PRECISAR DE MAIS UM PULSO DE CLOCK PARA RESOLVER:

(\*)“SE TÍNHAMOS UM CONVERSOR DE 8-BITS PRECISÁVAMOS DE 8 MILISEGUNDOS PARA CONVERTER. SE QUISERMOS AUMENTAR A RESOLUÇÃO DO NOSSO CONVERSOR E PASSAR DE 8-BITS PARA 10-

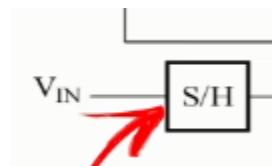
**BITS TERÍAMOS QUE GASTAR 10 MILISEGUNDOS PARA FAZER A CONVERSÃO E ASSIM POR DIANTE".**

**(\*)"QUANTO MAIS BITS, MAIS TEMPO SERÁ GASTO PARA FAZER A CONVERSÃO".**

**- A LIMITAÇÃO MAIOR DESSE CONVERSOR EM RELAÇÃO AO FLASH ESTÁ MESMO EMSUA VELOCIDADE. JÁ QUE PARA AUMENTAR A RESOLUÇÃO PRECISA-SE TORNAR O SISTEMA MAIS LENTO.**

**- EM CASO DE NÃO SE QUERER TAL SISTEMA COMO LENTO, HAVERÁ ENCARECIMENTO DO MESMO UMA VEZ QUE SE TERÁ DE UTILIZAR DE UMA OUTRA LOGICA MAIS RÁPIDA QUE A NORMALMENTE USADA. AINDA ASSIM TAL CONVERSOR CONSEGUE TAXAS DE AMOSTRAGEM BEM ELEVADAS!**

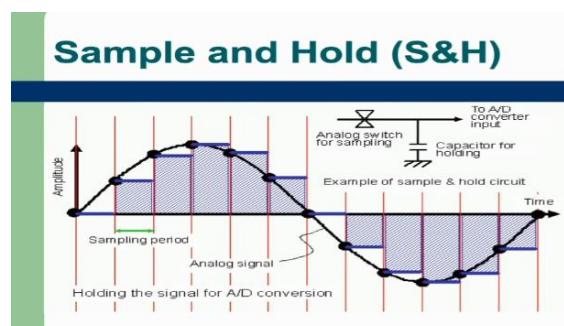
**- CIRCUITO S/H – “SAMPLE/HOLDING”:**



**- TAL CIRCUITO AJUDA A MANTER CONGELADA NA ENTRADA DO ADC ATENÇÃO ANALÓGICA. OU SEJA, ELE CONGELA A TENSÃO ANALÓGICA DURANTE TODO O TEMPO EM QUE A BUSCA BINÁRIA ESTÁ ACONTECENDO.**

**- GARANTE QUE O VALOR DE ENTRADA A SER DESCOBERTO PELO CONVERSOR NÃO SE ALTERE DURANTE A CONVERSÃO.**

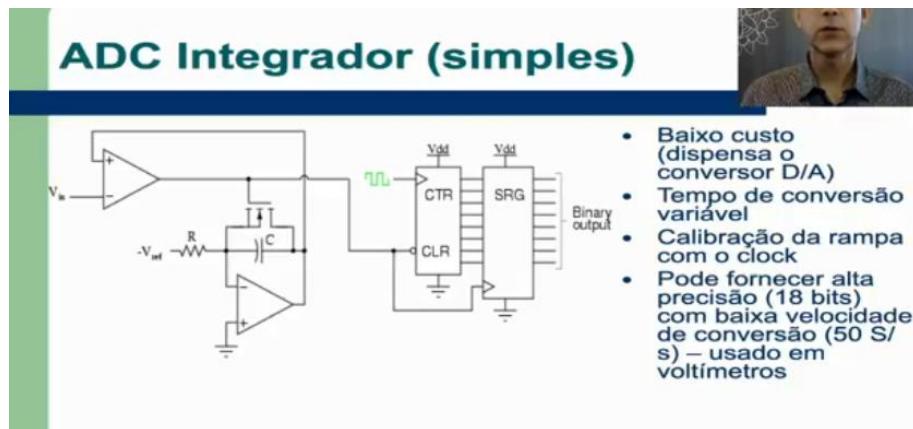
**- NOTE QUE O VALOR ANALÓGICO QUE UM CIRCUITO DESSES PRECISA TRABALHAR NÃO É GARANTIDO QUE ELE VAI FICAR PARADO NO TEMPO AFINAL É UM SINAL DE QUALQUER GRANDEZA, E ALGUMAS DESSAS GRANDEZAS CHEGAM A VARIAR MUITO RAPIDAMENTE SÓ QUE NÃO PODEMOS PERMITIR QUE A TENSÃO DE ENTRADA VARIE DURANTE O PROCESSO DE CONVERSÃO DO ADC, ENQUANTO O SAR ESTÁ FAZENDO A BUSCA BINÁRIA PRA DESCOBRIR QUAL O VALOR ANALÓGICO DE ENTRADA ESSE VALOR NÃO PODE SER ALTERADO ATÉ SER DESCOBERTO!(ESSE CENÁRIO É COMO SE NA BRINCADEIRA DO PENSE UM NÚMERO, NO MEIO DAS SUAS PERGUNTAS TENTANDO ADIVINHAR O NÚMERO PENSADO PELA PESSOA ELA MUDASSE ESSE NÚMERO).**



**- EXPLICAÇÃO MAIS TÉCNICA DE COMO FUNCIONA O S/H: "COLHE UMA AMOSTRA DA ENTRADA ATRAVÉS DE UMA CHAVE ANALÓGICA E CARREGA UM CAPACITOR COM ESSE VALOR, APARTIR DOMOMENTO QUE O CAPACITOR ESTÁ CARREGADO ELE ABRE A CHAVE E DEIXA TENSÃO CONGELADA DURANTE O TEMPO QUE O CONVERSOR PRECISAR, QUANDO A CONVERSÃO ACABAR**

O PRÓPRIO CONVERSOR JÁ PEDE UM NOVO VALOR E AO FAZER ISSO ELE AMOSTRA UMA NOVA ENTRADA, ASSIM, ELE VAI CAMINHANDO NAS LEITURAS SEM QUE A ENTRADA SEJA ALTERADA ENQUANTO A CONVERSÃO ACONTECE”.

- ADC – INTEGRADOR (SIMPLES):



- Há um circuito integrador dentro dele.
- Podemos compará-lo com o ADC de Aprox.Sucessiva.
- Pode DEMORAR BEM MAIS PARA ACHAR A RESPOSTA NO QUE NO ADC de Aprox.Sucessiva.
- BASEADONO SEU ALGORITMO PARA ACHAR A RESPOTA, NESSE ADC NÃO SE TEM COMO PREVER O TEMPO PARA ACHAR A RESPOSTA.
- NO ADC de Aprox.Sucessiva A QUANTIDADE DE “PERGUNTAS FEITAS PELO SISTEMA” É SEMPRE A MESMA QUE É O NÚMERO DE BITS DO CONVERSOR ADC, já no ADC INTEGRADOR (SIMPLES) só podemos garantir o pior caso (que é até 255) e o melhor caso que seria (0) mas não tem como garantir isso.

**\*ALGORITMO DO ADC - INTEGRADOR (SIMPLES):**

- **ESSENIALMENTE O QUE O CIRCUITO FAZ:** É GERAR UMA RAMPA E VAI COMPARAR O VALOR DESSA RAMPA COM O VALOR ANALÓGICO QUE TÁ COLOCADO NA ENTRADA. NO MOMENTO QUE A RAMPA SE ENCONTRAR COM O VALOR ANALÓGICO DA ENTRADA SABEREMOS QUE CHEGAMOS NO VALOR ANALÓGICO QUE ESTÁ SENDO COLOCADO PARA CONVERTERMOS. <<(por isso É TÃO DEMORADO E IMPREVISÍVEL!!!!)
- AO MESMO TEMPO EM QUE ESTAMOS GERANDO UMA RAMPA PARA COMPARAR COM O VALOR ANALÓGICO TEMOS UM CONTADOR CONTANDO DE ZERO ATÉ O NÚMERO BINÁRIO MÁXIMO QUE FOI ESTABELECIDO(DEPENDE DO NÚMERO DE BITS DO CONVERSOR: se for um conversor de 8-bits temos um contador contando de 0 até 255 ao mesmo tempo que a rampa está crescendo de 0 até o máximo valor analógico).
- QUANDO A RAMPA SE ENCONTRAR COM O VALOR ANALÓGICO O VALOR QUE TIVER NO CONTADOR É O VALOR DA COPNVERSÃO BINÁRIA DAQUELE NÚMERO(OU SEJA, A SAÍDA DO ADC).
- EXEMPLOS: MULTÍMETROS DIGITAIS SÃO GRANDES USUÁRIOS DE TAL TECNOLOGIA.

- VANTAGENS:

- **É UM CIRCUITO MUITO SIMPLES, DE FÁCIL FABRICAÇÃO.** TEM UM ALGORITMO BEM SIMPLES. ISSO O TORNA AINDA MAIS BARATO DE IMPLEMENTAR QUE O ADC DE APROX.SUCESSIVA E OP FLASH.

- BAIXO CUSTO(MAIS AINDA QUE O DE APROX.SUCESSIVA): SEU CIRCUITO NÃO PRECISA TER NELE DE UM DAC(CONVERSOR D/A) DIFERENTE NO ADC DE APROX.SUCESSIVA QUE PRECISAVA PARA SEU ALGORITMO DE BUSCA BINÁRIA ISSO
- FACIL DE FAZER PARA GRANDES QUANTIDADES DE BITS: BASTA AUMENTAR A QUANTIDADADE DE BITS QUE O CONTADOR TEM, E PRA FAZER ISSO NUM COTADOR ÉBEM FÁCIL(UMASIMPLES ADIÇÃO DE UM FLIP-FLOP AO SEU CIRCUITO JÁ BASTA).
- MESMO LENTO, É MUITO MAIS PRECISO QUE OS OUTROS ADC'S, E DEPENDENDO DA APLICAÇÃO TAL LENTIDÃO É ACEITÁVEL.

- DESAVANTAGENS:

- TEMPO DE CONVERSÃO VARIÁVEL QUE DEPENDE TOTALMENTE DO VALOR DE ENTRADA(NÃO DÁ PRA PREVER).
- MAIOR A QUANTIDADE DE BITS,MAIS LENTO ELE TENDE A SER!
- MAS: SE TORNA UMA OPÇÃO MUITO ATRATIVA DE CUSTO E FABRICAÇÃO PARA SISTEMAS ONDE NÃO HÁ PRESSA NA OBTENÇÃO DE RESPOSTAS E PRECISA-SE DE BOA PRECISÃO.

$$t_{conv_{MAX}} = \frac{2^{bits} - 1}{f_{clk}}$$

- EXEMPLOS PRÁTICOS: ADC DO KIT SPARTAN-3E 500/1600, ADC do dsPIC30F.

- ERRO de aliasing:



- TODA VEZ QUE UM CONVERSOR A/D ESTÁ EM OPERAÇÃO ELE ESTÁ CAPTURANDO DADOS DO MUNDO ANALÓGICO EM UM CERTO INTERVALO DE TEMPO.

- INTERVALO DE AMOSTRAGEM: "INTERVALO ENTRE UMA CAPTURA E OUTRA".

- DEFINE A TAXA DE AMOSTRAGEM: "QUANTAS AMOSTRAS POR SEGUNDO AQUELE SISTEMA É CAPAZ DE FAZER".

- OS SINAIS DO MUNDO ANALÓGICO (AS ONDAS QUE ESTÃO LÁ PARA SEREM CAPTURADA PELO SOFTWARE) ESTÃO VARIANDO NO TEMPO E NOSSO SISTEMA DE CAPTURA DE DADOS PRECISA SER CAPAZ DE OBTER DADOS NUMA VELOCIDADE ADEQUADA PRA QUE ELE NÃO PERCA INFORMAÇÕES IMPORTANTES COMPROMETENDO A INTERPRETAÇÃO QUE SEU SOFTWARE VAI DAR PARA AQUELES DADOS.

## Erro de aliasing

SOLUÇÃO MATEMÁTICA

- Baixa taxa de amostragem
- Critério de Nyquist ( $f_s = 2 \cdot f_{max}$ )

SE HOUVESSEM MAIS PONTOS COLETADOS ISSO TERIA SIDO EVITADO!

motivo do erro de interpretação das duas ondas onda desejada

onda que o software vai interpretar

- MOTIVOS DE ERROS DE ALIASING: MUITAS VEZES POR BAIXAS TAXAS DE AMOSTRAGEM.

- SOLUÇÕES: HÁ UMA FÓRMULA MATEMÁTICA-> [CRITÉRIO DE NYST.](#) SABENDO MINHA FREQUENCIA DE ENTRADA EU ESTABELEÇO QUANTAS CONVERSÕES TENHO QUE FAZER(CAPTURAS DE PONTOS NA FREQUENCIA) POR UNIDADE DE TEMPO PRA NÃO COMETER ESSE ERRO.

- ESSE CRITÉRIO VAI SER MUITO IMPORTANTE QUANDO TIVERMOS UMA APLICAÇÃO QUE CAPTURA SINAIS E PRECISA TOMAR DECISÕES EM RELAÇÃO A ESSE SINAIS(MUITOS SISTEMAS EMBARCADOS USAM DISSO! USAR ADEQUADAMENTE ESSE CRITÉRIO PODE SER DIFERENCIAL NO FUNCIONAMENTO DE SUA APLICAÇÃO.)

- RECAPITULANDO – CONCEITOS – CONVERSOR A/D:

## Conversor A/D - Conceitos

- Resolução – O número de valores discretos que podem ser produzidos
  - Exemplo: resolução de 8 bits => 256 valores possíveis
- Erro de quantização – Se deve à resolução finita de um ADC
- Tempo de resposta – tempo gasto para completar o processo de conversão
- Taxa de amostragem – Quantidade de amostras coletadas de um sinal analógico em uma determinada unidade de tempo
- Taxa de amostragem – Inverso do intervalo de tempo entre uma amostragem e outra

- RESOLUÇÃO:

- ASSOCIADA AO NÚMERO DE BITS DO CONVERSOR SEJA ELE A/D OU D/A.

- SE TEMOS UM CONVERSOR DE “8-BITS” É PORQUE SUA RESOLUÇÃO É DE 8-BITS TAMBÉM ALÉM DE QUE O MESMO TAMBÉM FAZ SUA “QUANTIZAÇÃO” EM 256 VALORES (A FAIXA DE TENSÃO ANALÓGICA QUE ELE TRABALHA FOI SEGMENTADA EM 256 FAIXAS E PRA CADA UM DESSES VALORES TEREMOS UM NÚMERO BINÁRIO EQUIVALENTE).

- ERRO DE QUANTIZAÇÃO:

- É IMPOSSÍVEL ZERAR ESSE ERRO PORQUE SEMPRE IREMOS TER UM NÚMERO FINITO DE BITS PARA O CONVERSOR SIGNIFICANDO QUE ESSE ERRO SEMPRE VAI ESTAR PRESENTE POR MENOR QUE SEJA.

- TODA VEZ QUE O CONVERSOR FAZ ESSA “SEGMENTAÇÃO”/“QUANTIZAÇÃO” ELE IRÁ PROVOCAR UM CERTO ARREDONDAMENTO. SE ESTABELECERMOS QUE O INTERVALO ENTRE OS VALORES ANALÓGICOS É DE 0,22 VOLTS POR EXEMPLO O NÚMERO BINÁRIO ZERO VAI SER EQUIVALENTE A TENSÃO 0 O NÚMERO BINÁRIO 1 VAI SER EQUIVALENTE ATÉNSÃO 0,22 VOLTS SIGNIFICANDO QUE QUASQUER TENSÕES ENTRE 0 E 0,22 VOLTS VÃO SER ARREDONDADAS PARA UM DESTES VALORES E ESSE ARREDONDAMENTO É UM ERRO, CHAMDO DE ERRO DE QUANTIZAÇÃO: **“QUANTO MAIS BITS TIVER MAIOR SERÁ A RESOLUÇÃO DO CONVERSOR E MENOR SERÁ O SEU ERRO DE QUANTIZAÇÃO”.**

- TAXA DE AMOSTRAGEM:

- **DEFINIÇÃO-01:** QUANTIDADE DE AMOSTRAS COLETADAS DE UM SINAL ANALÓGICO EM UMA DETERMINADA UNIDADE DE TEMPO(***TEMPO ENTRE UMA CONVERSÃO E OUTRA, SE CONSEGUIMOS CAPTURAR(CONVERTER A/D) 1000 AMOSTRAS EMUM SEGUNDO,TEMOS UMA TAXA DE 1000 AMOSTRAS POR SEGUNDO.***)

- **DEFINIÇÃO-02:** INVERSO DO INTERVALO DE TEMPO ENTRE UMA AMOSTRAGEM E OUTRA(***SE SUA AMOSTRAGEM É DE 1 MILISEGUNDOS SUA TAXA É DE 1000 AMOSTRAS POR SEGUNDO.***)

- TEMPO DE RESPOSTA:

- Todo conversor a/d E d/a tem que lidar com tempos finitos de resposta(Tempo que o sistema leva ou para produzir um valor analógico ou para reconhecer um novo valor analógico e transformá-lo para binário).

- É GASTO NO PROCESSO DE COMPLETAR A CONVERSÃO(Mas se pensarmos na aplicação como um todo existe a parte do microcontrolador pra tratar essa informação ou pra gerar essa informação se for um processo de saída de dados).

- **ESTÁ DENTRO DA IDEIA DE TAXA DE AMOSTRAGEM!**

//////////

**\*SOFTWARE EMBARCADOS:**

- ENTRE NOSSA APLICAÇÃO E O HARDWARE EXISTEM VÁRIOS OUTROS RECURSOS DE SOFTWARE QUE DÃO SUPORTE A ELA, O MAIS CONHECIDO DELES É O **SO(SISTEMA OPERACIONAL)**.

- INTRODUÇÃO:

- HOJE, SEMPRE AO COMEÇAR PROJETOS DE DESENVOLVIMENTO DE SOFTWARES PARTIMOS JÁ DE OUTROS PRÉ-EXISTENTES.NUNCA SE COMEÇA UMA APLICAÇÃO DE SOFTWARE REALMENTE DO ZERO, IREMOS SEMPRE REAPROVEITAR OUTROS SOFTWARES, SEJAM PARTES DA APLICAÇÃO OU OUTROS RECURSOS DISPONÍVEIS.

- **REUSO** DE PROPRIEDADE INTELECTUAL (IP – “INTELLECTUAL PROPERTY”): REAPROVEITAMENTO DE OUTROS RECURSOS DE SOFTWARE.

- ALGUMAS FORMAS DE REUSO DE SOFTWARE:

- BIBLIOTECAS.

- SISTEMA OPERACIONAL.

- BANCO DE DADOS.

- MIDDLEWARE (**Intermediário entre aplicação e SO**).

- **SOBRE O CUSTO DO DESENVOLVIMENTO DE SOFTWARES:**

- GERALMENTE DIZ-SE QUE O SOFTWARE SEMPRE É A PARTE MAIS CARA DO DESENVOLVIMENTO DE UM PRODUTO. E É VERDADE, O FATO SECONPROVA PELOS PREÇOS ABSURDOS QUE SE CHEGA A COBRAR POR LINHAS DE CÓDIGO.VEJA:

**Sobre custo do código**

- The cost of firmware at \$20 to \$40 (Per line of code - LOC)
- The Space Shuttle code cost \$1000/LOC.

- OBS. O SOFTWARE QUE VEM NUM DISPOSITIVO EMBARCADO É COMUMENTE CHAMADO NO MERCADO DE FIRMWARE.

- **PLATAFORMA:**

- QUANDO VAMOS DESENVOLVER UMA APLICAÇÃO EMBARCADA TRABALHAMOS EM CIMA DE UMA PLATAFORMA.

- É ALGO QUE SERVE DE BASE PARA DESENVOLVER UMA APLICAÇÃO EMBARCADA.

- OFERECEM VÁRIOS RECURSOS PARA QUE SEUS DESENVOLVEDORES CONSIGAM UM TEMPO DE PROJETO MAIS CURTO. DENTRE ESSES RECURSOS OFERECIDOS TEMOS:

- IDE(COMPILADOR,DEPURADOR).
- PILHA DE PROTOCOLOS (TCP/IP/ETH, USB, BLUETOOTH).
- PRIMITIVAS DE ACESSO AO HARDWARE.
- SISTEMA OPERACIONAL.
- PROGRAMAS EXEMPLO (CÓDIGOS PRONTOS QUE IMPLEMENTAM VÁRIAS DAS FUNÇÕES DE DETERMINADA PLATAFORMA).

- VEJA A SEGUINTE DEFINIÇÃO:

**Plataforma?**

- Uma plataforma é uma família de arquiteturas que satisfazem a um conjunto de restrições impostas para o reuso de componentes de hardware e de software.

- **ANÁLISE DA DEFINIÇÃO ACIMA:**

- **FAMÍLIA DE ARQUITETURAS:** ARQUITETURA É UMA PALAVRA PRA DEFINIR CONJUNTO DE INSTRUÇÕES DE PROCESSADORES(ENTÃO AO FALAR ARQUITETURA É QUASE COMO ESTAR FALANDO A PALAVRA “PROCESSADOR” EM SI). FAMÍLIA DE ARQUITETURAS SE REFERE ENTÃO AS FAMÍLIAS DE PROCESSADORES.

- **REUSO:** TODA A IDEIA DE UMA PLATAFORMA GIRA EM TORNO DO REUSO,NÃO SÓ REUSO DE HARDWARE OU SÓ DE SOFTWARE MAS DAS DUAS COISAS TAMBÉM.

- PLATAFORMAS PROCURAM CRIAR CONDIÇÕES PARA FACILITAR ESSE REUSO DOS SEUS RECURSOS DISPONÍVEIS.

- **FACILIDADE DE RE-USO DE SOFTWARE:** VOCÊ DESENVOLVE UMA APLICAÇÃO PARA UM CERTO PROCESSADOR E AMANHÃ TALVEZ POSSA RE-USAR UMA PARTE DESSE SOFTWARE EM UMA NOVA APLICAÇÃO E TAL NOVA APLICAÇÃO PODERÁ FUNCIONAR EM UMA OUTRA ARQUITETURA DA MESMA FAMÍLIA DADO QUE AS FERRAMENTAS QUE O FABRICANTE DA PLATAFORMA OFERECE SE INTEGRAM ENTRE SI PODENDO POSSIBILITAR AO CÓDIGO MIGRAR DE UMA ARQUITETURA PARA OUTRA SEM HAVER GRANDE ESFORÇO DE REDESENHO DE PROJETO.

- **FACILIDADE DE RE-USO DE HARDWARE:** TUDO QUE É CONSTRUÍDO PARA SE CONECTAR NOS PROCESSADORES FACILMENTE SE CONECTA EM QUALQUER UM DOS PROCESSADORES DA FAMÍLIA E CADA VEZ QUE SE É CRIADO UM PROJETO NOVO É POSSÍVEL RÉ-APROVEITAR O HARDWARE DO PROJETO ANTERIOR ACRESCENTANDO NOVOS COMPONENTES OU TIRANDO PARTE DAQUELES COMPONENTES QUE NÃO VÃO SER NECESSÁRIOS NESSE NOVO PROJETO.

- Entretanto, uma plataforma de hardware não é suficiente. Projetos rápidos, confiáveis e evolutivos demandam o uso de Interfaces de Desenvolvimento de Aplicações (API) para estender a plataforma em direção ao software de aplicação.

- **ANÁLISE DA DEFINIÇÃO ACIMA:**

- **API:** PROCURA DEIXAR TUDO PRONTO PRA QUE O DESENVOLVEDOR DE APLICAÇÕES CONSIGA TERMINAR SEU PROJETO NO MENOR TEMPO POSSÍVEL.

- **HÁ UMA PREOCUPAÇÃO EM REUSO QUE EM ÚLTIMA ANÁLISE É UMA PROCURAÇÃO EM DIMINUIR O TIME-TO-MARKET QUE É UMA DAS GRANDES PRIORIDADES QUANDO SE CONSTRÓI UM PRODUTO EMBARCADO.**

Em geral, uma plataforma é um processo de abstração que admite muitos refinamentos até chegar ao nível mais baixo.

- **ANÁLISE DA DEFINIÇÃO ACIMA:**

- **REFINAMENTO DE NÍVEIS DE ABSTRAÇÃO:** FALAMOS DE PLATAFORMA EM VÁRIAS CAMADAS DE ABSTRAÇÃO DE UM PRODUTO.

- SE OLHARMOS PARA UM PROCESSADOR PODEMOS CONSIDERÁ-LO TAMBÉM COMO UMA PLATAFORMA PARA DESENVOLVIMENTO DE UMPRODUTO.



- SE OLHARMOS PARA O CONJUNTO LINGUAGEM DE PROGRAMAÇÃO E COMPILADORES SOPRE UM PROCESSADOR ISSO TAMBÉM É UMA PLATAFORMA SÓ APENAS O NÍVEL DE ABSTRAÇÃO QUE ESTÁ UM POUCO MAIS ALTO.



- SE PENSARMOS AINDA NUM SO QUE FUNCIONA SOB UM CERTO PROCESSADOR JÁ COM UMCONJUNTO DE BIBLIOTECAS COM VÁRIAS FUNÇÕES IMPLEMENTADAS TAMBÉM TEMOS AÍ UMAPLATAFORMA SÓ QUE COM UM NÍVEL DE ABSTRAÇÃO AINDA MAIS ALTO.

**“ESSAS VÁRIAS CAMADAS DE ABSTRAÇÃO TODAS SÃO ACEITAS COMO DEFINIÇÃO DE PLATAFORMA”.**

**- SISTEMAS OPERACIONAIS:**

**-PORQUE USAR UM SO?**

**- É O RECURSO DE SOFTWARE MAIS UTILIZADO COMO PLATAFORMA PARA MUITAS APLICAÇÕES.**

**- INTERFACE ENTRE HARDWARE E SOFTWARE:**

**1- PRIMEIROS MODELOS DE INTERFACE :**

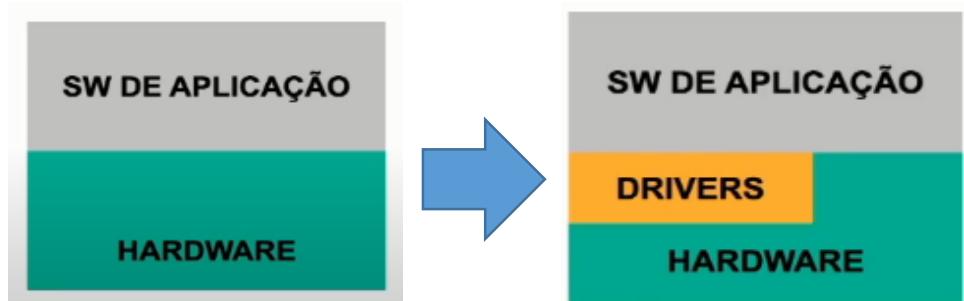


**- AS PRIMEIRAS APLICAÇÕES EMBARCADAS DESENVOLVIDAS ERAVAM MUITO SIMPLES, BASICAMENTE O QUE ERA OFERECIDO ERA UM HARDWARE QUE IMPLEMENTAVA ALGUM TIPO DE SERVIÇO NECESSÁRIO E A CAMADA DE SOFTWARE SOB ELE ERA MUITO BÁSICA (COM MUITO POUCAS LINHAS DE CÓDIGO E MUITAS VEZES EM LINGUAGEM DE MÁQUINA DO PRÓPRIO PROCESSADOR).**

**- MAIOR PARTE DO ESFORÇO DE ENGENHARIA PARA DESENVOLVER AQUELE PRODUTO ESTAVA NO HARDWARE.**

**- MESMA EQUIPE QUE DESENVOLVIA TODO O PRODUTO (OS MESMOS ENGENHEIROS QUE FAZIAM O PROJETO DO HARDWARE DE SUAS PLACAS E SUAS MONTAGENS ERAVAM OS MESMOS QUE FAZIAM SEU SOFTWARE DE APLICAÇÃO).**

**2- SEGUNDOS MODELOS DE INTERFACE :**



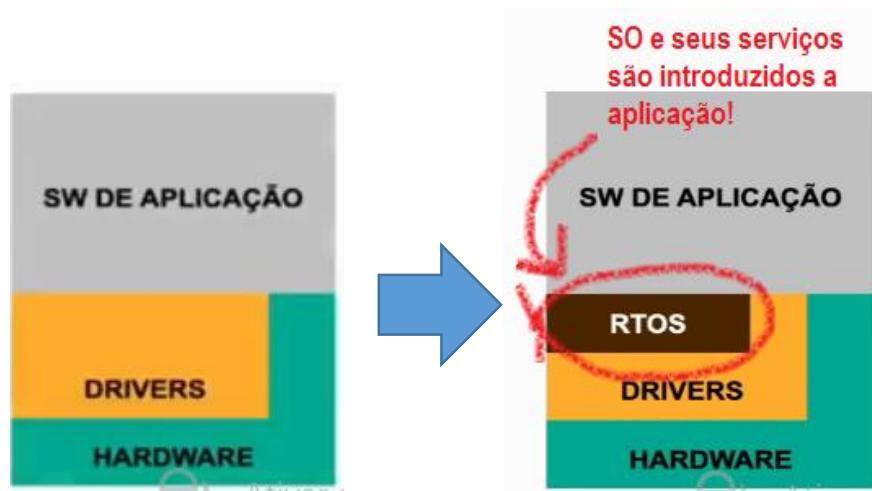
**- AS APLICAÇÕES EMBARCADAS DESENVOLVIDAS FICAVAM MAIS SOFISTICADAS, NECESSITAVAM DE MAIS LINHAS DE CÓDIGOS E O ESFORÇO DE PROGRAMAÇÃO DA APLICAÇÃO FOI AUMENTANDO.**

- NESSE PONTO SE NOTA UMA DIVISÃO NA EQUIPE DE PRODUÇÃO DA INTERFACE DO PRODUTO, UMA PARA DESENVOLVER O HARDWARE E OUTRA O SOFTWARE.

- **DRIVERS:** UMA CAMADA DE ABSTRAÇÃO ENTRE AS DUAS EQUIPES É DEFINIDA PARA QUE POSSAM TRABALHAR DE MANEIRA INDEPENDENTE .

\***DRIVERS:** SÃO UM CONJUNTO DE PRIMITIVOS QUE ACESSAM O HARDWARE DE DETERMINADO PRODUTO.

### 3- TERCEIROS MODELOS DE INTERFACE - ATUAIS:



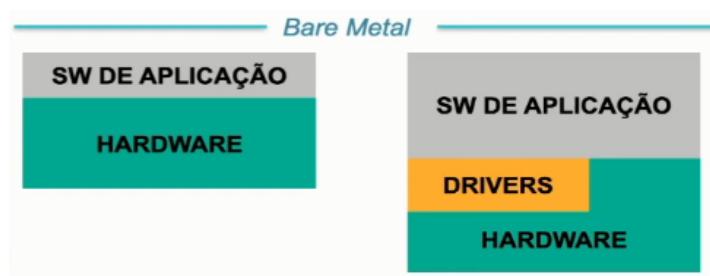
- AS APLICAÇÕES EMBARCADAS DESENVOLVIDAS FICAM AINDA E AINDA MAIS SOFISTICADAS E O ESPAÇO OCUPADO PARA O DESENVOLVIMENTO DO SOFTWARE DE APLICAÇÃO NO PRODUTO FICA BEM MAIOR E MAIS COMPLEXO DO QUE ANTES, ATÉ MESMO MAIS QUE O DO DESENVOLVIMENTO DO HARDWARE E DOS DRIVERS JUNTOS. COM ISSO ESSAS APLICAÇÕES PRECISAVAM DE MAIS E NOVOS RECURSOS PARA SEREM GERENCIADAS E DE NOVAS E MAIORES EQUIPES PARA DESENVOLVÊ-LAS .

- **SOLUÇÕES:** UMA DAS COISAS QUE PODE AJUDAR A AUMENTAR A PRODUTIVIDADE DESSA EQUIPE DE SOFTWARE É A DISCIPLINA IMPOSTA POR UM SISTEMA OPERACIONAL:

- CADA SO TEM SUAS REGRAS PRA DEFINIR OS MECANISMOS DE ACESSO AO HARDWARE ESSAS REGRAS VÃO IMPOR ALGUNS PADRÕES PARA O DESENVOLVIMENTO DA APLICAÇÃO QUE VAI FACILITAR O INTERCÂMBIO DE COMPONENTES DE SOFTWARE ENTRE VÁRIAS APLICAÇÕES. AGORA VAI SER POSSÍVEL A TROCA DE COMPONENTES ENTRE SOFTWARES FEITOS PARA UM MESMO SO EM OUTRAS PLATAFORMAS DE HARDWARE POR EXEMPLO.

\***OBS:** NOTE QUE O "SO" USADO NA APLICAÇÃO É UM "RTOS"!

- **Bare Metal** - INTERFACES:



- O QUE É: Quando desenvolve-se uma aplicação embarcada diretamente sobre os recursos de hardware ou sobre algum tipo de API oferecida pelo fabricante de hardware, diz-se que aquela aplicação foi escrita em “Bare metal”, significando dizer que não há um SO entre a aplicação e o hardware , ou, que a aplicação vai direto ao recurso mais básico de hardware.

- Usa-se da estratégia de desenvolvimento “Bare Metal” para se obter melhor desempenho na aplicação já que não havendo uma camada intermediária é possível se tirar menor tempo de computação por exemplo.

- Também pode-se usar dessa estratégia por questão de simplicidade, em caso de uma aplicação que não precise necessitar dos serviços típicos oferecidos por sistemas operacionais. Então, acaba-se por não introduzir um SO em determinado conjunto por questões de redução de custos ou de economia de recursos de hardware mesmo. Um SO tipicamente oferece: Gerenciamento de memória, escalonamento de tarefas, gerenciamento de sistema de arquivos. E pode ser que determinada aplicação não precise de nenhuma dessas coisas ou até precise mas somente de UMA delas, mas aí pode-se encontrar tal serviço em bibliotecas por exemplo e não necessariamente em um SO!

- **RTOS(SISTEMAS OPERACIONAIS DE TEMPO REAL):**

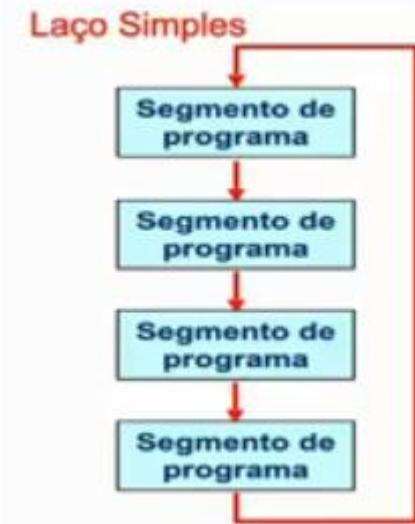
- É MUITO COMUM QUE APLICAÇÕES EMBARCADAS PRECISEM ATENDER A RESTRIÇÕES DE TEMPO, POR ISSO ELAS SERIAM APLICAÇÕES DE TEMPO REAL(DAÍ O USO DE RTOS). **ISSO NÃO VAI ACONTECER SEMPRE EM TODOS OS CASOS MAS EM MUITAS LITERATURAS É COMUM ENCONTRAR RTOS SE REFERINDO A SISTEMA OPERACIONAL PARA APLICAÇÕES EMBARCADAS.**

- **ORGANIZAÇÃO DAS CAMADAS NUMA APLICAÇÃO:** É MUITO COMUM ORGANIZAR ESSA ESTRUTURA DE CAMADAS PARTINDO DA APLICAÇÃO E CHEGANDO ATÉ O HARDWARE. SENDO QUE ENTRE A APLICAÇÃO E O HARDWARE PASSA-SE PRIMEIRO PELO SO E DEPOIS PELOS DRIVERS, É ISSO QUE NOTAMOS POR EXEMPLO NA ORGANIZAÇÃO DAS CAMADAS DOS NOSSOS COMPUTADORES(PC) POR EXEMPLO.

- **ORGANIZAÇÃO DAS CAMADAS NUMA APLICAÇÃO EMBARCADA (USADA COM FREQUÊNCIA):** O SO NÃO INCLUI OS DRIVERS, OS DRIVERS SÃO COLOCADOS SOBRE O KERNEL DO SO, ISSO ACONTECE PORQUE PARA QUEM DESENVOLVE UM SO É DIFÍCIL SABER APRIORI QUAIS SERÃO OS HARDWARES QUE TAL APLICAÇÃO IRÁ UTILIZAR, QUE TIPOS DE DRIVERS ELA PRECISA CONTER, ASSIM, AO INVÉS DE ENCAPSULAR ESSES DRIVERS DENTRO DE UM SO ELES SÃO COLOCADOS DEPOIS, POR CIMA DO SO.

## - MODELOS DE PROGRAMAÇÃO

### - LAÇO SIMPLES:



- ATRAVÉS DE UM LAÇO SIMPLES, CADA UM DOS SERVIÇOS QUE PRECISA SER ATENDIDO É UM BLOCO DE CÓDIGO, E ELES SÃO EXECUTADOS EM SEQUÊNCIA CONTINUA(OS SERVIÇOS), ENQUANTO O DISPOSITIVO CHECA SE HOUVE ENTRADA NAS INTERFACES PELO USUÁRIO, SE HOUVER O CÓDIGO TOMARÁ ALGUMA ATITUDE. EX: **PORTEIROS ELETRÔNICOS QUE ABREM COM IMPRESSÃO DIGITAL/SENHA.**

- **LIMITAÇÃO: ESSES BLOCOS NÃO PODEM SER SENSÍVEIS A ATRASOS.**

- **EXEMPLO:** "SE UM USUÁRIO CHEGA NUM PORTEIRO ELETRÔNICO PRA DIGITAR UMA SENHA E NO EXATO MOMENTO EM QUE ESTE VAI TECLAR SUA SENHA O SOFTWARE ESTIVER SE COMUNICANDO COM OUTRA INTERFACE NO MOMENTO(DIGAMOS UM INTERFACE ETHERNET) ESSE SOFTWARE NÃO VAI PODER RECONHECER QUE A TECLA FOI PRESSIONADA MAIS LOGO DEPOIS DE UM CURTO TEMPO O SOFTWARE JÁ CHEGA PRA LER O TECLADO" << ESSE PEQUENO DELAY NA LEITURA DO TECLADO NÃO É PERCEBIDO PELO USUÁRIO JÁ QUE O MESMO NÃO CONSEGUE PERCEBER O FATO DE "APERTAR UM BOTÃO E 1 DÉCIMO DE SEGUNDO DEPOIS O PROCESSADOR DA APLICAÇÃO CHEGA E RECONHECE A TECLA PRESSIONADA", PARA APLICAÇÕES COMO A DESSE EXEMPLO A ESTRATÉGIA DE LAÇO SIMPLES É MUITO BOA, PORQUE CADA TRECHO DE CÓDIGO DESSE PODE SOFRER ATRASOS SEM PROBLEMAS DE MAL-FUNCIONAMENTO E DESCONFORTO PARA O USUÁRIO.

### - VANTAGENS:

- MUITO BOA PARA APLICAÇÕES EM QUE A RESPOSTA DE TEMPO PODE SER UM POUCO ATRASADA. CADA TRECHO DE CÓDIGO PODE SOFRER ATRASOS SEM PROBLEMAS DE MAL-FUNCIONAMENTO E DESCONFORTO PARA O USUÁRIO.

- **DESAVANTAGENS:**

- APLICAÇÕES QUE EXIGEM RESPOSTAS UM POUCO MAIS RÁPIDAS(AO PISAR NUM FREIO ENQUANTO DIRIGIMOS UM AUTOMÓVEL POR EXEMPLO, DE FORMA ALGUMA DEVE HAVER ATRASO NO PROCESSAMENTO DE RESPOSTAS NO SISTEMA EXTENDENDO O TEMPO SOB PENA DE FALHAS NA APLICAÇÃO).

- SITUAÇÕES ONDE EXISTE UMA RESTRIÇÃO DE TEMPO DE RESPOSTA EM CADA UM DOS SEGMENTOS DE PROGRAMA. **POIS NÃO TEMOS COMO GARANTIR O TEMPO QUE VAI LEVAR PARA UM CERTO SERVIÇO SER EXECUTADO.**

- **LAÇO COMBINADO COM SERVIÇO DE INTERRUPÇÃO(ISR):**



- CADA TRECHO DE CÓDIGO ESTÁ ASSOCIADO A UMA INTERRUPÇÃO, QUANDO UMA DETERMINADA INTERFACE RECEBE UM PEDIDO(MENSAGEM/ALGUÉM TOCAR UM TECLADO/COLOCAR O DEDO NUM LEITOR DE IMPRESSÃO DIGITAL) ISSO DISPARA UMA INTERRUPÇÃO QUE VAI ACIONAR O PROCESSADOR PARA EXECUTAR AQUELE SEGMENTO DE PROGRAMA.

- (**DESvantagem**)**OS PRÓPRIOS DESENVOLVEDORES TEM QUE GERENCIAR O USO DO SISTEMA DE INTERRUPÇÃO:** ACABA POR DIMINUIR UM POCO A PRODUTIVIDADE DO TIME DE DESENVOLVIMENTO .

- **DIFERENÇAS ENTRE ESSE MODELO E O DE LAÇO SIMPLES:**

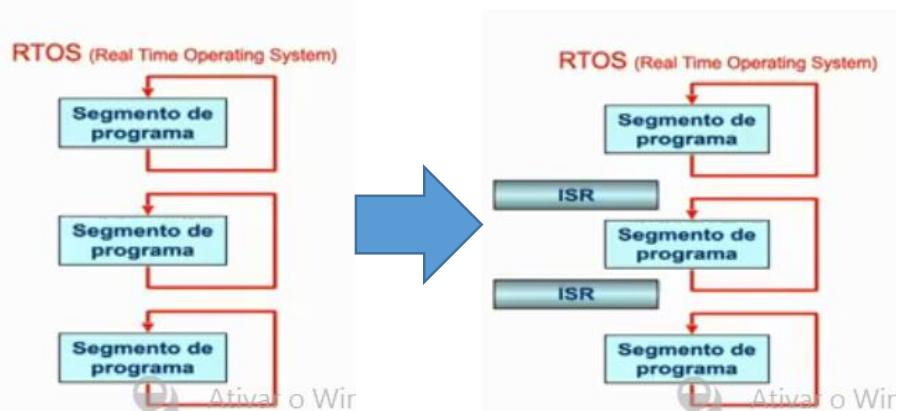
- NO MODELO CONTROLADO POR INTERRUPÇÃO OS SEGMENTOS DE PROGRAMA PODEM SER OS MESMOS SÓ QUE O PROCESSADOR NÃO FICA EXECUTANDO ELES EM SEQUÊNCIA(ENQUANTO NO LAÇO SIMPLES ELES FICAM EM SEQUÊNCIA CONTINUA), ELE VAI FICAR NUM “LAÇO PRINCIPAL” (QUE TENDE A SER BEM SIMPLES COMPUTACIONALMENTE FALANDO) E CADA NECESSIDADE DE SEGMENTO DE PROGRAMA VAI SER DISPARADA MEDIANTE A SOLICITAÇÃO DO HARDWARE. **ISSO AJUDA A DIMINUIR O TEMPO DE RESPOSTA ENTRE A CHEGADA DO**

PEDIDO NO HARDWARE E A EXECUÇÃO DO SOFTWARE QUE ATENDE AQUELA NECESSIDADE.

\*OBS. - ESSENCIALMENTE AMBOS OS MODELOS DE LAÇO SIMPLES E O CONTROLADO POR INTERRUPÇÃO SÃO ADEQUADOS PARA UMA PEQUENA QUANTIDADE DE SEGMENTOS DE PROGRAMAS.

- AMBOS OS DOIS MODELOS TÊM DIFICULDADES QUANDO QUER-SE DIVIDIR TAIS SEGMENTOS DE PROGRAMA ENTRE UMA EQUIPE DE PROGRAMADORES PORQUE ELES VÃO PRECISAR ESTAR MUITO INTEGRADOS PRA QUE O QUE CADA UM FAZ NÃO ATRAPALHE O QUE O OUTRO ESTÁ FAZENDO.

- RTOS(REAL TIMEOPERATING SYSTEM):



- É UM TERCEIRO TIPO DE MODELO USADO: PARA MELHORAR O GERENCIAMENTO DE UMA EQUIPE DE DESENVOLVIMENTO OU QUANDO SE TÊM UMA QUANTIDADE MAIOR DE SEGMENTOS DE PROGRAMA.

- DIVIDIMOS AS TAREFAS UTILIZANDO OS RECURSOS DO SO(OU MELHOR DIZENDO:RTOS).

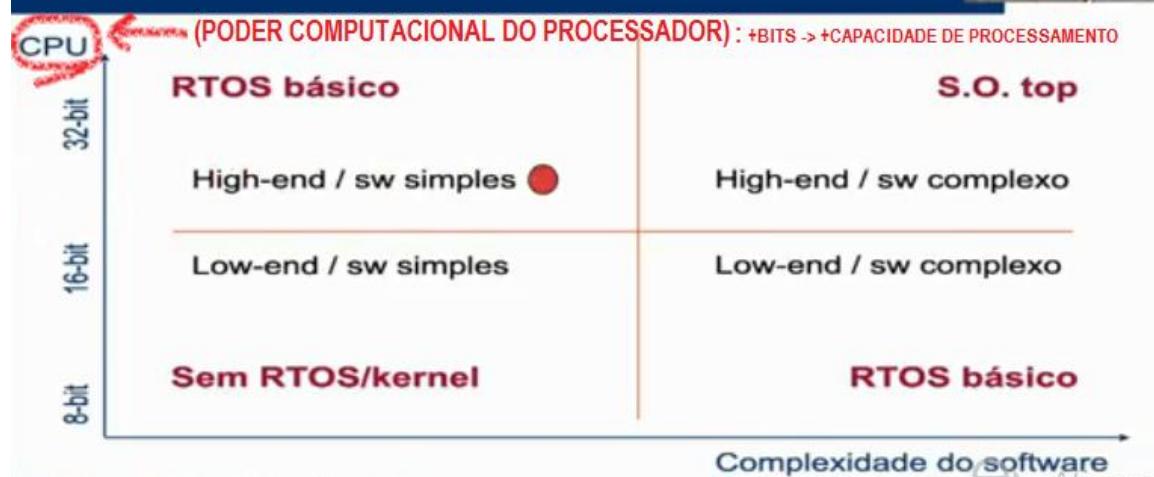
- AGORA CADA SEGMENTO DE PROGRAMA DESSE PODE SER UMA TAREFA OU UM PROCESSO E SUAS TROCAS(AS TROCAS ENTRE ESSAS TAREFAS) VÃO SER FEITAS PELO SISTEMA OPERACIONAL RTOS ONDE O SO IRÁ FAZER ISSO UTILIZANDO O SISTEMA DE INTERRUPÇÃO QUE OCORRE DE MANEIRA “TRANSPARENTE” PARA OS DESENVOLVEDORES, CADA UM DELES CONSTRÓI OS SEUS PROCESSOS E LANÇA PARA O SISTEMA OPERACIONAL QUE FAZ O ESCALONAMENTO.

- DIFERENÇAS ENTRE ESSE MODELO E O DE LAÇO COM INTERRUPÇÃO:

- NO DE LAÇO COM INTERRUPÇÃO: OS PRÓPRIOS DESENVOLVEDORES TEM QUE GERENCIAR O USO DO SISTEMA DE INTERRUPÇÃO: ISSO ACABA POR DIMINUIR UM POUCO A PRODUTIVIDADE DO TIME DE DESENVOLVIMENTO, ENQUANTO NO COM RTOS ISSO QUEM FAZ É O PRÓPRIO SO, FAZENDO A EQUIPE DE DESENVOLVEDORES MAIS PRODUTIVA.

- SO VS. APLICAÇÃO:

# S.O. vs aplicação



1- QUANDO TIVER UMA APLICAÇÃO MAIS COMPLEXA FUNCIONANDO SOB UM PROCESSADOR COM MAIS RECURSOS: PODERÍAMOS USAR DE UM SO COMPLETO(COM TODOS OS SEUS RECURSOS). [EXTREMO]

2- QUANDO TIVER UMA APLICAÇÃO DE MUITO BAIXA COMPLEXIDADE FUNCIONANDO SOB UM PROCESSADOR PEQUENO: NÃO SERIA NECESSÁRIO USAR DE UM SO, TANTO PORQUE A SUA COMPLEXIDADE NÃO IRÁ REQUERER ISSO E TAMBÉM PORQUE O PROCESSADOR TERÁ POUCOS RECURSOS DE MEMÓRIA PARA ATENDER AS NECESSIDADES DA APLICAÇÃO E DE UM EVENTUAL SISTEMA OPERACIONAL. [EXTREMO]

3- QUANDO TIVER UMA APLICAÇÃO COMPLEXA FUNCIONANDO SOB UM PROCESSADOR(CPU) SEM MUITOS RECURSOS: TALVEZ SEJA A SITUAÇÃO MAIS CRÍTICA UMA VEZ QUE PRECISAMOS DOS RECURSOS QUE O SO PROPORCIONA MAS NO PROCESSADOR NÃO HÁ ESPAÇO O SUFICIENTE PARA SE ABRIGAR O SO, TEREMOS QUE SER MUITO SELETIVOS COM O SO ECOLHIDO E MAIS AINDA COM QUE RECURSOS IREMOS USAR DELE NA APLICAÇÃO FINAL. [INTERMEDIÁRIO-PIOR]

4- QUANDO TIVER UMA APLICAÇÃO SIMPLES FUNCIONANDO SOB UM PROCESSADOR COM MUITOS RECURSOS: DESNECESSÁRIO KKKKKKKKKKKKKKKKKK. [INTERMEDIÁRIO-MENOS PROVÁVEL]

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

## RECURSOS DE UM SO PARA EMBARCADOS:

- EM UM PROJETO DE APLICAÇÃO EMBARCADA CABE A NÓS COMO ENGENHEIROS DA COMPUTAÇÃO DARMOS O SUPORTE NA TOMADA DAS DECISÕES PRA ESCOLHA DOS MELHORES RECURSOS PRA IMPLEMENTÁ-LA. COMO O SO É UM DESSES RECURSOS E AQUILO QUE ELE VAI DISPONIBILIZAR PRA APLICAÇÃO VAI IMPACTAR O PREÇO O TEMPO DE PROJETO E A CAPACIDADE DO PROCESSADOR QUE VAI DAR SUPORTE, VAMOS COMEÇAR A PENSAR O QUE É QUE UM SO PODE OU DEVE OFERECER PRA QUE A GENTE TENHA COMO UTILIZAR ELE MELHOR NUM PRODUTO EMBARCADO:

- **Configurabilidade/Flexibilidade**
  - Motivação: diversidade de plataformas e recursos usados
  - Consiste em remover funções/serviços não usados
- **Mecanismo de proteção desnecessário**
  - Motivação: aplicações pré-testadas
  - Processos podem acessar I/O
  - Redução no overhead

- **CONFIGURABILIDADE/FLEXIBILIDADE DO SO** >> É NECESSÁRIA POIS É MUITO DIFÍCIL PREVER EM TEMPO DE DESENVOLVIMENTO DO SO A QUE TIPO DE APLICAÇÃO ELE VAI DAR SUPORTE, OS PRODUTOS EMBARCADOS PODEM NECESSITAR DOS MAIS DIVERSOS TIPOS DE HARDWARE, DOS MAIS DIFERENTES TIPOS DE SOFTWARE QUE PODEM PRECISAR DE MUITA OU POUCA MEMÓRIA. COMO ISSO NÃO É VISÍVEL A PRIORI, O QUE O SO FAZ É PERMITIR QUE NÓS COMO DESENVOLVEDORES CONFIGUREMOS O PERFIL DE SERVIÇOS QUE QUEREMOS UTILIZAR, ESSA POSSIBILIDADE DE PODER REMOVER ALGUMAS FUNÇÕES DO SO VAI PERMITIR QUE ELE SEJA MAIS RÁPIDO OU QUE ELE GASTE MENOS MEMÓRIA. UM BOM SO DEVE PERMITIR QUE DESABILITEMOS TODOS OS SERVIÇOS QUE NÃO FORMOS UTILIZAR DELE! (AO FAZER ISSO A VERSÃO FINAL DO SO QUE VAI SER EMBARCADA JUNTO AO PRODUTO SERÁ UM VERSÃO DO SO MAIS "ENXUTA").

-EXEMPLO: PODEMOS ESTAR USANDO UM SO POR PRECISARMOS DE UM ESCALONADOR DE TAREFAS MAS TALVEZ NOSSA APLICAÇÃO NÃO PRECISE DE GERENCIAMENTO DE DISCO, ENTÃO, BASTA IR NA FERRAMENTA DE CONFIGURAÇÃO DO SO DESABILITAR TODOS OS SERVIÇOS RELACIONADOS A GERENCIAMENTO DE DISCO O QUE VAI FAZER COM QUE A VERSÃO FINAL DE DADO SO FIQUE MAIS CURTA.

- **MECANISMOS DE PROTEÇÃO DESNECESSÁRIO** >> É MUITO COMUM OS SOS QUE USAMOS EM COMPUTADORES PESSOAIS OFERECEM MECANISMOS DE PROTEÇÃO(EVITAR SOFTWARES MALICIOSOS INSTALADOS DE FORMA IRRESPONSÁVEIS). ESSE TIPO DE SITUAÇÃO É POUCO PROVÁVEL DE ACONTECER NUMA APLICAÇÃO EMBARCADA COMUM, JÁ QUE ESSA JÁ TEM SEUS CONJUNTOS DE SOFTWARES INSTALADOS DURANTE SUA FABRICAÇÃO E NÃO IRÃO NECESSITAR DE "INSTALAÇÕES EXTRAS" PARA FUNCIONAR, E MESMO QUE PRECISAR SERÁ FORNECIDO PELO PRÓPRIO FABRICANTE DO EQUIPAMENTO.

- UM SO NÃO VAI TER QUE LIDAR COM SITUAÇÕES DE APLICAÇÕES NOIVAS CHEGANDO PARA FUNCIONAR ACRESCIDAS A AQUILO QUE JÁ ESTÁ DISPONÍVEL PARA O EQUIPAMENTO. ASSIM, TAIS TIPOS DE RECURSOS DE MECANISMOS DE PROTEÇÃO PODEM ACABAR POR SENDO DESNECESSÁRIOS E MESMO QUE ESTEJAM DISPONÍVEIS PODEM SER DESABILITADOS QUANDO O DESENVOLVEDOR QUE ESTEJA USANDO DE UM SO COM TAIS FERRAMENTAS JULGAR NECESSÁRIO.

- SO's TAMBÉM OFERECEM PROTEÇÃO ATRAVÉS DE FILTROS PARA ACESSOS A DISPOSITIVOS DE ENTRADA E SAÍDA. EM UM PC's AS APLICAÇÕES NÃO CONSEGUEM IR DIRETAMENTE NO HARDWARE ISSO SÓ SE FAZ ATRAVÉS DO SO. **PARA APLICAÇÕES EMBARCADAS É DESEJÁVEL QUE NÃO SEJA DESSE JEITO JÁ QUE MUITOS RECURSOS DE HARDWARE PODEM SER ACRESCENTADOS DEPOIS QUE O SO TIVER INSTALADO, TIRANDO POIS, TAL RECURSO DE PROTEÇÃO CONSEGUIMOS AUMENTAR O DESEMPENHO DE NOSSO SISTEMA.**

#### RECURSOS DE UM RTOS PARA EMBARCADOS:

**Alto desempenho**

**Baixo consumo de memória**

**Baixa potência**

**Baixo custo**

- A IMAGEM ACIMA MOSTRA 4 CARACTERÍSTICAS DESEJADAS E IMPORTANTES PARA QUALQUER PRODUTO! COMO QUE UM SO(OU RTOS PARA APLICAÇÕES EMBARCADAS) AJUDA NA OBTENÇÃO DE TAIS CARACTERÍSTICAS? QUAIS ELEMENTOS VÃO CONTRIBUIR NO DESEMPENHO DE UM APRODUTO FINAL:

#### 1 – NO DESEMPENHO DA APLICAÇÃO:

<b>Alto desempenho</b>	<b>Baixa potência</b>
<b>Baixo consumo de memória</b>	<b>Baixo custo</b>

Desempenho da aplicação é função de:

- Desempenho da CPU
- Eficiência do código da aplicação
- Eficiência do S.O.

#### 2 – NA MEMÓRIA :

<b>Alto desempenho</b>	<b>Baixa potência</b>
<b>Baixo consumo de memória</b>	<b>Baixo custo</b>

- Memória afeta o preço e o tamanho do produto final
- Configurabilidade do S.O. ajuda a manter o uso de memória reduzido

#### 3 – NA POTÊNCIA :

<b>Alto desempenho</b>	<b>Baixa potência</b>
<b>Baixo consumo de memória</b>	<b>Baixo custo</b>

- Menor memória leva a redução na potência
- S.O. eficiente ajuda em:
  - CPU de organização menos complexa
    - Menor preço e menos potência
  - Freqüência de clock menor
    - Reduz a potência

#### 4 – NO CUSTO :

<b>Alto desempenho</b>	<b>Baixa potência</b>
<b>Baixo consumo de memória</b>	<b>Baixo custo</b>

- Redução na memória reduz o preço
- CPU mais simples é mais barata
- Custos de propriedade intelectual
  - Licenciamento
  - Suporte

- LICENCIAMENTO OU SUPORTE SÃO FORMAS QUE DEPENDEM MUITO DAS NOSSAS DECISÕES, OU VC INVESTE EM UM OU EM OUTRO (UMA EQUIPE MENOS LICENCIADA/LEIGA DO SISTEMA DEVERÁ



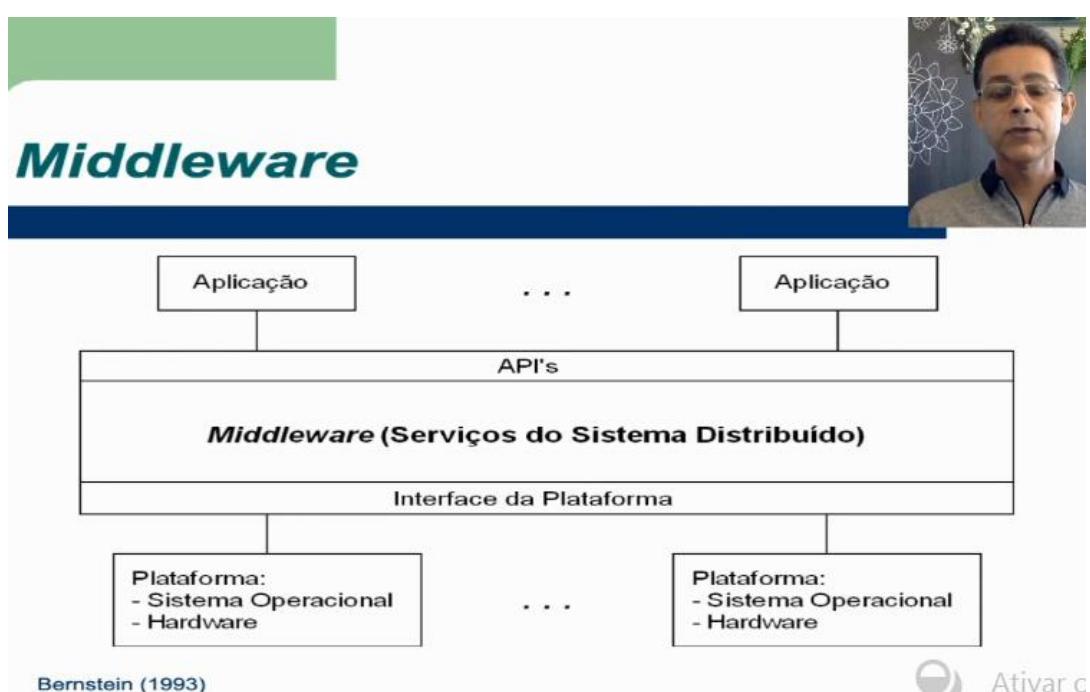
TER MAIS INVESTIMENTO EM SUPORTE PARA MEXER NO MESMO. OUTRA EQUIPE, PORÉM, QUE NÃO PRECISE DE SUPORTE DEVERÁ TER SEUS GASTOS EM TREINAMENTOS PARA SE TORNAR LICENCIADA NO SISTEMA TRABALHADO).

//////////

#### - MIDDLEWARE:

- CONCEITO PROPOSTO INICIALMENTE PELA COMUNIDADE DE SISTEMAS DISTRIBUÍDOS, PENSANDO EM COMPUTADORES DE CAPACIDADE MUITO ALTA.
- PALAVRA UTILIZADA EM TODA A COMPUTAÇÃO ATÉ MESMO EM SISTEMAS EMBARCADOS DE TODOS OS TIPOS E COMPLEXIDADES.
- DEFINIÇÃO BÁSICA: INTRODUZEM FUNCIONALIDADES DE COMUNICAÇÃO SOBRE FUNCIONALIDADES BÁSICAS PROVIDAS PELO SO.
- ORGANIZAÇÃO: COSTUMAM SER COLOCADOS ABAIXO DA SUA APLICAÇÃO E SOBRE O SO(PODENDO ALGUNS FUNCIONAREM SOBRE VÁRIOS SO's DIFERENTES).

- \*ilustração:



#### - FRAMEWORK vs MIDDLEWARE:

- É muito comum um intercâmbio entre essa expressões mesmo elas não sendo exatamente iguais.
- Existem muitos middlewares no mercado que tem exatamente a função que muitos frameworks tem também nesse meio, a de ampliar os serviços que o SO oferece tentando facilitar a trocar de recursos do hardware sem que seja necessário modificações significativas na aplicação.

#### - MIDDLEWARE ou FRAMEWORKS:

- AMBOS SÃO DE CERTA FORMA CONJUNTOS DE SOFTWARES/ARTEFATOS DE SOFTWARE QUE VÃO AJUDAR A DESENVOLVER APLICAÇÕES EM NÍVEIS DE ABSTRAÇÃO UM POUCO MAiores FACILITANDO O REUSO.
- COM O MIDDLEWARE PODEMOS MAIS FACILMENTE TROCAR OS RECURSOS DE HARDWARE QUE DETERMINADA APLICAÇÃO UTILIZA SEM PRECISAR FAZER MODIFICAÇÕES SIGNIFICATIVAS NELA.

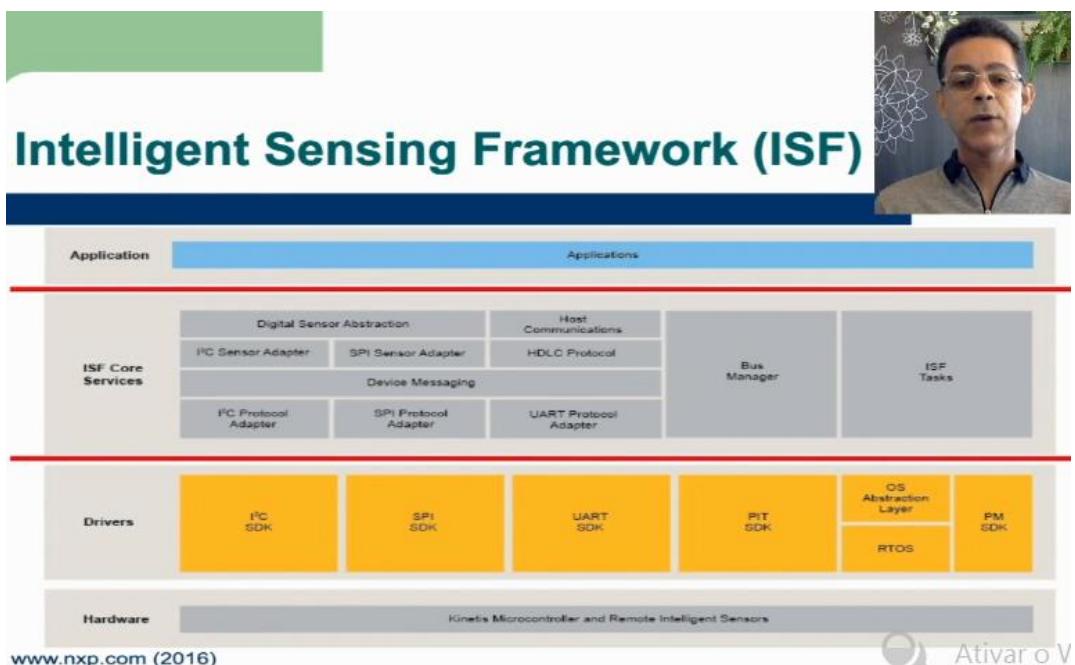
## Exemplo de Framework



### Intelligent Sensing Framework (ISF)

- Standard ANSI C code supplied as source code
- Minimum Flash footprint 34 KB
- Sensor Fusion included
- Minimum RAM requirements 6 KB

(AS LINHAS GRIFADAS DE VERMELHO MOSTRAM COMO OS RECURSOS DOS FRAMEWORKS PODEM ACABAR SE REDUZINDO PARA APLICAÇÕES EMBARCADAS PORQUE ELAS NECESSITAM DE MENOS DOS SEUS RECURSOS COMO AS MEMÓRIAS FLASH E RAM QUE SÃO BEM MENORES DO QUE ERAM OFERTADAS PELO FRAMEWORK).



- **REUSO É A PALAVRA CHAVE!**: PROCURAR REAPROVEITAR PEDAÇOS DE CÓDIGOS DE APLICAÇÕES PRÓPRIAS OU RECURSOS DE SOFTWARE DISPONÍVEIS POR TERCEIROS: MIDDLEWARES, SO's/RTOS's, BIBLIOTECAS.
- LICENCIANDO PROPRIEDADE INTELECTUAL DE TERCEIROS PODEMOS TAMBÉM DIMINUIR O TIME-TO-MARKET DO PROJETO.
- **SISTEMAS OPERACIONAIS E BIBLIOTECAS: TAMBÉM DÃO IMPORTANTÍSSIMO SUPORTE PARA O DESENVOLVIMENTO DE APLICAÇÕES.**

#### Comunicação(e suas tecnologias em sistemas embarcados):

- Muitas aplicações embarcadas vão utilizar comunicação pelos mais diversos motivos.

\*Que requisitos as aplicações embarcadas solicitam das tecnologias de comunicação.

- Por que comunicar:

- A comunicação é própria de tal aplicação (Inerente a aplicação).

- Exemplos: Controle remoto, telemetria celular. Neles é muito comum haver placas com microcontroladores conectadas a algum tipo ou vários tipos de sensores e as medições feitas por esses são armazenadas localmente e transmitidas para algum receptor remoto sendo que estas transmissões são transmitidas normalmente via comunicação sem fio podendo também utilizar de canais de satélite e outras comunicações mais baratas.

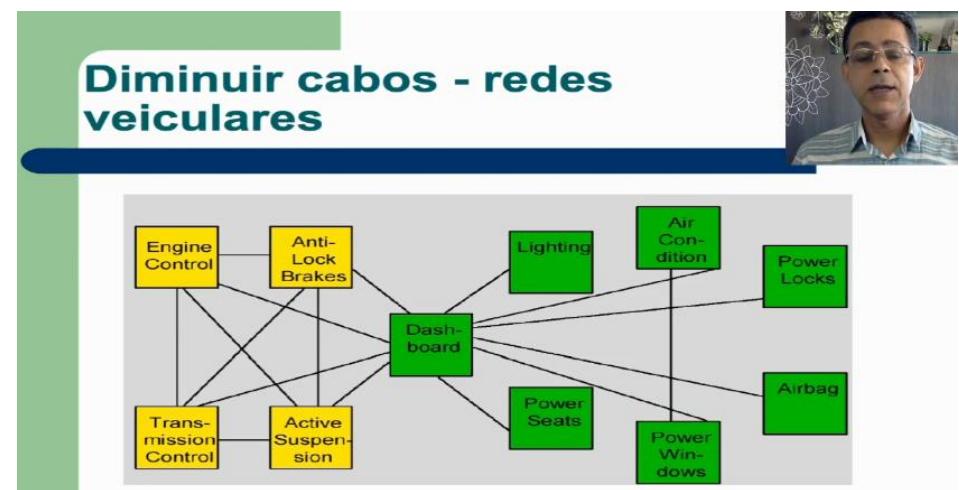


- Diminuir a complexidade das conexões (Tipicamente, diminuir cabos):

- Isso ocorre em equipamentos ou máquinas que produzem grandes quantidades de dados distribuídos dentro delas (Atividades fisicamente distribuídas). Solução: Uso de redes de computadores pode ajudar a diminuir a quantidade de cabos (USAR REDES DE DADOS).

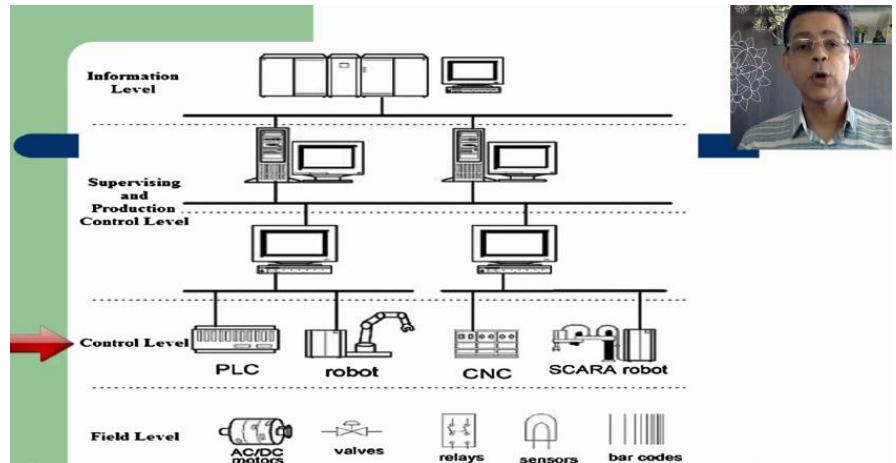
- Exemplos: Veículos modernos, robôs (uso de rede local).

- A imagem abaixo mostra um diagrama que é uma simplificação das conexões típicas dentro de um automóvel: sistemas de transmissão, equipamentos de ar-condicionado, lâmpadas, sensores de portas. (**Problema**) E todos esses dispositivos devem ser conectados entre si ou conectados a um painel do automóvel, isso gera uma quantidade de cabos muito grande. Solução: Fazer com que cada um desses dispositivos ganhe algum tipo de inteligência pra que possam ser capazes de conectar-se a uma rede de dados (temos então agora um cabo para rede de dados percorrendo todo o equipamento ao invés de ter-se cada dispositivo conectado ponto-a-ponto a um painel de controle).



- Modulação:

- Nesse caso é preciso ter vários computadores dentro de um equipamento agindo como núcleos independentes que irão cuidar de parte do processo se comunicando por rede de dados de maneira que cada um responde pela sua missão especificamente sendo a comunicação entre eles dada em alto nível. Por exemplo, na imagem abaixo ver-se que em chãos de fábrica nas indústrias temos várias máquinas inteligentes, elas tem seu próprio sistema de controle que é conectado entre si via rede, assim, uma máquina pode informar resultados de alto nível para outra máquina para que ela possa organizar seu processo de tal forma que todas essas informações possam “subir a um nível de controle mais elevado” onde seus próprios operadores possam saber em que fases de seus processos tais máquinas estejam trabalhando por exemplo.



#### - Tolerância a falhas:

- Muito parecido com a [MODULARIDADE](#). Porém, além do que é visto na modularidade há ainda a redundância de funções, ou seja, passa-se a ter equipamentos na rede que são capazes de fazer a mesma coisa estando conectados em rede em um monitora o outro: **Se houver falha de um deles o outro pode entrar em substituição.**

#### - Desempenho:

- Utilizar da comunicação para se ter vários equipamentos distribuindo a computação entre eles e, assim, aumentando o desempenho do conjunto(Não muito comum em SEMBs).
- Exemplo: “[rede intra-chip](#)”(NOC).
  - Surgiu nos últimos anos e vem sendo proposta(já existe hoje provavelmente kkkk).
  - É o conceito de redes dentro de um chip. Dentro de uma mesma “pastilha” há centenas de processadores conectados em rede estando aptos a cooperar na execução de um algoritmo complexo.

#### - Requisitos da comunicação(devem seguir a exigência que SEMBs pedem, a de serem enxutos e objetivos):

## Requisitos da comunicação



- **Bandwidth**
- **Robustez**
  - Temperaturas extremas, umidade, interferência eletromagnética
- **Tolerância a falhas**
  - reenvio, ack, checksum
- **Manutenção**
- **Privacidade dos dados**
  - criptografia, fios x wireless
- **Tempo-real**

Ativar o Wi-Fi

### - Bandwidth(Largura de banda):

- É normalmente associada a velocidade de comunicação da rede. A faixa de frequências que o canal de comunicação suporta. Se estamos utilizando por exemplo um par de fios para transmitir uma informação é possível avaliar a faixa de frequências que pode passar por aquele fio o que vai determinar em última análise a taxa de transmissão de dados hoje em dia que aquele meio de comunicação suporta.

- Existem softwares que podem ser colocados em computadores para a medição da taxa de transferência de dados entre um computador e algum servidor na internet por exemplo, uma forma de saber qual é a banda disponível na conexão de internet usada.

- Cada tecnologia de comunicação que existe oferece uma certa largura de banda, uma certa capacidade de transmissão de dados.

- Bandwidth: **Infra-vermelho x Bluetooth.**

### - Infra-vermelho:

- Taxa de transmissão de dados muito baixa.

- Não possuí muita velocidade.

- É utilizado para que seu controle remoto de televisão possa informar coisas para seu aparelho de televisão e para essa situação é necessária a transmissão de uma pequena quantidade de dados e não precisa de uma pequena quantidade de dados.

- Ambas as tecnologias do **Bluetooth** e do **Infra-vermelho** são feitas para necessidades diferentes!

### - Robustez:

- É muito comum dizer que um computador ROBUSTO é aquele que é muito rápido. Mas tal palavra não está associada a ALTO DESEMPENHO, ROBUSTEZ está associado a capacidade de alguma coisa resistir a uma situação adversa e se manter funcionando("de pé"), a exemplo de "condições adversas" temos por: Temperaturas extremas, umidade, interferência eletromagnética.

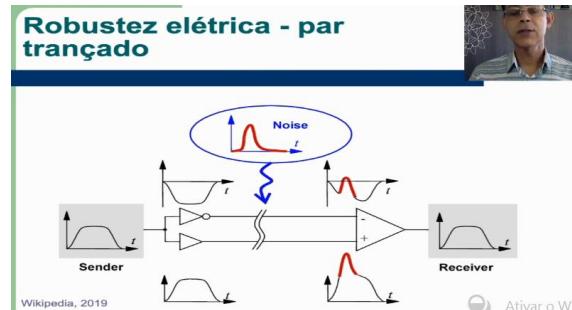
- **COMUNICAÇÃO ROBUSTA É UM MEIO DE COMUNICAÇÃO QUE AGUENTA CONDIÇÕES ADVERSAS(MESMO QUE ALGUMAS COISAS TENTEM COMPROMETER A QUALIDADE DA TRANSMISSÃO A ROBUSTEZ DA COMUNICAÇÃO PERMITE ESTA AINDA FUNCIONE E SEJA FEITA).**

## - ROBUSTEZ ELÉTRICA – PAR TRANÇADO:

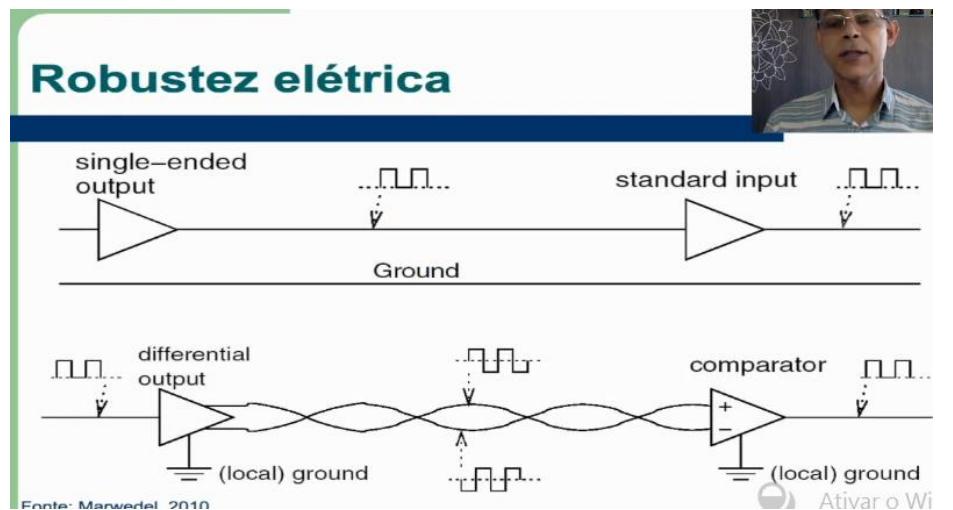
- Usada em cabos ETHERNET, cabo USB.

- Usada como forma de aumentar a robustez do canal de comunicação. Protegendo ele de coisas que procurariam comprometer a qualidade da transmissão.

## - FUNCIONAMENTO:



- A imagem acima demonstra o funcionamento da tecnologia do par-trançado: Na transmissão utilizando par-trançado dois fios são utilizados para transmitir a informação num deles você terá o sinal NORMAL e no outro ele INVERTIDO. O dois sinais de entrada SIMÉTRICOS(mais invertidos) vão trafegar pelo par de fios da origem até o destino onde há um circuito subtrator(Amplificador Operacional) que vai pegar o sinal e subtrair do seu inverso onde sairá um sinal DOBRADO (SIGNAL  $-[- \text{SINAL}]$ ). O ruído vai entrar igualmente nos dois fios e é por isso que estes devem ser muito próximos e bem enrolados, para que o ruído que irá ser induzido num fio seja exatamente o que irá ser no outro(Quando esse ruído chegar no subtrator ele é subtraído dele mesmo e será zero, não haverá, portanto, ruído na saída).



- A estratégia do par-trançado se contrapõem aquela “TRADICIONAL” vista na imagem acima pois no modelo TRADICIONAL tem-se o tráfego da informação e o fio terra como referência, então, se algum ruído for induzido no sinal não há como diferenciar o sinal da saída. O modelo do par-trançado aumenta exatamente a robustez dos canais de comunicação TRADICIONAIS(já que o ruído induzido é cancelado na saída!).

- **VANTAGENS:** Na prática a tecnologia do par-trançado vai ser mais imune/mais robusta a ruídos. Isso faz com que tal tipo de cabo possa SER MAIS LONGO, SE EXPONDO A QUANTIDADES MAIORES DE RUÍDOS SEM QUE A INFORMAÇÃO SEJA COMPROMETIDA AO SER TRANSMITIDA POR ELE.

- **Tolerância a falhas:**

- Mecanismos de proteção contra falhas, capacidade de reenvio de informações, confirmação de que dadas informações chegaram a seus respectivos destinos e etc(reenvios, ack, checksum...).

- **Manutenção:**

- **A facilidade de manutenção** é outra necessidade importante pois dependendo de onde determinada rede esteja instalada pode não ser possível mandar um técnico por exemplo “concertar” determinado problema no cabo danificado. **Para isso tecnologias sem fio são ESSENCIAIS no ramo!**

- **Privacidade dos dados:**

- **FIOS x WIRELESS: COMUNICAÇÕES SEM FIO SÃO MAIS FACILMENTE INTERCEPTADAS DO QUE AQUELAS POR FIOS!**

- CRIPTOGRAFIA DOS DADOS TRANSMITIDAS(TAL CRIPTOGRAFIA PODE SER INERENTE A TECNOLOGIA DE COMUNICAÇÃO QUE ESTEJA SENDO UTILIZADA).

- **Tempo-Real:**

- Várias aplicações embarcadas são de Tempo-Real, ou seja, elas precisam ter alguma garantia de que a computação vai acontecer dentro de um certo limite de tempo, em comunicação é a mesma ideia, precisa-se que determinada rede de comunicação de dados garanta que determinado dado seja entregue ou que esse seja entregue com uma latência/atraso máximo pré-determinado

**-(É O CASO DE APLICAÇÕES AUTOMOTIVAS ONDE-SE TEM UMA REDE DE DADOS DENTRO DO AUTOMÓVEL E EXISTE REQUISITO DE TEMPO PARA QUE A MENSAGEM TRAFEGUE DA ORIGEM AO SEU DESTINO, SE A MENSAGEM NÃO CHEGAR DENTRO DO PRAZO ISSO PODE VIR A COMPROMETER A INTEGRIDADE DA APLICAÇÃO).**

- **Acesso ao meio físico:**



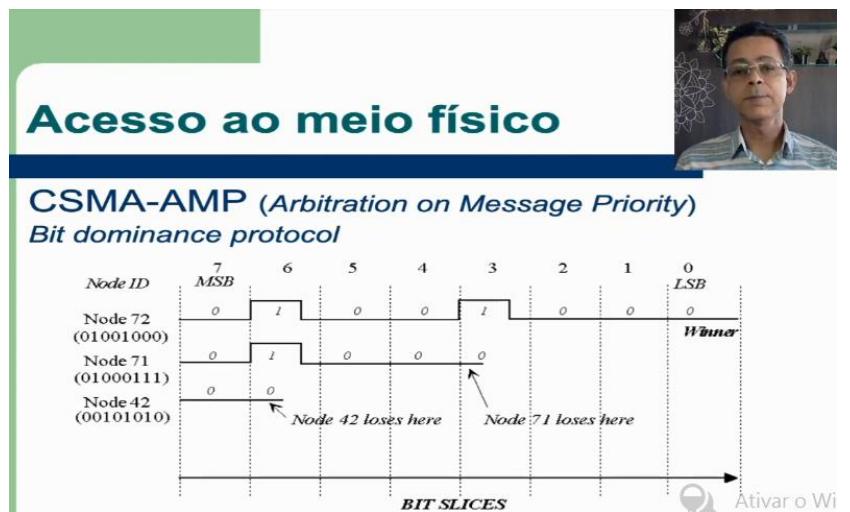
- CSMA é um sigla padrão para definir protocolos de acesso ao meio físico e possuí diversas variações. É um de muitos mecanismos para garantia de entrega de mensagens ou para controle de prioridade de acesso ao meio físico de comunicação.

**-EXEMPLOS QUE ADOTAM TAIS PROTOCOLOS E SUAS VARIANTES: WIFI, ETHERNET.**

**-PROBLEMA-01:** Tanto no wifi quanto na ethernet temos protocolos de acesso ao meio físico que não garantem que a mensagem vai ser entregue e não garantem nenhuma prioridade pra ninguém! <<(Isso funciona muito bem em redes corporativas por exemplo, onde as mensagens eventualmente podem atrasar um pouco mais cabam chegando a seus destinos! MAS, há redes como as utilizadas em automóveis modernos chamadas de veiculares, que não se pode oferecer o tipo de protocolo do “faremos o melhor possível” kkkkk, precisa-se garantir que os dispositivos transmitam seus dados pela rede para aqueles que estão aguardando!):

**-SOLUÇÃO: “PROTÓCOLO DE BIT-DOMINANCE”.**

- Para evitar esse problema protocolos como o CSMA-AMP usam do protocolo de “BIT-DOMINANCE”-DOMINÂNCIA DE BIT, ou seja, aquele dispositivo que colocar o bit que tem mais PRIORIDADE É QUE VAI CONSEGUIR MANDAR SUA MENSAGEM PELA REDE. Cada dispositivo ao colocar seu número de identificação verifica se ele ganhou acesso ao meio ou se há alguém com mais prioridade que ele tentando utilizar o meio de comunicação. Na imagem a baixo veja que inicialmente no gráfico todos colocam zero e ficam empurrados, depois, dois colocam 1 e são prioridades e o outro zero, os com 1 ficam empurrados o com zero para sua tentativa de transmissão e espera o próximo ciclo para tentar de novo. Depois os que lançaram 1 lançam DUAS VEZES seguidas 0 e empurram até que o “Node 72” lança 1 que é prioridade sobre o 0 lançado pelo “Node 71” que deve parar sua tentativa e esperar o “Node 72” utilizar o meio de comunicação, após isso um novo ciclo se iniciar para ver quem usa o meio de comunicação.



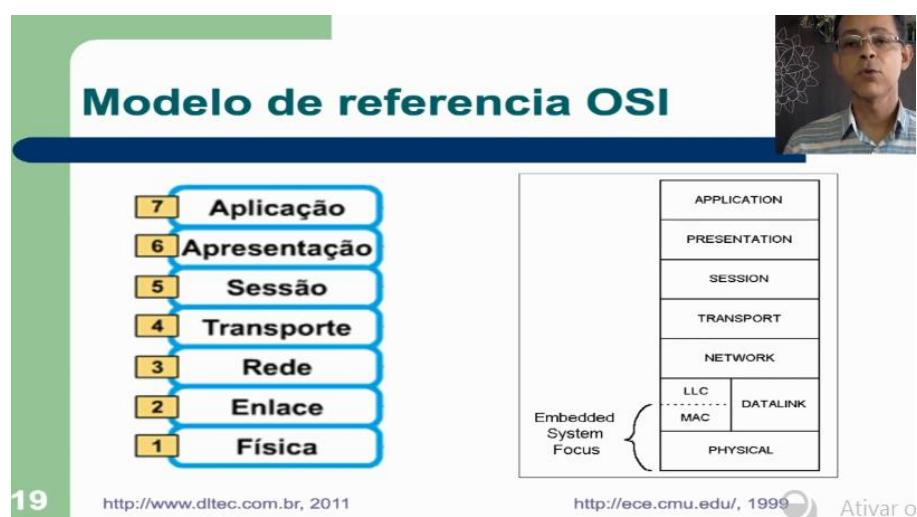
- Do ponto de vista prático o dispositivo que está conectado na interface “Node 72” não percebeu que havia outros dispositivos tentando acessar o meio de comunicação e transmitiu sua mensagem. Por isso, quando se montam projetos utilizando tal tipo de tecnologia colocam-se os “Nós”-“Nodes” de NÚMEROS MAIS ALTOS para os dispositivos que tem mais prioridade no uso da rede e com tal associação de prioridade quem tem que ganhar o acesso ao meio ganha e quem tem que esperar espera.

**-PROBLEMA-02(LOGICAMENTE kkkk) TODO PROTOCOLO BASEADO EM PRIORIDADE ESTÁ SUJEITO AO FENÔMENO DE “STARVATION”, dispositivos que tem menos prioridade eventualmente podem nunca chegar a transmitir suas mensagens!**

**SOLUÇÃO:** O próprio projetista do sistema tem que garantir que os dispositivos de mais alta prioridade não utilizem exageradamente o canal de comunicação.

**- Modelo de referência OSI:**

- É um modelo bastante antigo.
- Possui várias camadas estabelecidas para hierarquizar as várias etapas de funcionamento de uma rede de comunicação de dados.
- Nesse protocolo/modelo encontram-se geralmente 7 camadas para se trabalhar porém no mundo dos embarcados por razões de eficiência não teremos todas essas 7 camadas em funcionamento na verdade o que será mais comum de se encontrar será uma “simplificação desse modelo”. O mundo dos SEMB's tipicamente se concentra em implementar as “camadas mais baixas” do modelo (Enlace e Física) pra garantir que a comunicação exista e sem se preocupar em tornar o processo de comunicação muito abrangente/genérico e sim mais restrito que é o que as aplicações embarcadas visam(essa objetividade e reestritividade).
- Há dificuldades que são criadas uma vez que há essa limitação de se trabalhar com as camadas mais baixas apenas, que dizem respeito a elevar o nível de abstração no processo de comunicação.
- Porém o foco aqui é exatamente aumentar a eficiência pra que o dispositivo embarcado não tenha que gastar muita energia implementando várias camadas e vários protocolos e isso acabe por precisar de que seja necessário um processador com maior capacidade/desempenho na aplicação embarcada.
- A imagem abaixo mostra o modelo completo a esquerda e a direita aquilo que geralmente se trabalha em SEMB's:



**- Programação por troca de mensagens:**



Geralmente sistemas embarcados não possuem memória compartilhada.

- **Estabelecimento de conexão**
  - Mensagens têm destino explícito
- **Publish-subscribe**
  - *Data-push* (baseado em transferência de dados)
  - Envio periódico de dados

21

Ativar o

- Em aplicações que utilizam comunicação é natural que seus dispositivos troquem MENSAGENS ENTRE SI e o mais comum quando se tem dispositivos remotamente localizados é que esses utilizem algum padrão para troca de mensagens.

- A literatura de redes de computadores nos oferece dois grandes grupos de estratégias para trocas de mensagens:

- Estabelecimento de conexão:

- Típico para quem faz aplicações orientadas a internet onde existe um cliente e um servidor onde os dois estabelecem uma conexão e ambos ficam trocando mensagens entre si dentro dessa conexão, nesse caso o destino da mensagem é EXPLÍCITO:  
*Quando o dispositivo manda uma mensagem pro servidor ele informa pra quem ele está mandando essa mensagem. O mesmo se dar na volta, quando o servidor manda uma mensagem pra nós ele informou pra quem aquela mensagem está sendo enviada!*

- (**Vantagens**): Simplicidade após o estabelecimento da conexão já que a aplicação, agora, tudo que quiser mandar para determinado canal ela irá mandar através de uma certa (conexão pré-definida).-----

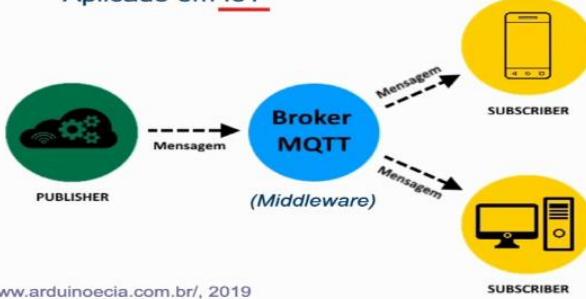
-----> - (**Problema**): O problema é que essa conexão vai precisar de uma certa estrutura de dados construída construída na memória do dispositivo embarcado e muitas vezes tal quantidade de memória para a estrutura passa a ser um pouco grande o que cria dificuldades para que o dispositivo possa dar suporte a essa tecnologia de comunicação.

- Publish-subscribe:

## Publish/Subscribe (publicador/assinante)



- MQTT (*Message Queue Telemetry Transport*)  
– Aplicado em IoT



- Baixo acoplamento
- Escalabilidade

22

www.arduinoecia.com.br/, 2019

Ativar o Wir

- Alternativa BEM MAIS BARATA do que a de “Estabelecimento de conexão” (Orientada a conexão).
- Muito utilizada quando o dispositivo embarcado precisa mandar dados periodicamente pra alguém.
- Oferece um \*BAIXO ACOPLAMENTO entre o cliente e o servidor, na verdade não existe o conceito de “servidor e cliente” mas sim o conceito de “alguém que produz um certo dado(PUBLISHER) e alguém que produz esse dado(SUBSCRIBER)”.
- Numa rede existem dispositivos, como sensores, que estão produzindo dados e os estão publicando, eles vão publicar esses dados num equipamento que vai funcionar como um administrador dessas mensagens, quando tais mensagens que o “PUBLISHER” publicar chegarem no “BROKER”/ADMINISTRADOR(na imagem acima seria o middleware da aplicação embarcada, aqui usado como exemplo o MQTT) existem dispositivos que estão cadastrados como assinantes dessas mensagens então quando um assinante se inscrever para receber mensagens de um publicador cada vez que uma mensagem do publicador chegar para o BROKER o assinante/SUBSCRIBER será notificado, assim, não existirá uma CONEXÃO DIRETA entre quem publica e quem é assinante:

- (Somente o BROKER irá saber quem está publicando e quem está consumindo exatamente.) -----

-----> - Isso aumenta a \*ESCALABILIDADE dessa topologia pois facilmente pode-se colocar novos assinantes para consumir informação sem que o publicador saiba que novos assinantes foram inseridos na rede!

[Vantagem do modelo: Publish-subscribe]

[Estabelecimento de conexão] X [Publish-subscribe]:

- [Publish-subscribe]:

- Então, tanto assinantes quanto publicadores podem entrar e sair da rede sem que os demais elementos da rede precisem ser notificados disso.

- [Estabelecimento de conexão]:

- No modelo de “Estabelecimento de conexão” o publicador teria que criar uma nova conexão com o novo assinante.

- MQTT:

- Tecnologia muito popular hoje que implementa o método de “Publish-subscribe”.
- Existem muitas implementações desse protocolo que podem ser usadas para funcionar até no seu próprio computador.
- Conectar sensores num ambiente residencial a um servidor com o protocolo MQTT e conectar dispositivos consumidores dos dados de tais sensores como celulares e computadores também.
- *O MQTT ganhou muita visibilidade com a popularidade do conceito de IoT nos tempos atuais!*

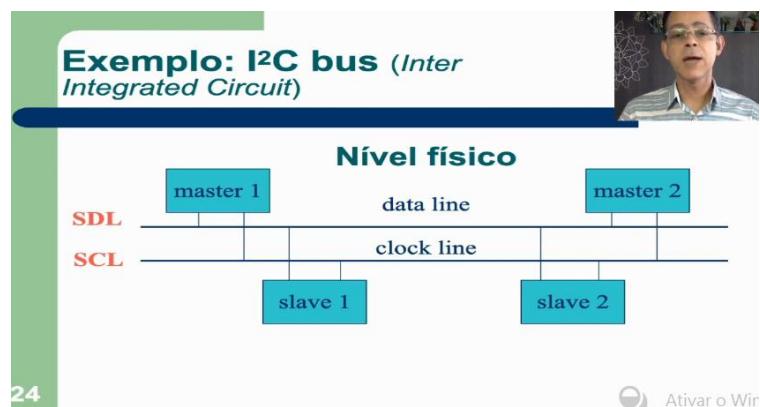
- Exemplos de protocolos conhecidos:

- Nesses protocolos de comunicação quem faz o CLOCK pra transmissão dos dados é o MESTRE/“MASTER”.

- I<sup>2</sup>C:



- Muito utilizado para conectar alguns sensores em microcontroladores(tanto que já foi-se falado dele antes kkkkk).
- É um padrão de conexão pra ser utilizado dentro da própria placa do computador embarcado, ou seja, não é utilizado pra que um dispositivo em um certo ambiente se comunique com outro numa localidade diferente ou até mesmo na mesma localidade(mesmo que a distância entre os dois seja mínima!), é um padrão de comunicação PARA FICAR DENTRO DO EQUIPAMENTO(Dois ou mais dispositivos DENTRO do equipamento embarcado VÃO SE COMUNICAR UTILIZANDO DESSA TECNOLOGIA, por isso o nome “*Inter Integrated Circuit*”).



- A figura acima mostra um exemplo de funcionamento: Temos 2 fios de comunicação(SDL e SCL) além do fio de terra sendo um para tráfego de dados(DATALINE) e outro para o clock, de maneira simples ele funcionará como aqueles registradores de deslocamento (estudados em Eletrônica Digital). Os dois registradores de deslocamento, o que transmite e o que recebe, vão utilizar o mesmo clock que é gerado pelo “Master” e os dados vão trafegar pelo SDL.

- SPI:

- A proposta do SPI é a mesma trazida pelo I<sup>2</sup>C, por isso é bom compará-los.

- SPI versus I<sup>2</sup>C:

The diagram is a slide titled "SPI (Serial Peripheral Interface Bus) versus I<sup>2</sup>C". It features a green vertical bar on the left with the number "27" at the bottom. On the right, there is a portrait of a man with glasses and a white shirt. The main content area lists advantages of SPI:

- Maior vazão de dados
- Escolha arbitrária do tamanho da mensagem (não limitado a palavras de 8 bits)
- Circuitos mais simples => menor potência
- Sinais unidirecionais => facilita isolamento
- Requer mais pinos (SCK, MISO, MOSI, SS)
- Admite apenas um dispositivo mestre
- Philips - I<sup>2</sup>C; Motorola - SPI

27



- Ambos os dois são volatdos para comunicação numa mesma placa!

- O SPI admite uma taxa de comunicação mais alta e portanto uma vazão maior de dados.

- O SPI também permite que a palavra transmitida entre os dois dispositivos tenha mais de 8bits.

- Quantidades de fios:

- Lembrando: Nesses protocolos de comunicação( I<sup>2</sup>C, SPI ) quem faz o CLOCK pra transmissão dos dados é o MESTRE/"MASTER".

- No I<sup>2</sup>C tínhamos um fio pra dados e outro pra clock pois o pino de dados era bidirecional, os dados vão e voltam pelo mesmo pino. (O endereçamento vai dentro da própria mensagem).

- No I<sup>2</sup>C é possível que mais de um dispositivo possa gerar clock no barramento de comunicação.

- No SPI existe um pino para os dados irem e um pino para os dados voltarem além do clock. (Existirá um pino para que o MESTRE/"MASTER" informe o destino daquela mensagem).

- No SPI é possível que apenas UM dispositivo MESTRE/"MASTER" possa gerar clock no barramento de comunicação.

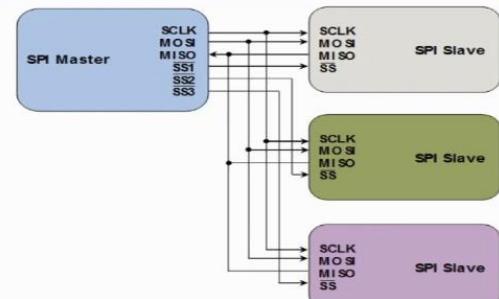
- Curiosidade: O padrão I<sup>2</sup>C foi proposto pela PHILIPS por isso é um pouco mais antigo que o SPI, proposto pela MOTOROLA. <<(Ainda assim são usados por muitos fabricantes independente de quem os fez!)

- Como se dar a conexão entre alguns dispositivos SPI:

## SPI (Serial Peripheral Interface Bus)



29



Ativar o Wiz

- A figura acima ilustra um exemplo com um dispositivo SPI “MASTER”([mestre](#)) e 3 dispositivos SPI “SLAVES”([escravos](#)).

- O pino de CLOCK é comum a todos os dispositivos e o “Master” é que vai gerar o sinal de clock que vai ser recebido pelos demais pinos dos outros dispositivos.

- O pino MOSI é por onde os dados do “MASTER” sairão e por onde os 3 “SLAVES” vão recebê-los.

- Os pinos SS1, SS2 e SS3 é por onde o “MASTER” escolhe poder se conectar a um dos “SLAVES”(“SS = Slave Select”). Para caso de mais escravos haverá mais pinos SS, que são apenas simples portas i/o somente para a seleção do dispositivo para quem ele quer transmitir ou de quem ele quer receber.

- CAN - (“Control Area Network”):

## Control Area Network (CAN) Bus



33

- Desenvolvido para o mercado automotivo. Estendido para eletrodomésticos, indústria etc
- Especificação CAN:
  - Nível físico
  - Nível de protocolo
  - Nível de filtragem de mensagem
- Baud-rate máximo de 1 Mbps (40 m).
- Utiliza transmissão orientada a mensagem.
- Não existe endereço de destino para as mensagens.

Ativar o Wiz

- Esse protocolo ou rede é um padrão de comunicação de dados que foi proposto inicialmente para ser utilizado em automóveis.

- ([Proposta](#)): A ideia desse padrão é garantir a comunicação de dados de uma forma segura. Esse tipo de necessidade depois veio a se extender para outras áreas da indústria além da de automóveis.

- [O padrão CAN hoje está presente na maioria dos automóveis de uso pessoal e também nos automóveis maiores como ônibus, tratores e equipamentos maiores.](#)

- ([Na sua especificação são dados detalhes sobre nível físico, de protocolo e de filtragem de mensagens.](#))-----

-----> Note que isso coincide com uma figura lá pra trás que dizia que os padrões pré-embarcados normalmente se concentram mais “nas camadas mais baixas” !!! << (Esse nível de filtragem de mensagens não chega nem a ter a complexidade de uma camada de redes é algo mais simples que isso!)\*\*\*\*

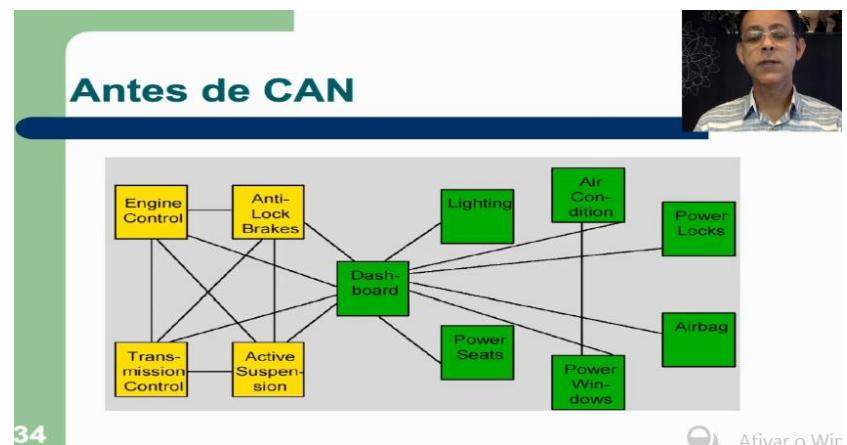
- Sua TAXA DE TRANSMISSÃO não é MUITO ALTA se comparado com outros padrões e isso naturalmente tem a ver com a Proposta/Objetivo do protocolo CAN:

- Não há necessidade de, por exemplo, no CAN se transmitir imagens em sua rede o que nos faz não precisar de uma BANDA que dê vazão a sinais de alta densidade, o que costuma trafegar portanto em sua rede devem ser: “DADOS VINDOS DE SENSORES OU INFORMAÇÕES PARA ORIENTAR ALGUNS ATUADORES DENTRO DO EQUIPAMENTO”.

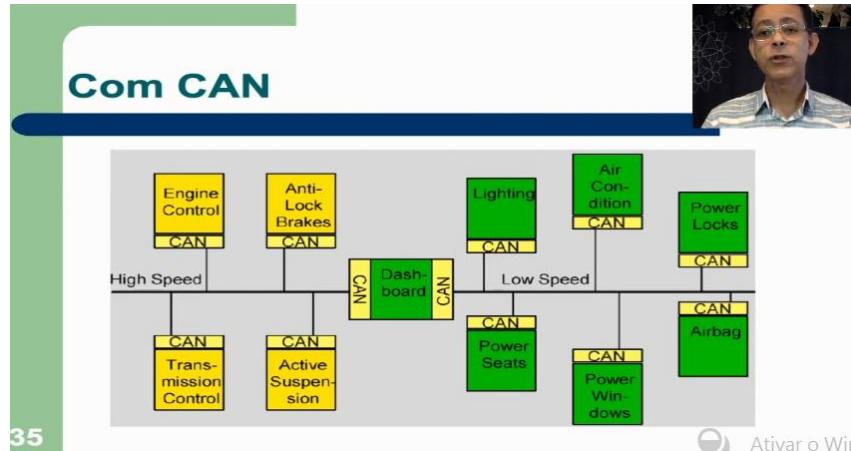
- Não possuí alcance MUITO ALTO já que o cabeamento dentro de veículos não precisam cobrir grandes distâncias.

- Utiliza o padrão “Publish-subscribe” orientado a mensagens e nesse caso não existirá um endereço de destino fixo da mensagem, cada dispositivo quando publica sua mensagem informa seu próprio endereço e o dispositivo assinante dessa mensagem é que vai saber se tem interesse em recolher essa mensagem da rede ou não.

- Na figura abaixo vemos que a DISPOSIÇÃO DE CONEXÃO entre os equipamentos dentro dos automóveis tende a ser um pouco caótica devido as várias necessidades específicas de conexões ponto-a-ponto entre vários equipamentos o que gera uma grande quantidade de cabos:



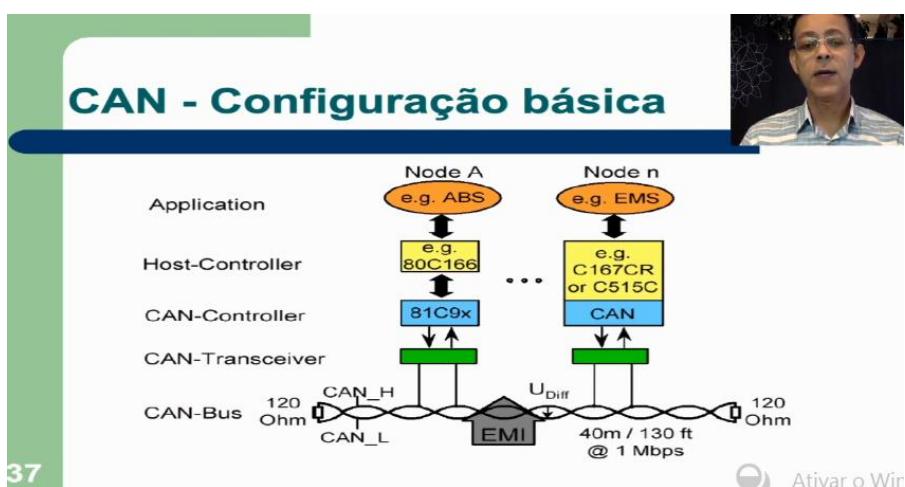
- Na figura abaixo vemos agora que quando adotamos uma REDE ESTRUTURADA dentro de um equipamento, no caso analizamos para um carro, como a rede CAN, nós diminuímos a complexidade do cabeamento usado e diminuímos a quantidade também de fios:



- Cada dispositivo passa a usar apenas os cabos do padrão CAN que usam do padrão PAR-TRANÇADO, onde, este mesmo PAR-TRANÇADO que irá percorrer toda a região onde estão os sensores ou atuadores que estão conectados naquela rede.

- É comum também criar redes separadas tanto para dispositivos mais críticos, como é o caso da rede AMARELA na figura acima, quanto para dispositivos que não precisam ter uma troca de mensagens muito crítica, como na rede VERDE da figura acima.

- Abaixo uma figura que ilustra o padrão de comunicação físico do CAN que é o PAR-TRANÇADO, que ajuda a aumentar sua imunidade a ruídos(do CAN) e EMI's(interferências eletromagnéticas) pois é mais ROBUSTO do que o PADRÃO COMUM DE FIOS:



- Na imagem acima temos dois exemplos ("Node A" e "Node n") de dois dispositivos mostrando o caminho dos seus dados até chegar no software de aplicação lá em cima.

- Temos um "Transceiver"(da rede CAN) que é um circuito eletrônico.

- Temos um chip/"Controller" (da rede CAN) que é o controlador.

- Temos acima de ambos “Controller” e “Transceiver” um microcontrolador “Host-Controller” que vai se comunicar com o “Controller” e receber ou enviar mensagens via protocolo “CAN”.

- Outra característica importante do CAN é a possibilidade de filtragem de mensagens por hardware. Como já vimos no padrão “Publish-subscribe” o dispositivo que é o assinante recebe as mensagens SE FOR DO SEU INTERESSE, ou seja, se a aplicação tiver interessada em receber mensagens daquele publicador. Esse filtro de “se a mensagem que tá chegando interessa aquele dispositivo ou não” pode ser feita por software que é o caso do “Controlador CAN básico” ilustrado abaixo:



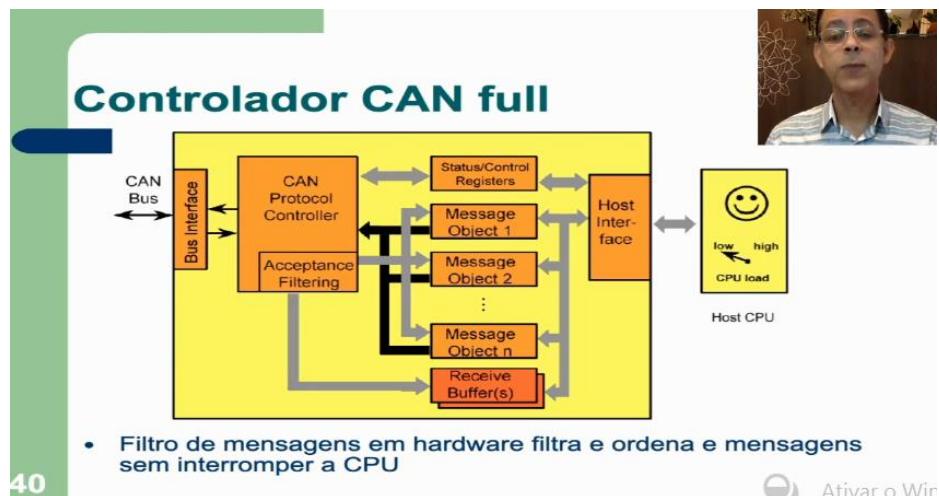
39

Ativar o WIR

- [Quando a mensagem chega ela é interpretada pelo protocolo CAN e o processador acima é avisado de que chegou uma mensagem mas sem saber se aquela mensagem é pra ele ou não. O processador, então, captura essa mensagem, o algoritmo que recebe essa mensagem identifica quem é o publicador e se não for de interesse descarta essa mensagem.]-----

----->(Problema): Isso fará com que a CPU do sistema fique todo tempo parando pra ver se as mensagens que estão passando pela rede são de interesse daquela aplicação ou não o que pode fazer com que a CPU fique SOBRECARREGADA!

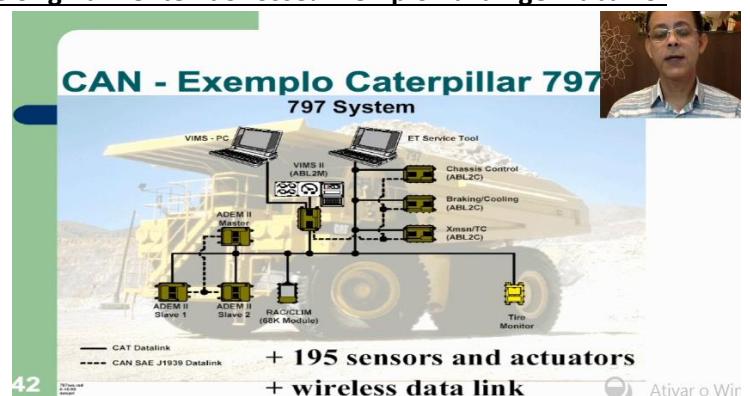
- Há ainda outra possibilidade de filtragem, a de se ter uma INTERFACE CAN “FULL”/COMPLETA:



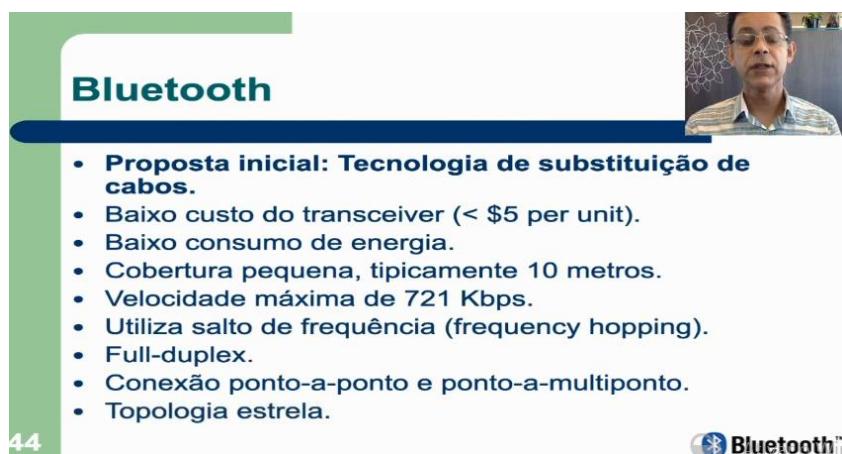
40

Ativar o WIR

- É capaz de filtrar as mensagens na própria interface.
- As mensagens vão chegar, mas previamente a aplicação já cadastrou quais são aqueles publicantes que ela está interessada em receber.
- Quando a mensagem chegar essa interface vai verificar se a aplicação assinou aquela mensagem se não for a mensagem vai ser descartada. Se a aplicação tiver assinado aquelas mensagens aí sim o processador vai ser interrompido e vai receber aquela mensagem. Com isso desloca-se para o hardware da interface uma das funções que poderia ser feita pelo processador.
- [\*Vamos ver que é extremamente comum quando falamos de aplicações embarcadas falar de “deslocar para o hardware alguma atividade que o processador iria fazer” .]----->>(Isso nos dá de cara um processador **MENOS OCUPADO!!!**)--->>(Se está menos OCUPADO ele pode ser um processador **MENOR** e portanto **mais BARATO**).
- \*Obs. É possível colocar na rede CAN uma comunicação SEM-FIO mesmo que originalmente não fosse! Exemplo na iamgem abaixo:



### - Bluetooth:

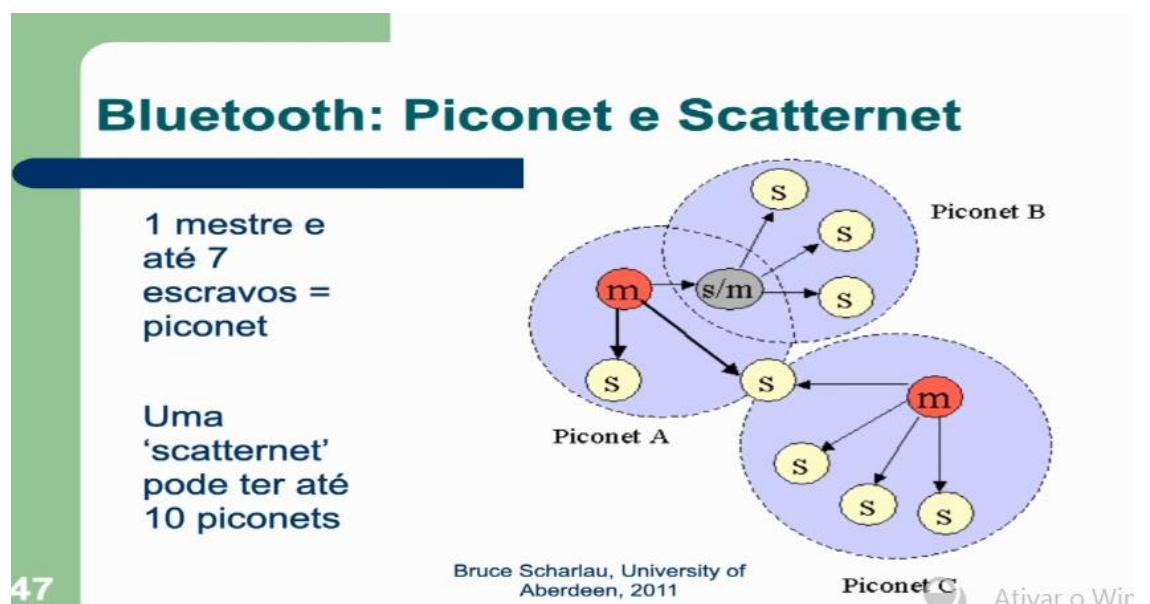


- Essa tecnologia se tornou bastante popular nos últimos anos.
- (Proposta Inicial): Conectar dispositivos próximos, substituindo aqueles cabos que ficam em cima das mesas por exemplo, conectando computadores, telefones, sistemas de som e muito mais. Assim, seus DOIS PRINCIPAIS objetivos eram:

- Um BAIXO CUSTO.

### - Um BAIXO CONSUMO DE ENERGIA.

- Considerando que a conectividade se daria entre dispositivos muito próximos o alcance estabelecido na PRIMERIA VERSÃO de BLUETOOTH era de apenas 10 metros.
- Sua velocidade de comunicação não precisava ser ALTA. TIPICAMENTE UTILIZADO PARA TRANSMITIR DADOS EM BAIXA QUANTIDADE OU SOM.
- Outra característica interessante é que essa tecnologia se propunha a ser CAPAZ DE CONVIVER COM OUTROS EQUIPAMENTOS DE COMUNICAÇÃO que pudessem ESTAR FUNCIONANDO PRÓXIMO. Para evitar a INTERFERÊNCIA entre eles é que foi inventada a tecnologia de TROCA AUTOMÁTICA DE CANAIS DE COMUNICAÇÃO.
- \*O sistema bluetooth consegue conviver muito bem com outros equipamentos que usem radio-frequência e que estejam em seu mesmo raio de alcance.
- Sua transmissão de dados é FULL-DUPLEX, ou seja, existe um canal para ida e outro para a volta dos dados que conseguem funcionar de MANEIRA CONCORRENTE.
- O uso MAIS COMUM que se vê do BLUETOOTH é na CONEXÃO PONTO-A-PONTO(Quando tipicamente conectamos um fone de ouvido a um celular ou um celular num computador, sempre UM DISPOSITIVO COM OUTRO).
- A tecnologia BLUETOOTH também permite a CONEXÃO PONTO-A-MULTIPONTO(Quando um DISPOSITIVO PODE TRANSMITIR E VÁRIOS PODEM RECEBER AO MESMO TEMPO). Para isso o bluetooth usa TOPOLOGIA EM ESTRELA(que é mais COMPLEXO que PONTO-A-MULTIPONTO kkkk).
- A figura abaixo nos dá uma ideia das possibilidades que o bluetooth tem ainda que a maioria dos produtos conhecidos por você leitor não utilize isso:



- Nota-se um complexa estrutura de rede onde um dispositivo pode transmitir para vários outros, e, além disso, um dispositivo pode ser utilizado como PONTE para transmitir dados de uma pequena rede para outra pequena rede. Essa tecnologia é muito pouco explorada pelos produtos que conhecemos hoje no cotidiano MAS ESTÁ DISPONÍVEL NA ESPECIFICAÇÃO BLUETOOTH.

### - Bluetooth versus Wifi:

- Comparação muito comumente feita.
- Ambas as propostas tem **OBJETIVOS DIFERENTES** e pretendem atender **NECESSIDADES DIFERENTES** das aplicações.

**Bluetooth versus wifi**

45

Ativar o Vir

- 802.11b
  - Baud-rate de até 11Mbps
  - Alcance de até 100 m
  - Projetado para conectar dispositivos com muita potência e velocidade
  - Opera a 2.4 GHz
- Bluetooth
  - Baud-rate de até 1Mbps (teórico)
  - Alcance de até 10 m
  - Menor necessidade de potência
  - Projetado para conectar pequenos dispositivos e periféricos
  - Opera a 2.4 GHz (banda ISM - Industrial, Scientific and Medical)

- Analizando a imagem acima e seus dois exemplos de wifi e bluetooth vemos que a **tecnologia WIFI(802.11b)**:

- Opera numa banda bem mais elevada (Baud-rate).
  - Tem o alcance mais alto do que o bluetooth padrão.
  - Vai precisar de uma potência mais alta para estabelecer sua conexão de mais alta velocidade.
- A FREQUÊNCIA DE 2.4 GHz é a frequência TÍPICA UTILIZADA PELOS DOIS PADRÓES, tanto BLUETOOTH quanto WIFI(embora HOJE TENHAMOS UMA PADRÃO DE 5 GHz para o WIFI).**

- Analizando a imagem acima e seus dois exemplos de wifi e bluetooth vemos que a **tecnologia BLUETOOTH**:

- Taxa de transmissão de dados BEM MAIS BAIXA(banda/baud-rate).
  - Tem o alcance mais baixo também.
  - \*Mais baixa potência de transmissão. **Tanto a baixa taxa de transmissão de dados quanto o baixo alcance são baixos exatamente por visarem preservar o requisito de MAIS BAIXA POTÊNCIA!**
  - Feito para conectar dispositivos que devem ser alimentador por baterias que devem ser pequenos e com um sistema de energia não muito sofisticado.
- A FREQUÊNCIA DE 2.4 GHz além de ser utilizada por WIFI e BLUETOOTH é utilizada por muitos outros produtos que utilizam comunicação sem-fio. Tá dentro dessa banda chamada “ISM” que é uma banda genérica(Qualquer produto que utilize essa banda não precisa de um protocolo MUITO SOFISTICADO e de licenciamento para utilizar essa frequência de 2.4 GHz o que explica porque essa frequência é muito utilizada).**

- **Tecnologias utilizadas para comunicação, especialmente no mercado de embarcados:**

- A **figura abaixo** nos mostra um panorama geral das várias tecnologias utilizadas para comunicar especialmente no mercado de embarcados.

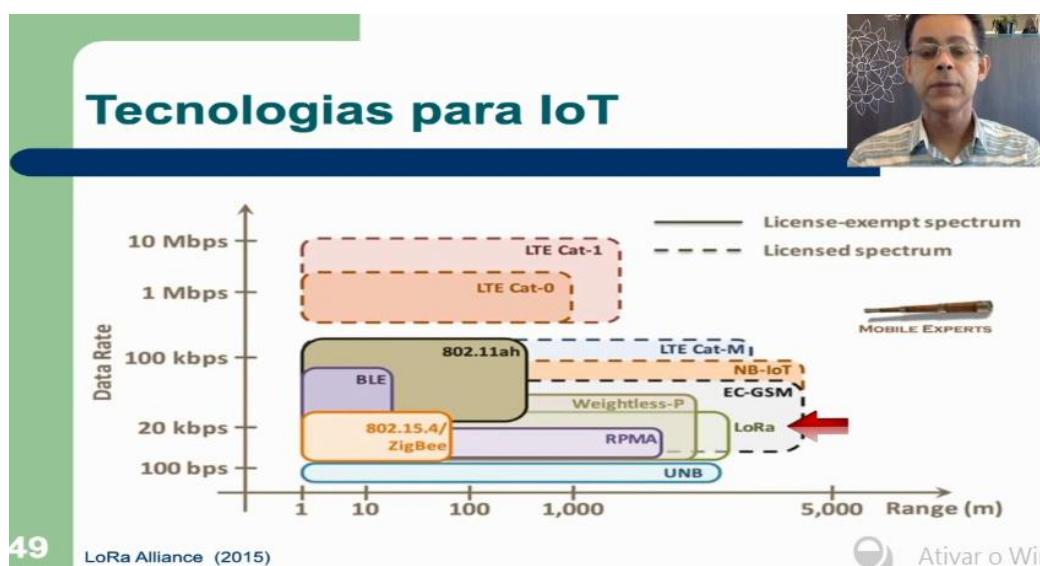
- Na figura abaixo temos o LTE(LTE Cat-1 e LTE Cat-0) em várias faixas de frequência como uma das tecnologias utilizadas pelos telefones celulares. Temos o bastante popular WIFI (802.11ah) o BLUETOOTH (BLE) e outros não tão populares como o ZigBee e LoRa.

- A figura abaixo pode ser vista como um mapa que pretende mostrar a disponibilidade de oferta de tecnologias de comunicação sem-fio combinando na mesma figura a taxa de transmissão(Data Rate) e o alcance(Range).

- Na figura abaixo percebe-se a grande quantidade de tecnologias que oferecem um alcance elevado com uma banda não muito elevada. Acontece que oferecer uma banda muito elevada e um alcance muito elevado implica na necessidade de uma ALTA POTÊNCIA para funcionar. Então, tentando economizar potência busca-se oferecer ou um alcance elevado ou uma taxa de transmissão elevada.

- A tecnologia LoRa tornou-se bastante popular nos últimos anos para aplicações de IoT.

- As tecnologias IoT apresentam cada vez mais novos produtos disponíveis e novas tecnologias popularizando-se ☺.



#### - Unidades de Processamento:

- O aprendizado de sistemas computacionais embarcados está muito ligado a tudo que gira em torno dos saberes sobre computadores de forma geral. Existe uma diferença ou um processo de “transposição” desse conhecimento para um SO com menos recursos, menos memória, menos poder de processamento. Vamos agora, começar a olhar um pouco mais de perto um SISTEMA COMPUTACIONAL em si, começando pela UNIDADE DE PROCESSAMENTO.

- Essa “coisa” de processamento kkkk, para quem é da computação, já é uma expressão bastante conhecida, para quem é dessa área, é possível que já deva ter escrito pelo menos um software simples ou os visto funcionando em computadores apresentados como [SISTEMAS DE COMPUTAÇÃO](#) possuidores da tal [“UNIDADE CENTRAL DE PROCESSAMENTO”](#) ou [“CPU”](#) que aqui iremos abordar ([Essa é uma forma](#) que vamos chamar de [“clássica”](#) de implementar a unidade de processamento, veremos que existem OUTRAS FORMAS de implementar o processamento de um algoritmo! ).

- Na imagem abaixo são mostradas algumas conhecidas estratégias para se implementar os processamentos dos algoritmos:

- Como aquelas usam de circuitos especializados: [ASIC e ASSP](#) (São duas das formas mais comuns de se fazer isso). Esse tipo de estratégia nada mais é do que [HARDWARES construídos especificamente para executar seus respectivos algoritmos](#). ([Tem melhor desempenho que os processadores!](#))

- Como aquelas que chamamos anteriormente de “[clássicas](#)” ou tradicionais, que são conhecidas pela maioria que trabalha/conhece da área: [Processadores](#) (Muitos acostumam-se com o uso de tal estratégia mas existem sim outras disponíveis como mostrado na imagem abaixo). Diferença para com as estratégias ASIC's ASSP's:

- [Processadores versus ASIC/ASSP](#):

- [No processador](#) se tem muito FLEXIBILIDADE já que somos NÓS que escrevemos o tal do software e esse software é que determina qual é o algoritmo que aquele processador vai implementar.

- [No ASIC/ASSP](#) o algoritmo está IMPLEMENTADO NA FORMA DE CIRCUITO, logo, não há muita flexibilidade em relação ao tipo de aplicação que aquele chip pode implementar ou aonde que ele pode ser utilizado.

- Há ainda uma TERCEIRA ESTRATÉGIA: [Reconfiguráveis](#)(Também conhecidos por [FPGA's](#)). Procuram ficar no “[meio do caminho](#)” oferecendo um [pouco de flexibilidade](#)(como é com os Processadores) já que são programáveis e oferecendo o que os chips dedicados(como os ASICs e ASSP's) podem oferecer que é o [desempenho melhor](#).

**Unidades de processamento**

- ASIC – Application-Specific Circuits (custom logic)
- ASSP – Application Specific Standard Product
  - Alto custo de projeto e de fabricação
- Processadores
  - Flexibilidade, eficiência (mesma lógica para várias funções), simplifica família de produtos
- Reconfiguráveis
  - Alia desempenho do hw dedicado a flexibilidade (e custo) dos processadores

- [Projetando com processadores](#):

## Projetando com Processadores



- Componentes:
  - Software
  - Hardware
- Depuração.
- Teste.

- A imagem acima mostra um panorama do que procurará ser abordado sobre o tema:

- Sobre os componentes desse computador baseado em processadores. Revisitando conceitos conhecidos pelos estudiosos da área de software e hardware.
- Sobre processo de depuração. Quando construimos um projeto(aqui veremos para o caso de computadores embarcados) temos que de alguma forma “validá-lo” e testá-lo.

- Microprocessadores:

## Microprocessadores



- 8 bits (microcontroladores)
  - Aplicações de baixo custo
  - Incluem memória e I/O
- 16 bits (microcontroladores)
  - Aplicações de médio custo
  - Incluem (ou não) memória e I/O
- 32 bits (microprocessadores RISC)
  - Aplicações intensivas em computação
  - Alto desempenho

- “Microprocessador” é o nome mais comum pra se referir a CPU ou Unidade Central de Processamento.
- Quando se fala de processadores embarcados (ou mesmo fora desse mundo dos embarcados) costumamos classificá-los de acordo com a quantidade de bits que a “Unidade Lógica e Aritmética”- ULA desses processadores é construída pra trabalhar. Por exemplo: Se temos um processador cuja ULA trabalha com 8-bits dizemos que é um Processador/Microprocessador de 8-bits e assim por diante.
- Vamos encontrar processadores de 8, 16, 32 ou mais bits. MAS, é mais comum no mundo dos embarcados nós encontrarmos essas 3 famílias iniciais (8, 16 e 32 bits)!

**\*\*\* ( Sendo que os de 8 e 16 bits normalmente chamados de “Microcontroladores” e os de 32 bits ou mais são chamados de “Processadores/Microprocessadores” e normalmente trabalham na forma RISC ) <<< **IMPORTANTE!!!****

- Como que os exemplos da imagem acima se distinguem então:

- Note que há uma certa DISTINÇÃO entre as duas famílias de 8 e 16 bits mas existe uma SUPERPOSIÇÃO muito grande entre essas 3 famílias de modo geral. É claro que há aplicações que podemos implementar tanto utilizando 8-bits quanto 16-bits como da mesma forma podemos encontrar várias aplicações que podem ser implementadas utilizando uma arquitetura 16 bits quanto uma de 32 bits(então note que existe uma “região de sombra” de “aplicabilidade” entre essas 3 famílias podemos assim dizer).
- Aumentar a quantidade de bits normalmente aumenta a quantidade de instruções e o poder dessas instruções o que leva a um processador com gasto de recursos maior que pode entregar um poder de computação maior. Então das 3 famílias abaixo podemos dizer que entre esses 3 mundos de 8, 16 e 32 bits temos na verdade 2 mundos, 1 para 8/16 bits e outro só de 32 bits que já está em outra categoria computacional.
- A briga na indústria hoje tenta cada vez mais aproximar essas 3 famílias umas das outras.

- 8-bits:

- Costumam ser muito PEQUENOS e muito BARATOS.
- Normalmente são voltados para aplicações de MUITO BAIXO CUSTO.

- 16-bits:

- Conseguem oferecer um pouco mais de desempenho. Normalmente isso pode vir a custa de um gasto um pouco maior de energia por exemplo.
- Normalmente são voltados para aplicações de MÉDIO CUSTO.

- 32-bits:

- Costumam vir em aplicações de mais alto custo normalmente.
- Normalmente são voltados para aplicações de MAIS INTENSIVAS em computação(muitas vezes aplicações MULTI-TAREFAS onde se espera alto desempenho).
- ALTO DESEMPENHO.

- Mas porque usar Microprocessadores?



## Porque usar microprocessadores

- Flexibilidade
- Facilita o desenvolvimento de uma família de produtos
  - Vários conjuntos de funcionalidades
  - Fácil de estender e adaptar a novas tendências de mercado
- Eficiência
  - Investimento dos fabricantes é diluídos entre os consumidores

- Um processador é bastante flexível como já sabemos. A mesma CPU/COMPUTADOR pode implementar uma infinidade de aplicações diferentes ( veja que, por exemplo, pra trocar de aplicação nós escrevemos um novo software e o colocamos na memória do computador e ele já está pronto para executar uma nova aplicação, simles e dinamicamente assim! ).

- Com um mesmo microprocessador podemos construir inúmeros produtos diferentes.

- Família de produtos! É muito comum que um fabricante, por exemplo, que tenha um certo produto também tenha outros parecidos dentro da mesma categoria sendo alguns deles com mais recursos e outros com menos. Se desenvolvemos isso em cima de uma plataforma com um microprocessador temos a facilidade de reaproveitar o software desenvolvido para um certa aplicação por exemplo, tornando-a mais completa, mais complexa, que oferecesse mais serviços a quem a adquirisse, assim como, também, poderíamos facilmente tirar algum desses serviços para vender exatamente o contrário, uma aplicação mais simples e acessível(fazendo o cliente conhecer o fabricante ao adquirir um produto mais barato e simples para depois migrar para um mais complexo por exemplo). \*Essa facilidade de alterar o software e aproveitar todo o resto é uma GRANDE VANTAGEM ao se trabalhar com microprocessadores ( ou seja no mercado é comum vermos uma mesma placa de um fabricante em mais de um produto diferente sendo alguns mais complexos e com mais funções e outros mais simples e limitados quanto a usabilidade do microprocessador da placa ). >> **Mas, qual a vantagem disso? Não estamos vendendo/entregando aos clientes hardwares pelos quais eles não estão pagando?**

- \* A resposta seria SIM kkkk. **PORÉM**, a economia de escala é que fala mais alto, ou seja, fabricar uma mesma placa 1 MILHÃO de unidades, por exemplo, sai MUITO MAIS BARATO do que termos DUAS PLACAS DIFERENTES com 500 MIL UNIDADES DE CADA UMA.

- \* Vale mais a pena pro fabricante ter uma escala maior (onde uma mesma placa atende 2,3,4 produtos diferentes e ele GANHA NA ESCALA).

- Família de produtos - Vantagens:

- Desenvolver uma família de produtos é, portanto, outra vantagem quando se fala de processadores já que trazemos a possibilidade de aumentar a ESCALA da nossa fabricação e de aproveitar o software desenvolvido pelos desenvolvedores/eng.software que o fizeram para ser colocado em diferentes

[produtos sem NENHUMA MODIFICAÇÃO](#) já que é o mesmo processador e a mesma arquitetura, sendo apenas uma questão de re-arrumar o código.

- [Fica fácil então de fazer ajustes/reajustes](#) e em algum momento [até criar serviços novos ou funcionalidades novas no produto](#) já que temos tudo do que precisamos de hardware no mesmo cabendo apenas a implementação de um novo software e sua gravação no produto mais uma vez, passando a vendê-lo com mais/menos recursos, maior/menor valor agregado e coisas do tipo.

- ([Eficiência](#)) Existe outra característica interessante quando se fala de processadores que é: ["Diluir o investimento do fabricante do processador"](#). Ou seja, quando se compra um certo processador no mercado se está comprando todo um esforço de engenharia tido para desenvolver tal produto e colocá-lo na prateleira de onde ele foi comprado, isso é um investimento ALTÍSSIMO, porém, todo esse dinheiro usado é DIVIDIDO por CENTENAS DE MILHÕES de processadores. [No fim, acaba que paga-se MUITO POUCO pelo custo que foi desenvolver tal processador](#). Essa é outra vantagem quando se usa processadores, por serem flexíveis e possíveis de se usar várias aplicações diferentes todo o custo pra desenvolver tal processador vai ser dividido com TODAS as aplicações que usarem dele fazendo com que haja uma [GRANDE EFICIÊNCIA NO APROVEITAMENTO DO INVESTIMENTO QUE FOI FEITO PRA DESENVOLVER ESSE PROCESSADOR](#). << ( [Então, a hora que compramos um processador/chip nós estamos dividindo os seus custos com todos que estão interessados nele](#) )-----  
-----> [Não é a toa que a maioria dos produtos embarcados utiliza processadores, então sempre que for possível usar um processador é interessante usá-lo!](#)

- Plataforma (básica) de hardware de um processador:

- Arquitetura de von Neumann(De onde se tira a descrição básica de um computador): “A Arquitetura de von Neumann (de John von Neumann, pronunciado Nóimánn) é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas”.

- Na imagem abaixo temos os componentes que formam essa plataforma:



- CPU(ou Processador, Unidade Central de Processamento).

- Barramento. É um padrão do processador.

- Memória. Haverá memória tanto para os programas quanto para os dados, sendo essas duas memórias DISTINTAS em termos de ARQUITETURA DO COMPUTADOR e elas podem ser tecnologias diferentes, chips diferentes.

- Dispositivos de E/S. No caso de aplicações embarcadas são ESSENCIAIS ([são uma importante chave no desenvolv. de apps embarcados devido a sua diversidade](#)) pois são MUITO DIFERENTES de uma aplicação para outra (cada aplicação tem seus conjuntos de sensores e coisas pra atuar com o mundo a sua volta: Os sensores usados por um Smart Watch são diferentes dos sensores usados por um leitor digital por exemplo).

- SoC("System on Chip"):

- Trata-se de [um sistema computacional em um único chip](#).

- ([Definição](#)) A CPU, o Barramento, as Memórias e, na medida do possível, os Dispositivos de E/S serão TODOS jogados em um chip. Isso sim vai ser uma grande e importante diferença quando se falar de computadores embarcados para computadores de propósito geral(notebooks, desktops), onde [ o sistema computacional não está em um único chip por mais complexa que seja a sua placa(comumente seus componentes computacionais vão estar espalhados pela máquina, haverá vários chips, em casos raros um chip7). ]-----

-----> - Essa forma em que os computadores de propósito geral são fabricados (com os componentes em diferentes chips e localidades na máquina) é chamada de [Arquitetura Discreta](#) ou [Chips Discretos \(Os chips espalhados todos juntos numa mesma placa é que formam o computador\)](#).

- Vantagens:

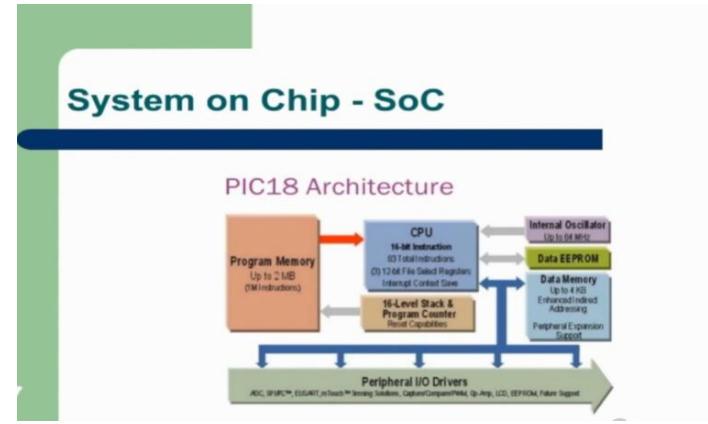
- [“Componentes discretos” versus “SoC”](#):

- Como dito acima já fica claro as muitas vantagens de espaço e custo de fabricação que um tem em relação ao outro.

- [No \(modelo discreto\) todas as conexões existentes entre os componentes computacionais são feitas na placa de circuito impresso o que gera placas com ALTA COMPLEXIDADE DE FABRICAÇÃO \(com várias camadas para viabilizar dezenas de milhares defios e conexões\). << \(DESVANTAGEM\)](#)

- [\\*No \(modelo SoC\) tranfere-se para o fabricante do chip fazer todas as conexões. Já é entregue todas as conexões feitas pelo fabricante do chip, TUDO JÁ ESTÁ CONECTADO E PRONTO. \\*\\*Só iremos precisar conectar apenas as partes onde esses elementos se comunicam com o resto do mundo SE ISSO FOR NECESSÁRIO. << \(VANTAGEM\)](#)

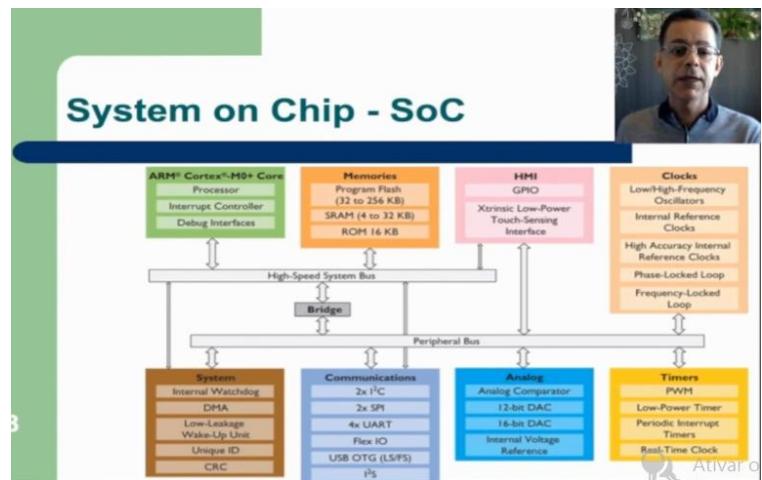
- MUITO MENOS espaço ocupado e CONEXÕES a se fazer.



- A imagem acima trata-se de um exemplo de uma arquitetura SoC de fabricação da Microchip: o PIC18.

- A imagem acima mostra uma versão simplificada da arquitetura SoC do PIC18. Analizando ela podemos relembrar vários elementos clássicos de um computador que estão todos num só chip(há casos de que numa aplicação embarcada haja mais de um chip, auxiliares talvez, mas em essencial sempre o sistema computacional de grande parte das aplicações embarcadas estará compactado num CHIP):

- Program Memory: Memória de programa(tipo de memória).
- CPU(un processador).
- Periféricos de E/S (que no PIC há várias combinações).



- A imagem acima trata-se de um exemplo de uma arquitetura SoC de fabricação de um ARM Cortex M0. A ideia segue a mesma da imagem anterior, tudo num só chip, esse desenho todo está num só chip! 😊

- Depurando sistemas embarcados(TESTES EM SISTEMAS EMBARCADOS):

## Depurando sistemas embarcados

11

- Desafios:
- Sistema alvo pode ser difícil de observar
- ... difícil de controlar
- ... difícil de gerar entradas realistas
- A seqüência de partida pode ser complexa



- Como já dito anteriormente, quando falamos de CONSTRUIR aplicações embarcadas falamos que uma hora em seu processo de desenvolvimento iremos ter que TESTÁ-LO/DEPURÁ-LO para descobrir suas falhas e corrigí-las.

### - Desafios:

- Nosso sistema alvo, aquilo que vai ser nosso produto final, o resultado que é saber se o código de tal sistema funciona ou não pode ser **DIFÍCIL DE OBSERVAR**.

- Quando testamos esse código na interação com sensores efetivamente e pega dados reais do sistema a sua volta e atua nesse sistema real percebemos que a situação fica **MAIS DIFÍCIL**.

- **É DIFÍCIL DE CONTROLAR O SISTEMA QUANDO COMEÇAMOS A QUERER TESTAR A INTERAÇÃO DE TAL SISTEMA EMBARCADO COM SENSORES QUE VÃO COMPOR TAL APLICAÇÃO NO QUAL O SISTEMA FAZ PARTE.**

- **Exemplo:** Tenhamos um sistema de freios ABS sendo feito para um carro(enfim um processador que tem sensores/atuadores que conseguem ler coisas e interferir no sistema de alguma forma e um algoritmo que faz isso). Como que nós testamos se tal código para tal sistema faz o que devia fazer? Como achar entrada características pras funções do código que estamos desenvolvendo? Que tipos de números o freio ABS vai poder computar e dar uma resposta?

-----> [ \***Obs. (não vamos olhar é claro para as operações do código que no meio do caminho podem ser testadas pelo fato das funções nas quais essas pertencerem terem uma saída e uma entrada óbvias e claras a ponto de validarem as unidades internas do software a que pertencem por conta desse princípio de simplicidade).** ]

- **(Seqüência de partida)** As vezes a aplicação precisa que uma série de coisas aconteça antes mesmo dela funcionar. << Como que então levantamos todas essas ocorrências com todos os seus dados produzidos e entregamos para a aplicação? **Veja a complexidade de se fazer isso! Testar pode se tornar algo muito complexo mas EXTREMAMENTE NECESSÁRIO NO MUNDO EMBARCADO, e cada vez mais palestras e teorias/ideias surgem para abordar meios de se responder essa e outras perguntas.**

### - Projeto com Host/Target:

- Quando falamos de desenvolver para SEMBs falamos não só de TESTAR a aplicação embarcada(TÓPICO ANTERIOR A ESSE) como, também, muito antes disso da necessidade de escrever o seu código num AMBIENTE MAIS “CONFORTÁVEL” (estaçao de trabalho/desktop/notebook) e depois gravá-lo em determinada “PLACA ALVO”.

- Exemplo: Ao estar-se desenvolvendo um relógio inteligente para um atleta correr e medir/Registrar coisas em/do seu corpo por exemplo. Não iremos desenvolver esse software no computador que vai literalmente “estar no braço” do usuário que vai estar correndo, nós vamos desenvolvê-lo em máquinas de maior porte e com sistemas bem mais completos (com IDE’s, compiladores e muitas outras ferramentas que nos ajudarão no nosso desenvolvimento e testes) que o do computador que vai estar no relógio inteligente no braço do usuário. Só depois, de tudo isso, se moverá o código para a placa alvo, de preferência no último momento que for possível no seu desenvolvimento, isso, leva a um tempo de desenvolvimento MAIS LONGO. A figura abaixo ilustra esse processo abordado no exemplo:

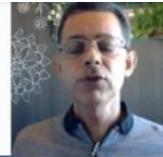


- Existe o sistema “HOST”(hospedeiro) e o sistema “TARGET”(alvo) a placa que é nosso produto embarcado ou o seu primeiro “protótipo”, ambos os sistemas conectados por um canal de comunicação SERIAL. Desenvolvemos o software na “Máquina hospedeira” e gravamos ele na “placa de desenvolvimento” que irá evoluindo após testes e testes até sua versão final como produto.

#### - Depurador in-circuit:

- No contexto abordado no tópico anterior (“Projeto com Host/Target”) chamamos esse processo de gravação e teste de “Depuração In-Circuit” ou “No circuito”, isso pois o processador que está lá na placa de desenvolvimento ou “placa alvo” (“TARGET”) foi instrumentado pelo fabricante com recursos pra que ele possa se comunicar com o software que está no computador “HOST”(hospedeiro) e dessa interação possa haver um teste do software efetivamente na placa e não somente simulado no computador “HOST” utilizando o hardware da “placa alvo” para fazer o software ser executado tranzendo de lá(da placa) algumas informações para checarem o seu comportamento(do software) no ambiente de desenvolvimento (“HOST”).

- [ Podemos testar o software usando processador REAL mas com apoio de uma MÁQUINA DE MAIS RECURSOS onde se é possível inspecionar variáveis, colocar “break points”, fazer testes um pouco mais cuidadoso de partes específicas do código estando ele sendo executado lá da “palca alvo” (“TARGET”)! ] << Já a muito tempo que isso vemsendo o “padrão” quando se fala de desenvolvimento para microcontroladores para sistemas embarcados.



## Depurador In-circuit

- Um microprocessador especialmente instrumentado para depuração
- Permite interromper execução, examinar estado da CPU, modificar registradores.

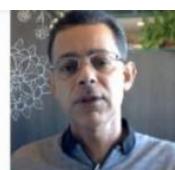
### - Ferramentas típicas:

#### - Cross compiler:

- Um compilador cruzado (inglês: cross compiler) é um compilador que é capaz de produzir código executável para uma plataforma diferente da qual o compilador está sendo executado(O compilador vai cruzar da sua plataforma para outra onde será executado, no “sistema alvo” ou “TARGET”).
- Ou seja, as instruções não são executadas na plataforma (“HOST”), estaremos compilando instruções que estão sendo executadas em OUTRA PLATAFORMA(“TARGET”) daí seu nome “cross” compiler(pois se faz um cruzamento entre plataformas a “HOST” e a “TARGET”).
- Exemplo: Um código é desenvolvido para uma placa que tem um processador PIC nela. Existe um compilador para o PIC que é executado para o computador desktop onde se desenvolve o código, esse compilador traduz de linguagem de alto nível(em C por exemplo) para um código binário que não é executável no computador desktop onde se estar desenvolvendo esse código que é de arquitetura INTEL e sim numa outra plataforma daí motivo dele cruzar de uma plataforma para outra onde será executado.

#### - Cross debugger:

- Temos um depurador que mesmo que tudo esteja acontecendo lá na plataforma alvo(“TARGET”) é no computador(“HOST”) onde temos total controle sobre o processo. Ou seja, as instruções não são executadas na plataforma (“HOST”), estaremos depurando instruções que estão sendo executadas em OUTRA PLATAFORMA daí seu nome “cross” debugger.



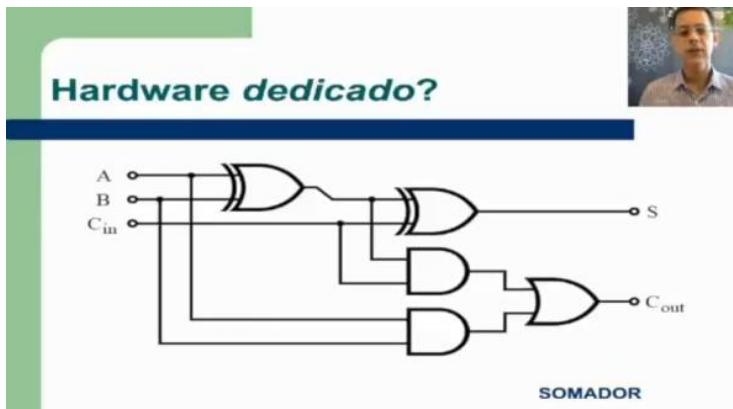
## Ferramentas típicas

- *Cross compiler*
  - Compila código no host para ser usado no sistema alvo
- *Cross debugger (depurador)*
  - Mostra estado do sistema alvo
  - Permite controlar o sistema alvo

14

|||||||||||||||||||||| Unidades de Processamento - Continuações e Aprofundamentos:

### - HARDWARES DEDICADOS:



- A [imagem acima mostra um exemplo de um](#), o [circuito somador](#). O circuito somador da imagem é um “hardware dedicado a somar dois bits” possuidor de suas respectivas entradas (A,B e Cin), vários blocos lógicos responsáveis pela sua computação e saídas que disponibilizarão os resultados quando disponíveis.

- Ou seja, um [hardware dedicado funciona](#): “[Recebendo todas as entradas que tem de receber e as encaminhando por vários blocos lógicos responsáveis por fazer a computação do hardware até que isso se propague em direção a saída do circuito que exibirá os devidos resultados obtidos quando disponíveis.](#)”

- O [tempo entre a disponibilização da entrada e o resultado na saída dependerá da complexidade dos BLOCOS LÓGICOS](#) responsáveis pela computação dos dados no circuito em questão já que [eletronicamente o sinal irá se propagar através de tais PORTAS/BLOCOS até que se obetenha uma saída](#).

- Mas hardwares dedicados não são apenas trabalhados com essa simplicidade, como que, então, eles se extendem ao pensarmos em algoritmos MAIS COMPLEXOS? Como evoluir desse hardware dedicado muito simples para algo mais comparado ao que um software pode fazer em cima de um processador desses de uso comum?

#### - ASIC/ASSP versus MPU:

- Abaixo vejamos como tanto um [PROCESSADOR](#)(representado aqui por “MPU”-“Micro Processor Unit”) quanto um [HARDWARE DEDICADO](#)(tomamos como exemplo as estratégias [ASIC/ASSP](#)) fariam para resolver a equação do delta:

#### - Processador:

- Na imagem abaixo admita uma linha do tempo para o [MPU/Processador](#) e intervalos de tempo consecutivos que representem “ciclos de máquina/execução de instruções”.

- Primeiro, note que qualquer código feito em linguagem de alto nível é traduzido para a linguagem de montagem do processador que é quebrada em intruções mais simples.

- Segundo, note também que esse processado provavelmente terá UMA “ULA(UNIDADE LÓGICA ARITMÉTICA)”. Tal unidade será reusada na linha do tempo da imagem abaixo para fazer cada uma das computações dessa equação do delta.

- ([Passo-a-passo](#)) Por exemplo, poderia-se começar realizando primeiro a operação de “ $a.c$ ” (haveria uma instrução do processador pra fazer isso). Depois pegaria-se o resultado da operação de “ $a.c$ ” e multiplicá-lo por 4. Em seguida se faria “ $b^2$ ” que seria “ $b.b$ ”. Agora, com os resultado de “ $b^2$ ” e “ $4.(a.c)$ ” podemos fazer a devida subtração e obter o valor de DELTA.



## ASIC/ASSP versus MPU

$$\Delta = \frac{b^2}{x} - \frac{4.a.c}{x}$$



### - Hardware Dedicado – ASIC:

- Na imagem abaixo admita uma linha do tempo para o Hardware dedicado-ASIC e intervalos de tempo consecutivos que representem “ciclos de máquina/execução de instruções”.
- Como explicado anteriormente sobre hardwares dedicados, no exemplo da imagem abaixo, vamos ter um bloco para fazer a multiplicação e para fazer essa equação do delta teremos 3 desses blocos além de um bloco para fazer a subtração.
- Vai-se pegar as entradas “b, 4, a, c” e entregá-las direto para esses blocos multiplicadores, eles farão numa primeira onda as multiplicações necessárias e numa segunda onda a subtração é feita em sequência seguida logo depois do resultado do delta.
- Note que diferente do MCU o Hardware Dedicado não se terá em sua “linha do tempo” vários momentos/ciclos de instrução, já tem-se todos os hardwares na quantidade certa necessária para executar o conjunto de operações desejado. Ou seja, entrega-se todas as entradas para o hardware e ele já me entrega o cálculo completo.



## ASIC/ASSP versus MPU

$$\Delta = \frac{b^2}{x} - \frac{4.a.c}{x}$$



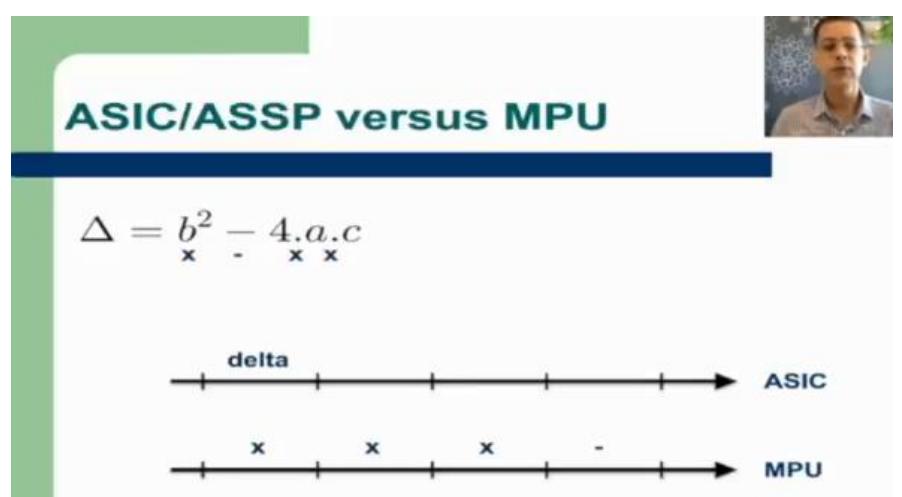
### - ASIC/ASSP versus MPU - Diferenças:

- Notamos entre os dois uma CLARA DIFERENÇA DE TEMPO. O hardware dedicado conseguirá fazer as contas em MENOS TEMPO que o processador uma vez que já terá tudo pronto em seu hardware só esperando as entradas para entregar as saídas não passando por tantos ciclos de máquina como o processador passa.

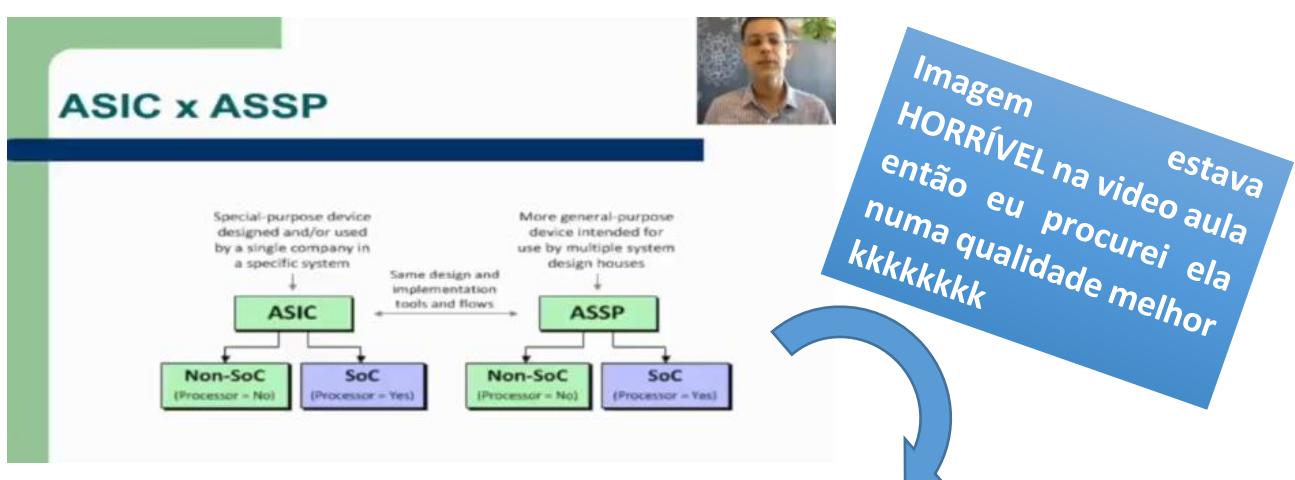
- O hardware dedicado vai ser “maior” em TAMANHO que o processador pois para cada operação que trabalhar precisará de um bloco para realizá-la o que o tornará maior na sua fabricação. Mas veja, e se tivéssemos um caso com 3000 multiplicações? O circuito do hardware dedicado seria 3000 vezes maior? Sim, poderia 😊 / Mas dependendo das situações pode-se fazer um número proporcionalmente menor de blocos de operação e ir reutilizando-os para se diminuir o tamanho da placa fabricada, no caso de 3000 operações pode-se ter 100 blocos de operação e ir reutilizando 30 vezes talvez, isso ainda faria o tempo de computação BEM menor que o de um processador e reduziria o tamanho da placa do hardware dedicado em 30 vezes seu tamanho!

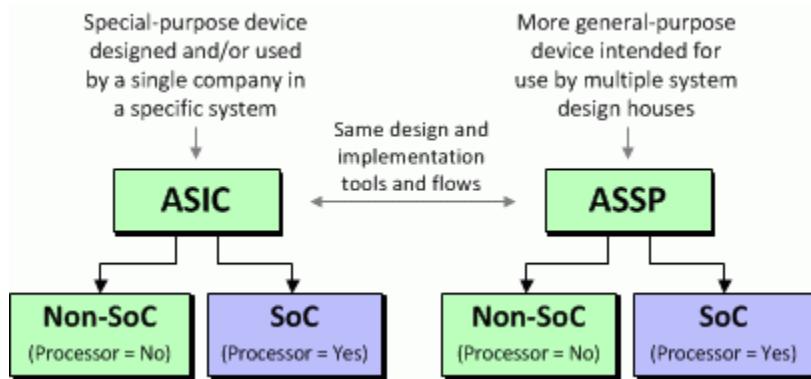
- Há uma relação, também, de custo benefício. O hardware dedicado tende a gastar muito mais rápido do que um processador no entanto ele tende a entregar sua computação num tempo bem menor ainda assim.

- Isso nos mostra as muitas possibilidades entre um hardware que faz tudo(HARDWARE DEDICADO) e um hardware que só faz uma operação de cada vez(PROCESSADOR).



#### - ASIC X ASSP:





- A figura acima mostra um resumo das diferenças entre ASIC e ASSP.
- Ambos os **dois** são **HARDWARES DEDICADOS**(ou **HARDWARES DE PROPÓSITO ESPECÍFICO** ou ainda **HARDWARES ESPECIALMENTE PROJETADOS PARA UMA DETERMINADA FUNÇÃO**).
- **A principal diferença básica** entre os dois é que o ASSP é um produto padrão de mercado que pode estar disponível para a compra de vários fabricantes e o ASIC não é específico de determinado hardware embarcado.
- **ASIC(Circuito de Aplicação Específica):**
  - Foi projetado para uma **ÚNICA COMPANIA** e é utilizado para os produtos específicos dessa compania.
  - Ele **NÃO É COMUM**. Só irá ser encontrado se certos produtos com seu chip literalmente **SOLDADO** naquela placa para aquele propósito. Ou seja, Não está disponível para obtenção comercial e por isso **é mais CARO(pelomenos que os ASSP's)**.
  - Quem compra ele, precisa de grande escala pra conseguir bancar sozinho o custo do seu desenvolvimento, que já foi dito antes: É um pouco mais caro.
- **ASSP(Produto Padrão de Aplicação Específica):**
  - Pode ser aproveitado em **VÁRIOS PROJETOS DIFERENTES**.
  - Está disponível para obtenção comercial.
  - Pouco mais **ACESSÍVEL** que o ASIC financeiramente falando, ou seja, **É MAIS BARATO**. Afinal, ao invés do fabricante disponibilizar certo chip a um **ÚNICO comprador** ele vai disponibilizá-lo comercialmente para **VÁRIOS consumidores**.
  - Em termos de economia de escalar é **mais viável economicamente**. Para fabricantes menores não há a possibilidade de se ter recursos o suficiente para a confecção de um chip próprio, mas há ainda a possibilidade de que um chip parecido possa estar disponível para compra e tenha-se acesso a um hardware dedicado por um custo menor.
  - **Exemplo-01:** Pensando num tocador de MP3, um certo fabricante pode projetar um hardware para tocar MP3 e fazer esse CHIP para usar em seu produto, se a SAMSUNG mandou esse fazer esse chip por exemplo ela o receberá do fabricante, o colocará nos seus produtos e os venderá, ninguém se beneficiará além dela mesma e o fabricante requisitado(ISSO É **ASIC**).
  - **Exemplo-02:** Se um certo fabricante de circuitos integrados fizer um tocador de MP3 e colocar no mercado para venda outros fabricantes de outros equipamentos, ou até a própria

SANSUNG usada no “Exemplo\_01”, podem comprar esse chip e colocar em seus produtos. Esse chip setorna então um “chip dedicado” mas é um padrão de MERCADO, ele só está disponível para outros fabricantes de eletrônicos comprarem e utilizarem(ISSO É ASSP).

- Ambas essas duas formas de fazer chips dedicados(ASIC e ASSP) ainda podem ser encontradas no mercado de outras formas diferentes:

- A imagem lá em cima mostra que ambos podem estar nas formas: SoC(Com processador dentro do circuito) e Não-SoC(Que não contém processador dentro do circuito).

- Um processador USA de um hardware dedicado para que o produto como um todo alcance UMA MAIOR EFICIÊNCIA ENERGÉTICA ou entregue o resultado da computação em MENOR TEMPO.

- ASIC/ASSP-SoC:

- Nessa forma haverá um processador dentro do circuito(Além de todos os elementos computacionais trazidos pelo SoC).

- Haverá integrado ao processador e todos os elementos computacionais trazidos pelo SoC hardwares dedicados. Serão chips que poderão fazer tudo o que um processador comporta mais a função do harware dedicado integrado.

- isso pode tanto ser para um chip proprietário(ASIC) quanto para um acessível comercialmente no mercado(ASSP).

- ASIC/ASSP-Não-SoC:

- Nessa forma não temos um processador dentro do circuito.

- Não haverá processador integrado aos hardwares dedicados para utilizar suas funções específicas ou “dedicadas”. Serão chips limitados a apenas suas funções específicas.

- isso pode tanto ser para um chip proprietário(ASIC) quanto para um acessível comercialmente no mercado(ASSP).

- Exemplo: Chips tocadores de MP3!

**MP3 player**

- VS1011e - MP3 audio decoder
  - 320kbit/s MP3
  - Clk interno 26MHz
  - Consumo médio: 47mA a 2,7V (126,9mW)
- ARM Cortex A53 (Raspberry Pi 3)
  - Clk 1,4GHz
  - Consumo médio: 271mW

Fonte: [www.vlsi.fi/](http://www.vlsi.fi/), 2017

- VS1011e: Tocar MP3 usando de um hardware dedicado(plaquinha especializada em tocar MP3). Mas aqui note que esse hardware tem **suas**

limitações, por exemplo, por ser “dedicado”, caso haja um arquivo em formato desconhecido daqueles que o chip dedicado trabalha ele não vai tocá-los!

- ARM Cortex A53: Tocar MP3 usando de um processador. Aqui, por termos um processador temos mais “liberdade” de flexibilidade lógica, então pode-se ainda lidar com arquivos em formatos diferentes dos que o hardware dedicado conhece!

- Note a diferença de recursos consumidos!

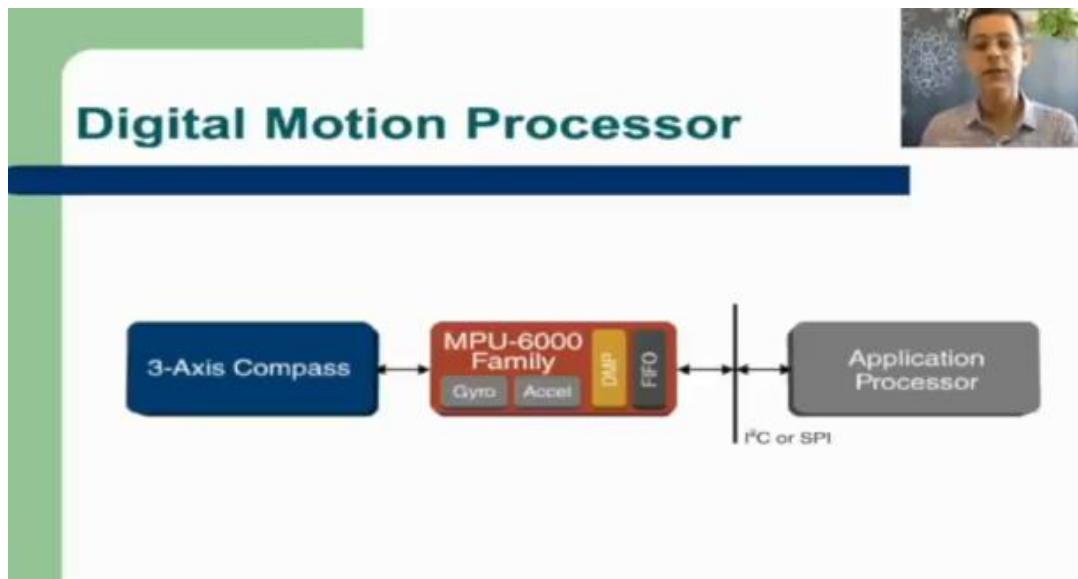
- (\*Conclusões) Basicamente o preço para fabricar ambos esses dois chips é o mesmo(bem como o esforço de engenharia, o tempo de projeto o custo e etc...). A diferença é realmente uma QUESTÃO DE MERCADO.

- A estratégia de se usar um hardware dedicado ao invés de um processador também tem a haver muito com a diminuição da potência consumida(porém não haverá muito uso para o chip além daquilo para o qual ele é “dedicado”).

- Usar um hardware dedicado NÃO TEM A VER COM FLEXIBILIDADE(FLEXIBILIDADE ESTÁ NO PROCESSADOR).

- Processador de movimento digital(“Digital Motion Processor”-DMP) - Outro exemplo de “Uso de Hardwares Dedicados”:

- Modelo-01:



- Na imagem acima vemos um MPU(além do processador principal da aplicação que se comunica através de interfaces de comunicação como SPI/I2C com o MPU que recebe dados de entrada internos pelo 3-Axis Compass). Esse é um sensor “Giroscópio Acelerômetro”.

- Se quisermos utilizar somente o microprocessador para ler os valores do giroscópio e do acelerômetro podemos sim.

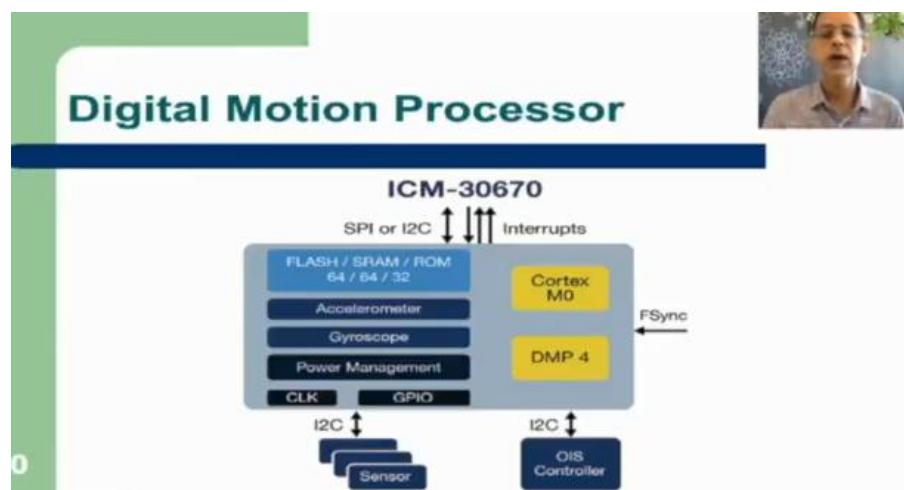
- O DMP é chamado de processador pela sua capacidade de se fazer computação, mas em última análise isso(DMP) segue a mesma ideia dos “hardwares dedicados”

já mencionados anteriormente: É feito especificamente para executar algumas operações utilizando dados do giroscópio e do acelerômetro.

- Ao usar esse sensor “Giroscópio Acelerômetro” tem-se a possibilidade de aproveitar esse hardware dedicado que está lá dentro desse sensor para fazer certas contas e as entregá-las desocupando o processador de fazê-las e diminuindo o custo energético da aplicação, diminuir o esforço computacional do processador principal da aplicação.

- Aqui nesse exemplo temos uma “outra aplicação” na mesma linha de antes só que com o hardware dedicado já estando integrado no mesmo chip com os sensores que ele tem que utilizar(a entrada de dados pra esse hardware dedicado já está lá interno no mesmo chip). Diferente do modelo “tocador de MP3” onde alguém tinha que entregar para o chip tocador de MP3 a sua entrada.

- Modelo-02:



- Esse mesmo modelo de DMP evolui pra uma ideia mais complexa: Quando se tem um hardware dedicado ASIC/ASSP que está dentro de um SoC(Nesse caso será o processador Cortex M0).

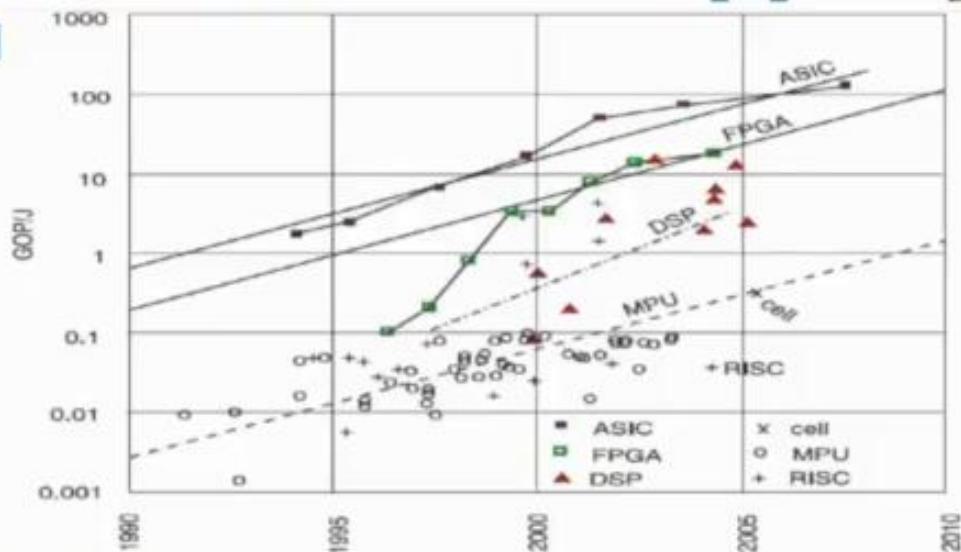
- Nesse caso temos um SoC que tem um hardware dedicado pra computação específica das aplicações que foram pensadas pra ele.

- Mas porque que os fabricantes fazem isso? Porque que o SoC se interessa em oferecer isso? \*Resposta: Porque existe uma GAMA DE APICAÇÕES que precisam medir informações de acelerômetro e giroscópio além da existência de uma caracterização específica em cima desses sensores. Essas aplicações querem utilizar microcontroladores, processadores de baixo poder computacional, que possam tornar a situação dinâmica compacta e acessível a usuário e fabricante.

- O chip desse Modelo-02 já contém a combinação de elementos pra implementar uma aplicação dessa a um baixo custo energético(como um medir de pulso para atletas, que tem um circuito pequeno e simples a ponto de ser colocado em um pulso e ser abastecido por uma bateria muito pequena). É interessante, também, que se tenha um processador para que quem for comprar esse chip possa desenvolver sua aplicação utilizando os dados desses sensores da forma que achar melhor, integrando com outras coisas e gerando um produto mais complexo.

- Eficiência de Hardwares Dedicados – APROFUNDAMENTO – GRÁFICO - 01:

## Eficiência do hardware [2]



Fonte: Marwedel, 2011

- Quando falamos de “hardware dedicado” a tendência das pessoas é pensar: “Mas isso seria algo que empresas grandes da área como a SAMSUNG ou APPLE pensariam em utilizar, pois são grandes fabricantes e poderão dispor de condições de contratar uma fábrica para fazer um chip só para eles e engenheiros de desenvolvimentos como o próprio professor Elias kkkkk, trabalhando no Brasil atualmente para um companhia que faz SEMB's, jamais precisariam se preocupar em utilizar hardwares dedicados nas suas aplicações”. << **Essa interpretação está ERRADA!**, quaisquer engenheiros de produtos morando em locais que trabalhem com a área PODEM e DEVEM utilizar hardwares dedicados em suas soluções.

- (No gráfico acima) **EIXO-X:** Anos em que tais produtos foram desenvolvidos/EIXO-Y: Eficiência Energética.

- No gráfico GOP/J seria quanto de energia se gasta por computação feita em cada tipo de estratégia. Então, QUANTO MAIS ALTO for esse valor de GOP/J quer dizer que MAIOR É A CAPACIDADE DE COMPUTAÇÃO feita com MENOS ENERGIA.

- A figura acima compara as estratégias de hardwares dedicados(ASIC) com as de processador(MPU) e reconfiguráveis(FPGA) para ver como esses se situam em relação a seus gastos energéticos.

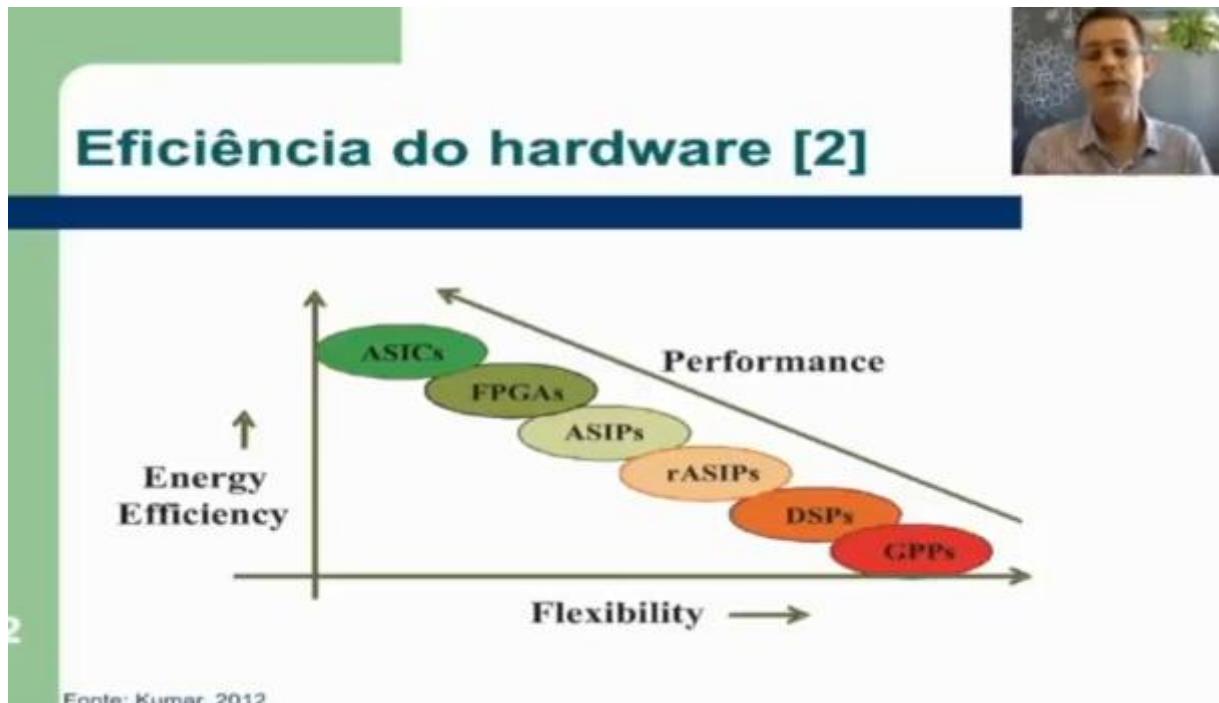
- Note então que o ASIC está no topo com MAIOR EFICIÊNCIA ENERGÉTICA sendo ele um **HARDWARE DEDICADO**. Assim, usar hardwares dedicados oferece maior eficiência energética: Razão pela qual sempre que possível fabricantes deslocam certa computação pra dentro de um hardware dedicados.

- Depende-se da capacidade de fabricantes de semi-condutores conseguirem fazer chips mais eficientes. Note que ao longo do tempo vamos tendo chips com maior capacidade computacional sob mesmos gatos de energia. Até mesmo alguns dos elementos no gráfico estavam acima por um tempo e depois abaixaram (como o processador CELL do playstation 2 que estava bem a frente quando produzido na sua época).

- Do ponto de vista prático, fabricar um chip com tecnologia de 2010 por exemplo é muito **MAIS CARO** do que fabricar um chip de 1990, afinal todas essas tecnologias estão disponíveis hoje cabendo aos fabricantes escolherem quais utilizar.

- Diferentemente de processadores como o CELL e o MPU(que com o passar dos anos vão ficando mais eficientes mas também MUITO MAIS CAROS do que hardwares dedicados), hardwares dedicados tem essa “compatibilidade” quanto a suas eficiencias computacionais, mesmo escolhendo aqueles que dentre eles sejam bem mais antigos(MAIS BARATOS e menos avançados) pode-se dizer que ainda SÃO COMPARÁVEIS aos seus modelos de 2010(mais avançados e MAIS CAROS) chegando até mesmo a ter suas eficiencias energéticas por preços BEM MAIS BARATOS.

**- Eficiência de Hardwares Dedicados – COMPARATIVOS – GRÁFICO - 02:**



- GGP's são “Processadores de propósito geral”. São os mais FLEXÍVEIS no gráfico daimagem acima, um mesmo chip serve para diversas aplicações.
- Um engenheiro ou quem trabalha NO ASSUNTO DEVE SABER QUAL O MELHOR ARRANJO para se usar em determinado produto, e NÃO NECESSARIAMENTE DEVERÁ SER SEMPRE O MESMO, cada produto tem sua própria análise e escolha de quais estratégias de hardware são as MELHORES pra se implementar neste, o que sempre varia de cada produto NÃO ESQUEÇA DISSO!
- Sempre é UMA BUSCA POR COMBINAR COMPONENTES, nem sempre sendo UM desses grupos no gráfico que solucionará TODOS OS PROBLEMAS E REQUISITOS da aplicação embarcada.
- Uma CARACTERÍSTICA MUITO IMPORTANTE dos SEMB's MODERNOS(dos MAIS SIMPLES aos mais COMPLEXOS) é: Sempre é possível se dividir as tarefas internas que tem de ser feitas do sistema em vários chips procurando que cada um deles faça o que tem de ser feito da maneira mais eficiente(muitas vezes hardwares dedicados em um SoC ou não).

**- Eficiência de Hardwares Dedicados – ASIC/ASSP – GRÁFICO – CUSTOS DE DESENVOLVIMENTO:**



- Fabricar chips é muito caro, mas quanto? Observando o exemplo gráfico da imagem acima podemos ver!

- (No gráfico acima) **EIXO-X**: A tecnologia de fabricação(em nanômetros)/**EIXO-Y**: Custo em MILHÕES.

- No gráfico acima, quanto mais a direita(MAIS RÁPIDO e MELHOR em EFICIÊNCIA ENERGÉTICA) MENORES SÃO OS TRANSISTORES EM DETERMINADA TECNOLOGIA DE FABRICAÇÃO. E TRANSISTORES MENORES conseguem oferecer MELHOR EFICIÊNCIA ENERGÉTICA e CONSEGUEM SER MAIS RÁPIDOS.

- **(IMPORTANTE!!!!)**Esse é o caminho que TODO MUNDO procura seguir nesse meio: Levar a computação para HARDWARES ESPECIALIZADOS e com isso AUMENTAR A EFICIÊNCIA ENERGÉTICA e BAIXAR TREMENDAMENTE OS CUSTOS.

- Esses HARDWARES DEDICADOS são MUITO MAIS COMUM em COMPUTAÇÃO EMBARCADA do que em COMPUTAÇÃO DE PROPÓSITO GERAL e principalmente SÃO MUITO MAIS VISÍVEIS. Isso pois a computação embarcada tem um mercado de processadores e de produtos muito grande, há uma necessidade muito grande de aplicações a ser atendidas e, consequentemente, mais fabricantes interessados em produzir hardwares dedicados para resolver problemas específicos(além da necessidade MUITO CLARA em aplicações embarcadas de reduzir custos de fabricação, energia gasta pelo produto, etc..) .

### **- Resumo das diferenças entre ASIC, ASSP, SoC e FPGA:**

<https://www.eetimes.com/asic-assp-soc-fpga-whats-the-difference/#>

#### **- Resumo das diferenças entre CPU, MPU, MCU, SOC, e MCM:**

<https://learningengineering1994.blogspot.com/2017/06/difference-between-cpu-mpu-mcu-soc-and.html>

10. The following table shows the number of hours worked by 1000 employees in a company.

- Aspectos da Arquitetura e da Organização de Processadores - SEMBs:

- Quando falamos de computação embarcada nem sempre o DESEMPENHO MÁXIMO é o PRINCIPAL OBJETIVO. Muitas vezes abre-se mão de um DESEMPENHO MAIS ELEVADO pra poder ter melhores características em relação a potência ou aos custos finais de determinado produto por exemplo.

- Mas então, qual o impacto de coisas como “pipeline” no custo final de um processador ou em sua eficiência energética?



## Quão rápida uma CPU pode ser?

- Pipeline
- Superescalar
- Sistema de memória
- Processadores especializados (ASIP)

- Estratégias como Pipeline, Superescalaridade e Sistema de memória foram desenvolvidas ao longos dos anos com o objetivo de tornar os processadores mais RÁPIDOS. Não se aprofundará muito nessas 3 estratégias e suas propriedades pois entende-se que você já saiba o que são!

- Das estratégias na imagem acima iremos entrar em mais detalhes em coisas como [Processadores especializados\(ASIP\)](#).

- Problemas do pipeline – na produção de um Processador Pipeline:



## Problemas do pipeline

- Desempenho
  - Dependência de dados
  - Instruções de desvio
- Potência
  - Mais unidades operando simultaneamente
  - Lógica de
    - Forwarding
    - Predição de desvio

- Para se produzir um processador pipeline os fabricantes tem que contornar alguns obstáculos para que ele continue executando os mesmos programas que outros processadores podem executar mas oferecendo as vantagens que o pipeline pode oferecer.

- Dependência de dados: Dentre essas dificuldades que os fabricantes tem que contornar para manter o desempenho os nossos programas tem essa característica chamada de “Dependência de dados” onde as variáveis em você escreveu alguma coisa mais adiante você vai precisar ler nessas variáveis.

- Instruções de desvio: Nossos programs tem MUITOS LAÇOS o que faz com que eles executem instruções de desvio.

- Essas duas características(Dependência de dados e Instruções de desvio) fazem com que o hardware de um processador pipeline precise ter algumas funções a mais do que um processador que não implementa essa propriedade.

- Os processadores pipeline possuem MAIS DE UMA UNIDADE que vão se manter em funcionamento ao mesmo tempo. Isso vai fazer com que eles precisem de MAIS POTÊNCIA para funcionar do que um processador equivalente.

- Também será necessário uma POTÊNCIA MAIOR para manter em funcionamento aqueles circuitos que contornam os problemas de desempenho:

- Dependência de dados: É contornada com o forwarding.

- Instruções de desvio(Problemas com desvio): São contornados com hardware de predição de desvio.

- Esses hardwares adicionais (forwarding e predição de desvio) TODO PROCESSADOR PIPELINE VAI TER e vai precisar portanto DISSIPAR UMA POTÊNCIA UM POUCO MAIOR do que um PROCESSADOR QUE NÃO FOSSE PIPELINE.

- Problemas do superescalar – na produção de um Processador Superescalar:

The slide has a green header bar with the title 'Problemas do superescalar'. Below the title is a video feed of a man with glasses and a light-colored shirt. The main content area is white with a blue horizontal bar at the top. It contains two bulleted lists under the heading '• Desempenho' and '• Potência'.

- Desempenho
  - Mesmos do pipeline
  - Execução fora de ordem
- Potência
  - Mesmos do pipeline
  - Várias unidades de execução
  - Hardware responsável pela extração de paralelismo

- Da mesma forma que com processadores pipeline os processadores superescalares também precisam ter hardwares adicionais para viabilizar aquilo que a sua estratégia oferece de vantagem(Que é a possibilidade de executar MAIS DE UMA INSTRUÇÃO SIMULTANEAMENTE).

- Desempenho:

- Vão enfrentar os mesmos problemas de desempenho que o pipeline. Já que processadores superescalares em geral também são pipeline.

- Podem executar instruções numa ordem diferente daquela que foi descrita pelo programador. Pra isso esse processador vai precisar de recursos pra garantir que as lógicas de seus programas NÃO SEJAM COMPROMETIDAS.

- Potência:

- Todos os recursos adicionais de hardware que esse processador demandar para funcionar pedem uma POTÊNCIA MAIOR, assim como no pipeline.

- Terá MAIS DE UMA UNIDADE DE EXECUÇÃO. Se o pipeline tinha uma ULA(Unidade Lógica Aritmética), por exemplo, apenas um processador superescalar terá a capacidade de fazer MAIS DE UMA OPERAÇÃO ARITMÉTICA ao MESMO TEMPO(Terá provavelmente VÁRIAS ULA's). Esses hardwares todos funcionando ao mesmo tempo(As mais de uma unidades de execução, como multiplas ULA's por exemplo) farão naturalmente com que um processador

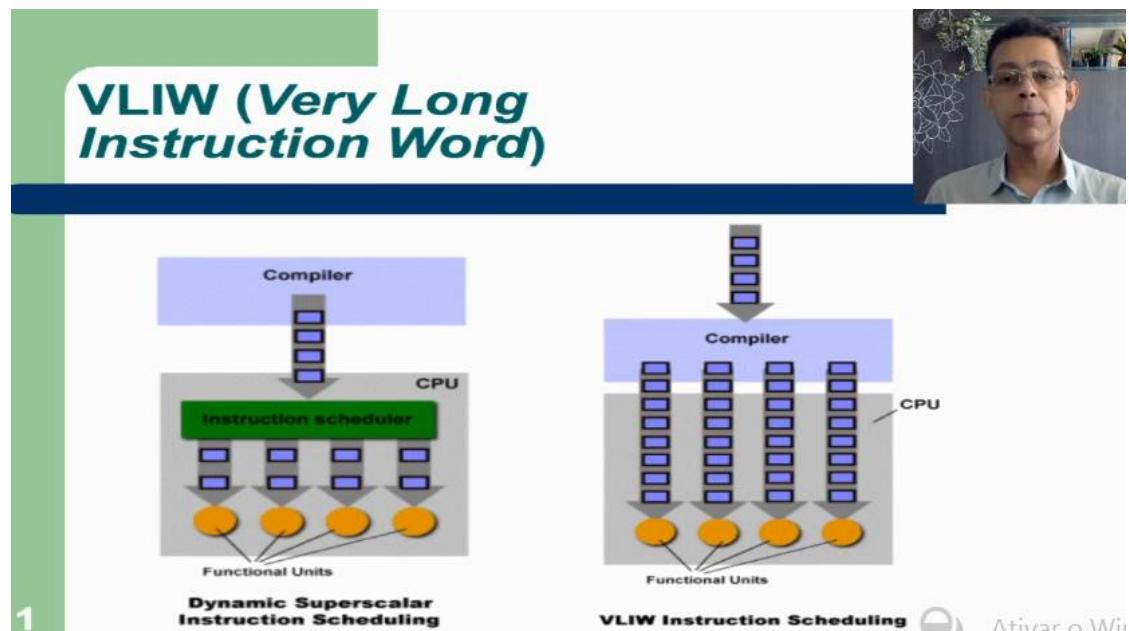
superescalar **DISSIPE MAIS POTÊNCIA** do que um processador que não tenha essa característica.

- O COMPONENTE PRINCIPAL nessa característica de aumentar a potência do processador superescalar é esse hardware responsável pela extração do paralelismo. Já que o processador superescalar consegue executar mais de uma instrução ao mesmo tempo (consegue executar instruções em paralelo) existe um hardware dentro dele que é “capaz de olhar para seu programa” e identificar o que é possível fazer em paralelo em um determinado grupo de instruções nesse programa:

- O hardware que faz isso tende a ser um **hardware CARO** no sentido de ocupar BASTANTE ESPAÇO no **SILÍCIO** do processador e CONSUMIR também uma **POTÊNCIA SIGNIFICATIVA**.

- Alternativas:

- VLIW (“Very Long Instruction Word”):



- Visto como alternativa na estratégia de superescalaridade.
- Modelo DIFERENTE DO CONVENCIONAL.
- TAMBÉM É UM PROCESSADOR SUPERESCALAR só que com a pequena diferença. Note na imagem acima:

- A esquerda temos a superescalaridade “convecional” ou “dinâmica”. É o processador como se conhece comumente (como os da Intel que muitos utilizam). O compilador traduz o código de alto nível para instruções em linguagem de máquina e essas instruções são entregues em ordem para o processador. Lá dentro do processador existe esse hardware verde (como visto na imagem acima a esquerda pelo nome “*Instruction Scheduler*”) que é o Escalonador/Agendador de instruções que é o hardware responsável pela extração do paralelismo (ele quem vai identificar que instruções desse conjunto que tá chegando pra serem executadas pelo processador podem ser executadas em paralelo, e ele é que vai despachar essas instruções para essas “bolinhas” laranjas na imagem acima a esquerda que são as UNIDADES FUNCIONAIS desse processador):

- São as **UNIDADES FUNCIONAIS** que conseguem executar instruções de soma, leitura e memória e as diversas instruções que o processador é capaz de fazer.

- A direita temos a estratégia VLIW. Ele vai fazer o que chamamos de “**ESCALONAMENTO ESTÁTICO**”. Por que estático? Pois é feito em **TEMPO DE COMPILAÇÃO**. Enquanto o superescalar da esquerda faz o dispatcho das instruções para serem executadas em paralelo lá no hardware em **TEMPO DE EXECUÇÃO**, no **VLIW** isso é feito em **TEMPO DE COMPILAÇÃO** e o compilador já entrega as instruções montadas de maneira que elas possam ser executadas em paralelo pelo processador:

- Na imagem acima a direita pode-ser ver que a comunicação entre o compilador e o processador é na forma de um pacote de instruções, daí o nome **VLIW**, que refere-se a “instrução MUITO LONGA”(É como se tal pacote de instruções fosse uma GRANDE INSTRUÇÃO).

- Essa GRANDE INSTRUÇÃO(Ou pacote de instruções) já foi previamente analisado pelo compilador e sabe-se que ele pode ser executado ao MESMO TEMPO por essas **UNIDADES FUNCIONAIS** que estão lá dentro do hardware do processador.

- (Principal característica) Na imagem acima a o que nos “salta aos olhos” na imagem que representa o VLIW é que se comparada ao processador superescalar dinâmico nela não há esse bloco/hardware que faz o agendamento das instruções. Sim, no VLIW não há esse hardware que faz o agendamento das instruções(bloquinho verde na imagem a esquerda), e por isso ele terá um NÍVEL DE AQUECIMENTO MUITO MENOR, uma EFICIÊNCIA ENERGÉTICA MUITO MAIOR, por isso, SE TORNA UMA SOLUÇÃO BEM INTERESSANTE PARA APLICAÇÕES EMBARCADAS.

- Mesmo sendo uma solução interessante para aplicações embarcadas porque não seria interessantes para outras aplicações(Por que que empresas como a INTEL por exemplo não utilizam essas estratégia do VLIW e tornam seus processadores mais eficientes energeticamente?):

- O problema dessa estratégia VLIW é que o compilador deve CONHECER A ARQUITETURA INTERNA do processador. Quando o código é compilado já se sabe quais são as unidades funcionais que esse processador tem e uma vez que foi compilado pra esse processador com essas 4 unidades funcionais esse binário terá dificuldade para ser executado por um novo processador que o fabricante lance que tenha uma quantidade diferente de unidades funcionais. Então aquele código binário está preso aqule hardware, se houver mudanças significativas no hardware o código terá quer compilado novamente, e isso DEFINITIVAMENTE NÃO É INTERESSANTE para você e PRINCIPALMENTE para a INTEL e o seu MERCADO kkkkk(Veja, seria interessante para você quando fosse comprar um computador ter que compilar TODO o seu sofotware novamente? O que na prática seria ter que pagar por eles novamente ou pelo menos acionar o fornecedor pra que ele te entregue um novo código binário e isso possivelmente terá um custo!).

- (MELHOR CASO PARA SE USAR VLIW) Essa estratégia VLIW funciona BEM quando o software e o hardware do computador são vendidos em conjunto, e não há necessidade de separar um do outro.

- (ONDE NÃO SE USAR VLIW - MAS SE USAR SUPERESCALARIDADE DINÂMICA) No caso da arquitetura que tem MAIS SUCESSO HOJE pra computadores como nossos notebooks e desktops acontece uma coisa diferente, você compra um computador de um fornecedor e você vai comprando softwares de acordo com sua necessidade. Esses softwares “comprados” são “descasados” do hardware e eles são trocados independentemente das trocas que você faça no hardware, tal grau de independencia só é possível na SUPERESCALARIDADE DINÂMICA e NÃO na ESTÁTICA(VLIW)!

- (ONDE SE USAR VLIW) Em SISTEMAS EMBARCADOS se compra o equipamento já com o SOFTWARE DENTRO DELE, o que vêm como um CONJUNTO para nós, não precisaremos trocar o software durante a “vida” daquele equipamento e quando for-se querer trocar de hardware não irar-se querer aproveitar esse software anterior. Então, cria-se um CENÁRIO PERFEITO para utilização dessa estratégia de superescalaridade chamada de VLIW!

- Resumindo:

**VLIW (resumo)**

- Máquinas que exploram paralelismo no nível das instruções
- Em uma máquina VLIW, várias instruções independentes são codificadas em uma mesma palavra longa
- Teoricamente, tanto processadores superescalares quanto VLIW conseguem explorar o mesmo paralelismo existente nos programas, apenas com mecanismos e custos diferentes

2

## EPIC – Explicit Parallelism Instruction set Computers

Ativar o Wili

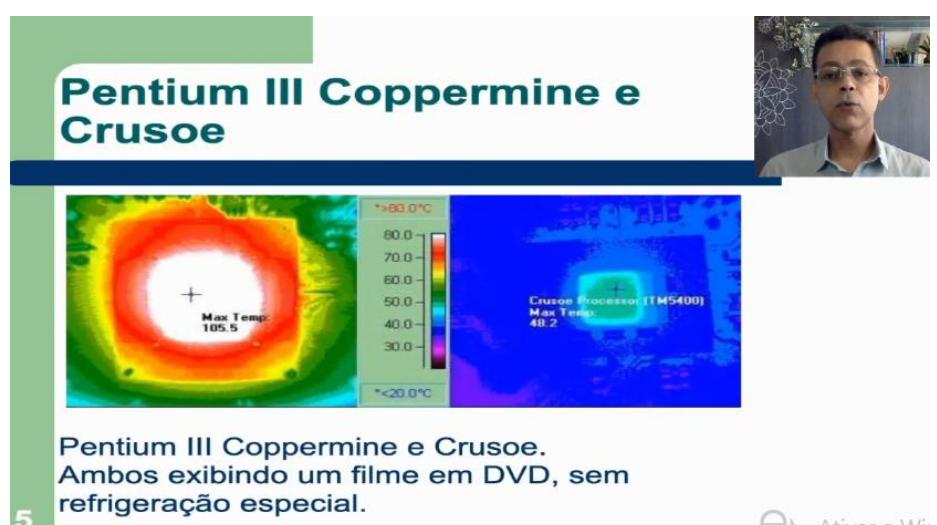
- Processadores VLIW são máquinas SUPERESCALARES.
- São máquinas que EXPLORAM PARALELISMO EM NÍVEL DE INSTRUÇÕES.
- [ Não se confunda pensando que processador superescalar é o mesmo que processador MULTICORE ou PARALELISMO EM NÍVEL DE PROCESSO! ]-----  
-----> [ - O paralelismo desse processadores superescalares é num nível bem baixo(aquelas instruções de máquina que o processador é capaz de executar MAIS DE UMA AO MESMO TEMPO). ]

- TEORICAMENTE, tanto o processador superescalar dinâmico quanto o VLIW conseguem o mesmo GRAU DE DESEMPENHO porque quem limita isso na verdade é o próprio programa/algoritmo que será trabalhado pelos processadores. O que estiver disponível para ser feito em paralelo no

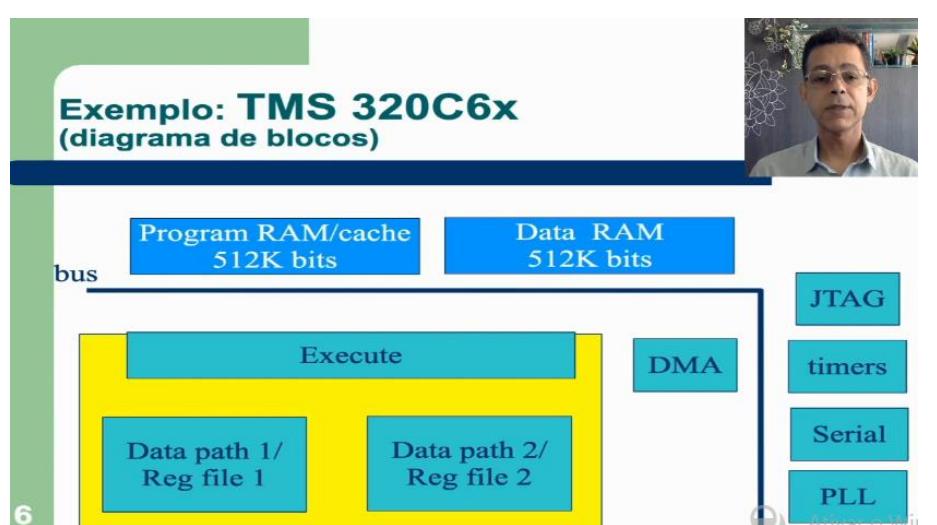
programa ambas as duas estratégias conseguem descobrir, sendo a diferença que uma delas consegue descobrir EM TEMPO DE EXECUÇÃO no hardware(superescalar dinâmica) e a outra descobre EM TEMPO DE COMPILAÇÃO(VLIW- superescalar estático).

- O nome/termo “VLIW” é bastante utilizado e conhecido nas academias e nos livros que falam sobre computação. No MERCADO existe OUTRO NOME que no fim das contas é a mesma coisa que esse termo(falam da mesma coisa): “EPIC”(“Explicitly Parallel Instruction Computing”).

- Na direita da imagem abaixo veja como a TEMPERATURA do processador superescalar estático é bem MAIS BAIXA quando ele trabalha do que o superescalar dinâmico(a esquerda na imagem abaixo), tudo por causa da ausência do hardware de escalonamento de instruções(que torna o processador muito mais “frio” e prolonga a vida das baterias que o alimentam, o exemplo da imagem abaixo a direita é um Crusoe que foi muito usado em NOTEBOOKS exatamente por essa característica):



- Exemplo mais moderno de processador que implementa a estratégia superescalar VLIW(ABAIXO um simplificado diagrama de blocos e suas características resumidas):



## Caminho de dados do 320C6x



- Registradores de propósito geral (A and B, 16 words each).
- Oito unidades funcionais:
  - .L1, .L2, .S1, .S2, .M1, .M2, .D1, .D2
- Duas unidades load (LD1, LD2).
- Duas unidades store (ST1, ST2).
- Dois registradores *cross paths* (1X and 2X).
- Dois operadores dado-endereço (DA1 and DA2).

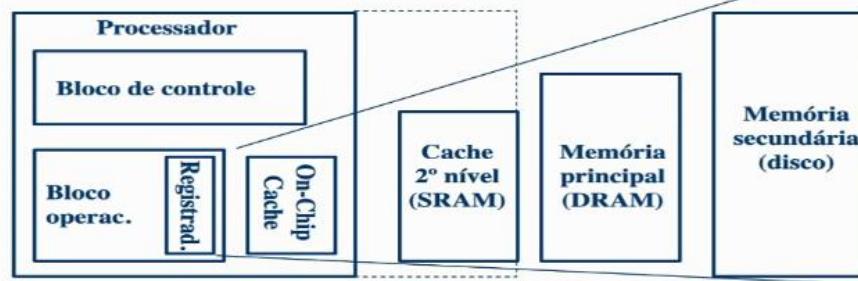
7

Ativar no Wii

- Na segunda imagem acima atente-se ao que já foi dito sobre esses processadores que implementam a estratégia superescalar VLIW sobre possuirem várias UNIDADES FUNCIONAIS fazendo coisas diferentes, esse por exemplo tem 8 UNIDADES(Não sendo iguais pelas suas nomenclaturas diferentes apesar de se haver repetição de TIPOS de UNIDADES como M1 e M2 que ambas são multiplicadoras)!

- Hierarquia de memória:

## Hierarquia de memória



2

Velocidade (s): 0,1n 1n 2-5n 10-20n 10 m .  
Tamanho (bytes): 100 16 K 512 K 256 M G

Ativar no Wii

- Esse é o nome geral para incluir memória VIRTUAL e CACHE.

- Na figura acima vai-se desde a memória que está DENTRO do processador até as mais externas a máquina:

- No nível MAIS INTERNO do processador há os Registradores (Onde as instruções são executadas) subindo pelo nível de CACHE INICIAL chegando até o DISCO.

- Note que a imagem não é recente kkkkk. Mas veja que quando é com os registradores e CACHES DE NÍVEL 1 a quantidade de memória tipicamente gasta são de valores baixos, essas

quantidades de memória vão aumentando a medida que vamos nos afastando do processador.

- Nos dias de hoje é claro que os computadores tem bem mais memória na memória principal do que na figura acima.

- (JUSTIFICATIVA) Olhando também para a VELOCIDADE vemos que não houve um GANHO TÃO GRANDE quanto na CAPACIDADE DE ARMAZENAMENTO(Mesmo na hierarquia das memórias). Ao longo do tempo as memórias não crescem tão rapidamente na VELOCIDADE mas crescem muito rapidamente na CAPACIDADE DE ARMAZENAMENTO. Por isso que existe a hierarquia de memória, porque os processadores sim crescem muito em relação a VELOCIDADE e pra não se ficar preso a uma memória que não é tão rápida existe essa ideia de hierarquia.

- (IDEIA) Na hierarquia as memórias mais PRÓXIMAS dos PROCESSADORES são MAIS RÁPIDAS e existe um hardware(que não aparece na figura acima) que procura levar para as memórias mais próximas dos processadores aqueles dados/instruções que estão sendo necessárias nesse momento da execução do programa, assim, quando uma CPU precisar de um dado ou instrução existirá a chance desse dado já estar nas memórias mais próximas da CPU que estão no TOPO da hierarquia de memória.

- Problemas do sistema de memória:

The slide has a green header bar with the title 'Problemas do sistema de memória'. Below the title is a blue horizontal bar. On the right side of the slide, there is a small portrait of a man with glasses. In the bottom left corner, there is a large number '3'. At the bottom right, there is a small icon and the text 'Ativar o Wii'.

- Desempenho
  - **Prejuízo por falha de Cache**: tempo extra devido ao cache miss.
  - Caches introduzem indeterminismo no tempo de execução. (aplicações de tempo-real).
- Potência
  - Lógica de gerenciamento (MMU)

- Algumas características levam a **MEMÓRIA CACHE** a criar problemas aos projetistas.

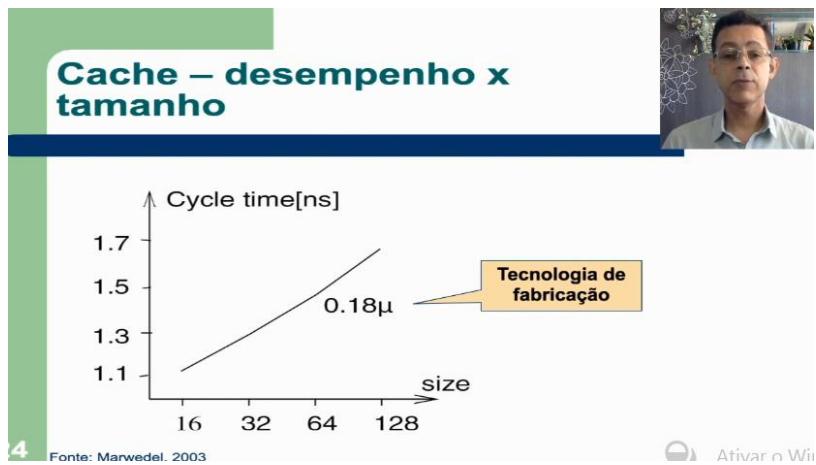
- [ Nem sempre um dado que um processador precisa vai estar na CACHE, as vezes ele não está lá. Quando isso acontece existe um tempo extra que a CPU tem que esperar pra que o sistema traga pra memória mais próxima dele aquela informação que poder ser uma instrução/um dado: ]-----

-----> - (Problema - DESEMPENHO) Não é muito fácil de prever quando isso acontece. É uma situação indeterminada. Como isso não é previsível aquelas aplicações onde o TEMPO DE EXECUÇÃO da CPU é CRÍTICO (Aplicações de tempo real) a memória CACHE pode ser mais problema do que solução, porque em algumas situações a execução de um trecho de código pode demorar 10 vezes mais do que acontece com muita frequência e essa variação pode levar a um mau funcionamento da aplicação.

- (Problema - POTÊNCIA) Existe esse hardware para gerenciar a memória cache e a hierarquia de memórias de modo geral chamado: MMU. Esse hardware vai estar no sistema computacional, se

não tivesse CACHE ele não estaria lá, portanto a sua POTÊNCIA é um ADICIONAL que o computador vai precisar consumir porque tem HIERARQUIA DE MEMÓRIA IMPLEMENTADA.

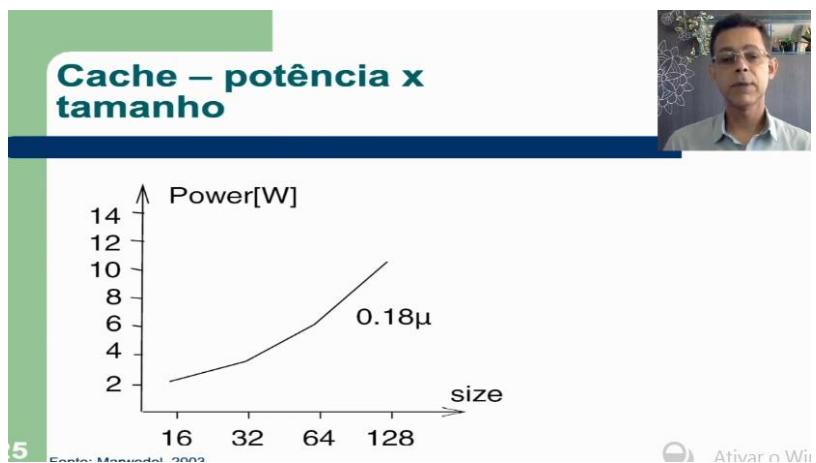
- CACHE – Análise gráfica – Desempenho x Tamanho:



- Interpretações do gráfico da imagem acima:

- Quando fazemos uma comparação entre o TAMANHO DA CACHE(eixo X: "size") e o seu DESEMPENHO(eixo Y: "Cycle time") notamos que CACHES MAIORES tendem a ser MAIS LENTAS do que CACHES MENORES.
- MAIOR O TAMANHO DA CACHE – MAIOR SEU TEMPO DE RESPOSTA/DESEMPENHO.
- Se fabricarmos uma CACHE com a mesma tecnologia por trás de uma outra que é menor utilizando os mesmos transistores que respondem na mesma velocidade porém com um tamanho maior ela passará a ter um tempo de resposta MAIOR do que a CACHE de tamanho menor. Isso explica por que que as CACHES do topo da hierarquia de memória são CACHES pequenas, além de consumirem potências MENORES elas são MAIS RÁPIDAS.

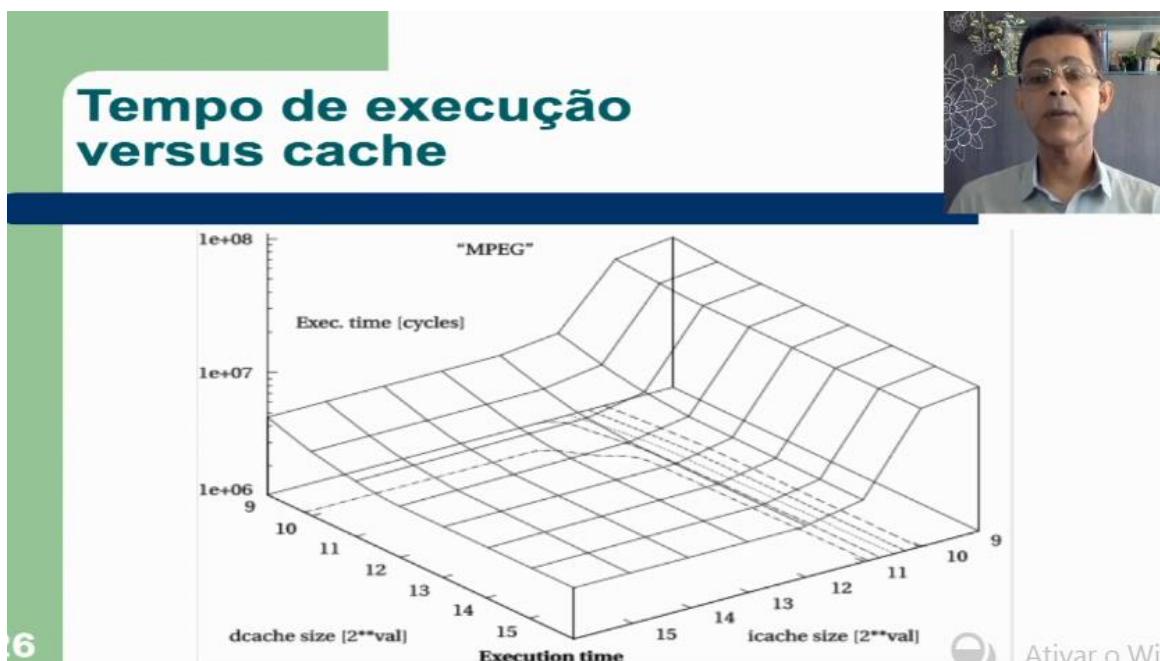
- CACHE – Análise gráfica – Potência x Tamanho:



- A mesma comparação feita de TAMANHO DE CACHE com VELOCIDADE também pode ser feito comparando TAMANHO DE CACHE com a POTÊNCIA DISSIPADA sendo que agora a variação é praticamente quadrática.

- MAIOR O TAMANHO DA CACHE – MAIOR SUA POTÊNCIA DISSIPADA em ordem QUADRÁTICA.

- CACHE – Análise gráfica – *Tempo de execução x CACHE*:



- Como já dito antes, não precisa-se ter uma CACHE muito GRANDE para se ter o MELHOR DESEMPENHO POSSÍVEL.

- [ Veja que no "icache size" no gráfico da figura acima começando pelo valor mais baixo de tamanho que é  $2^9$  o tempo de execução é um, depois que esse valor aumenta pra  $2^{10}$  o valor dá uma diminuída mais muito pouco para se perceber, após ir para  $2^{11}$  há uma QUEDA SIGNIFICATIVA no tempo de execução e daí pra frente( $2^{12}$ ) ao aumentar-se o tamanho da CACHE usada é praticamente irrelevante, ela ainda continuará a diminuir mas não será perceptível de forma evidente como foi de  $2^{10}$  para  $2^{11}$ . ]-----

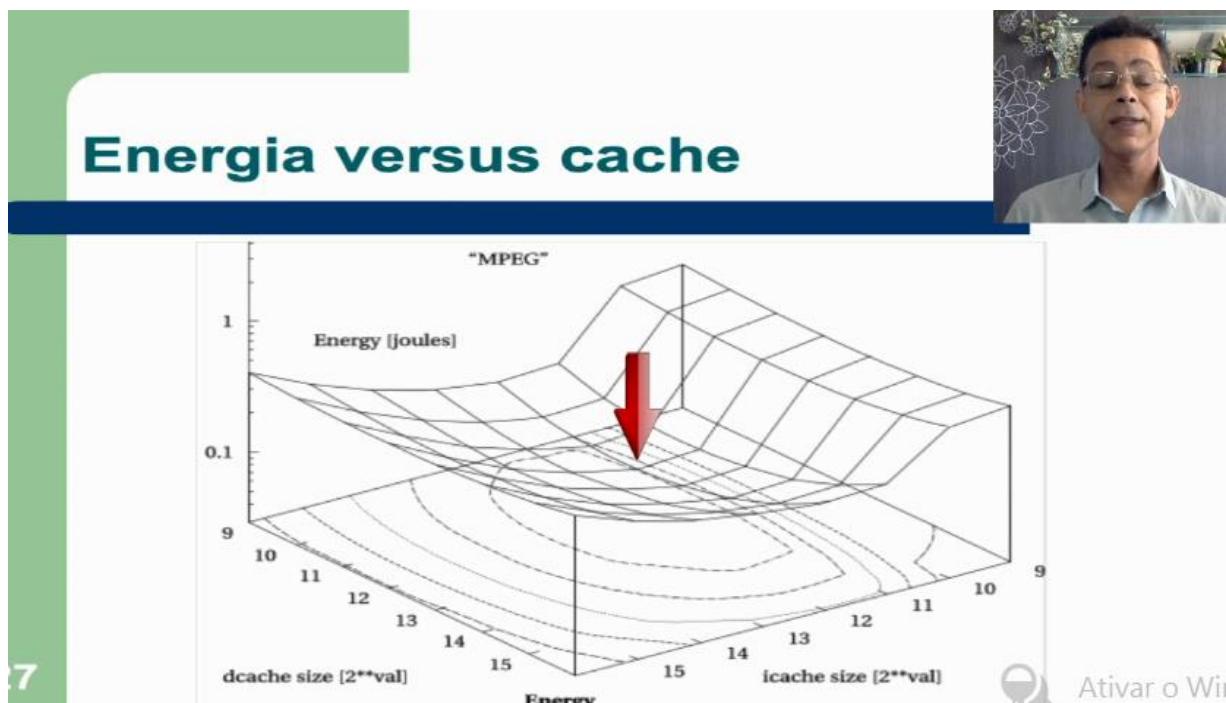
-----> - Ou seja, existe um tamanho de CACHE que é SATISFATÓRIO para a aplicação teste “MPEG” usada nos resultados do gráfico da imagem acima, e daí pra frente não adianta mais AUMENTAR o tamanho da CACHE(o mesmo é preceptível para o "dcache size", note que quando chega no  $2^{12}$  já não adianta mais aumento de tamanho).

-----> - Essa é uma característica interessante de se explorar nos processadores embarcados porque sabendo-se antecipadamente quais são os algoritmos que irão utilizar determinado processador o fabricante pode CALIBRAR o TAMANHO DA CACHE para as suas necessidades sem ter que EXAGERAR no tamanho daquela memória. \*Uma facilidade como essa um fabricante como a INTEL, por exemplo, não tem, afinal, quando fabricantes como ela fazem processadores para seus computadores eles não tem como saber a priori qual é o tipo de algoritmo que será usado com

maior frequência dos seus usuários(varia de um para outro),  
eles então fazem TESTES DE DESEMPENHO mas acabam por  
pegar números estimados que serão satisfatórios para a  
maioria dos usuários.

- Aumentar o tamanho da CACHE causa AUMENTO do desempenho até um certo momento em que não precisa-se mais aumentar(fica constante). << (esse fenômeno acontece tanto pra CACHE DE INSTRUÇÕES quanto pra CACHE DE DADOS).

- CACHE – Análise gráfica – Energia x CACHE:



- Um gráfico semelhante ao apresentado anteriormente para o mesmo algoritmo MPEG, agora, avaliando a ENERGIA.

- Note que existe um “PONTO ÓTIMO” (Seta vermelha no gráfico acima) em relação a ENERGIA quando se trata de TAMANHO DE CACHE pra que tire a melhor EFICIÊNCIA ENERGÉTICA do sistema. **(PROBLEMA): Isso também depende do ALGORITMO kkkkk, o gráfico da imagem acima por exemplo só vale para o MPEG, com outros algoritmos a configuração do gráfico é outra e seu “ponto ótimo/baixo” iria mudar.** Então, novamente um fabricante de um sistema embarcado que conhece previamente qual o algoritmo que vai funcionar no seu produto TEM UMA VANTAGEM COMPETITIVA EM RELAÇÃO AO PROCESADOR DE USO GERAL(Ele consegue ali, sabendo que tipo de algoritmo aquele equipamento vai precisar executar, escolher uma combinação melhor de tamanho de CACHE). → Exemplo:[ Imagine um processador que foi desenvolvido para equipar um SMART TV, já se sabe mais ou menos quais os algoritmos que vão ser executados nesse equipamento então assim o fabricante pode ajustar o tamanho de sua CACHE para essas devidas necessidades com maior precisão tanto ENERGÉTICA quanto COMPUTACIONAL/DESEMPENHO ]

- Pra energia notamos que ela vai BAIXANDO quando se AUMENTA o tamanho da CACHE chegando uma hora em que VOLTA a AUMENTAR DE NOVO(esse fenômeno acontece tanto pra CACHE DE INSTRUÇÕES quanto pra CACHE DE DADOS, embora seja muito mais perceptível na de INSTRUÇÕES quanto na de DADOS).

- (Revisão) Essas estratégias até aqui abordadas foram desenvolvidas para **QUALQUER COMPUTADOR** e foram muito focadas em **AUMENTAR A VELOCIDADE E O DESEMPENHO**(melhorar) dos computadores. Quando falamos dessas tecnologias no contexto “embarcado” é preciso refletir sobre “**POTÊNCIA DISSIPADA**” e “**CUSTOS**” para que tenhamos soluções mais adequadas aos objetivos de cada produto embarcado, o que nem sempre é somente uma questão de **MELHOR DESEMPENHO**(O computador embarcado como já dito antes pretende ser **MAIS** enxuto e **MAIS** otimizado em certas características deixando o desempenho em segundo plano).

- Buscando aumentar a eficiência/desempenho de um sistema computacional mais estratégias foram desenvolvidas visando ou a **EFICIÊNCIA DO CÓDIGO**(que visam basicamente reduzir os gastos com memória do nosso sistema computacional) ou o uso de **INSTRUÇÕES ESPECÍFICAS**(que otimizam a memória, reduzindo seu gasto, melhorando o desempenho do seu SO). A imagem abaixo resume essas ideias que veremos agora:

**Buscando aumentar a eficiência (de um sistema baseado em MCU)**

- Eficiência no código
  - Reduzir gastos com memória
- Instruções específicas (ASIP)
  - Memória e desempenho

- Tamanho do código:

**Tamanho do código**

- Máquinas CISC (*Complex Instruction Set Processors*)
  - RISC x CISC
- Técnicas de compressão
  - 2º conjunto de instruções
    - Requer compilador específico
  - Dicionários
    - Hardware “tradutor”

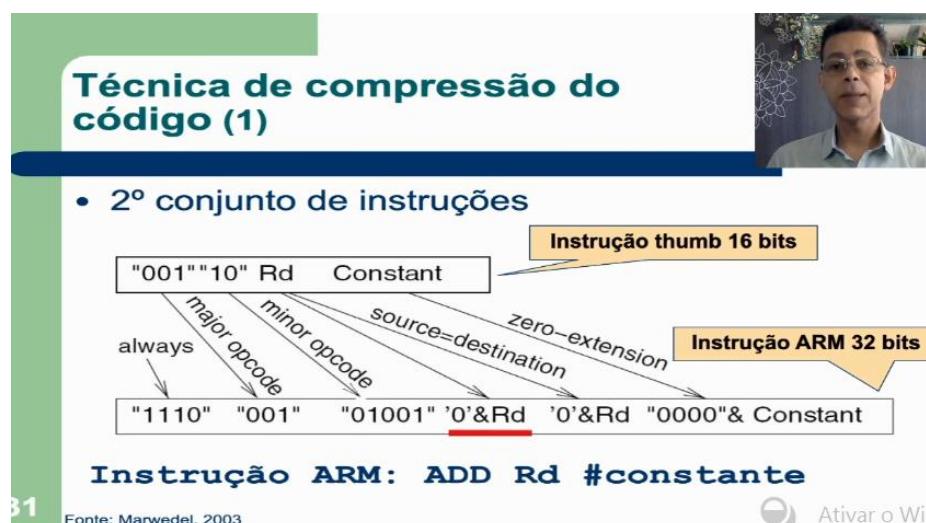
- **CISC**: Pensando em tamanho de código esse conceito CISC, que é o mais antigo de projeto de processadores, tinha exatamente isso como objetivo ---> O código deveria ser pequeno porque naquela época as memórias eram CARAS e LENTAS, então, era interessante que as memórias fossem assim. +Quando usar um CISC hoje em dia? >> \*(Alguns **PROJETOS MAIS ANTIGOS DE PROCESSADORES** ainda continuam utilizando essa estratégia, o que pode ser uma opção se seu projeto precisa reduzir a memória de código que ele utiliza. USAR UM PROCESSADOR CISC pode ser uma solução interessante numa aplicação embarcada de BAIXO CUSTO onde a MEMÓRIA precisa ser MUITO PEQUENA)

- RISC: Com o passar dos anos essa estratégia RISC acabou por dominar o mercado pois essa restrição do custo de memória deixou de ser um problema importante(que na época “CISC” era um grande obstáculo). O processador RISC oferece um DESEMPENHO muito MELHOR com BOA EFICIÊNCIA ENERGÉTICA, assim, acabou por SUBSTITUIR os processadores CISC.

- As técnicas complementares nessa linha de “DIMINUIR O TAMANHO DO CÓDIGO” são chamadas de TÉCNICAS DE COMPRESSÃO:

- DICIONÁRIOS: Estratégia de SUBSTITUIR O CONJUNTO PRINCIPAL por um CONJUNTO ALTERNATIVO introduzindo um HARDWARE que faz a TRADUÇÃO entre o CONJUNTO COMPACTO e o CONJUNTO ORIGINAL do PROCESSADOR.

- Técnica de compressão do código-2º conjunto de instruções:



- Como TÉCNICA DE COMPRESSÃO DE DADOS vamos pegar o exemplo mais popular: O processador ARM(Esse é de longe o PROCESSADOR MAIS USADO em APLICAÇÕES EMBARCADAS).

- Eles(ARM) propuseram um conjunto de instruções chamadas de “thumb”(esse conjunto de instruções tem palavras de 16 bits). As instruções ORIGINAIS do ARM são de 32 bits, então, ao se propor esse conjunto alternativo “thumb” para nosso processador ARM em nosso exemplo o mesmo vai passar a usar instruções que tem METADE do TAMANHO das instruções originais dele(que no caso eram 32 bits, agora, passariam a ser de 16 bits) o que é um BELO GANHO(muito bom, ótimo kkkk), acontece que ainda, assim, não vamos conseguir que nosso programa tenha METADE do TAMANHO do PROGRAMA ORIGINAL!

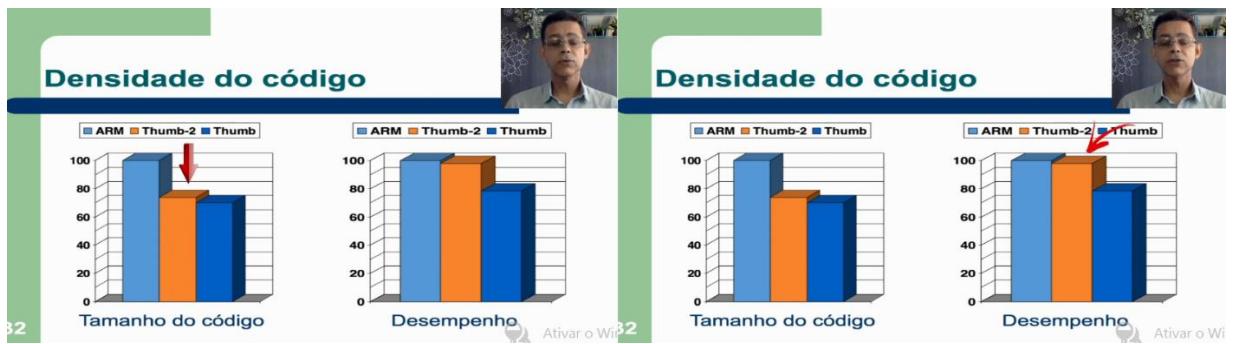
- Olhando para a imagem acima pode-se ver que algumas coisas são perdidas quando passamos uma instrução de 32 bits(onde se pode representar MUITO MAIS INFORMAÇÕES para uma instrução menor) para 16 bits(onde algumas coisas não conseguem **mais** serem representadas, como era com 32 bits). Exemplo(na imagem acima):

[ - Um processador ARM é originalmente de 32 bits, ou seja, tem 32 REGISTRADORES o que demanda 5 bits na PALAVRA DE INSTRUÇÃO para representar essa informação. Na instrução compacta(aqui sendo: “Instrução thumb 16 bits”) um desses 5 bits é perdido fazendo com que o compilador fique preso ao utilizar METADE dos REGISTRADORES disponíveis na arquitetura do processador. ]

-----> - Isso explica porque que executar instruções no modo

**THUMB** nos leva a ter **DESEMPENHO INFERIOR** e a **NÃO CONSEGUIR EXATAMENTE METADE DO TAMANHO** do que seria usando 32 bits. < (**PROBLEMAS AO SE USAR THUMB**)\*

- Estudo-Densidade do código:



- A imagem acima é um estudo comparando **TAMANHO DO CÓDIGO** e **DESEMPENHO**.
- Note que a ARM tem **DUAS VERSÕES** para o padrão **THUMB**: Thumb(barra azul escuro nos gráficos da imagem acima) e Thumb-2(barra laranja nos gráficos da imagem acima). Thumb-2 é uma **OTIMIZAÇÃO** da Thumb.
- Olhando a imagem acima notamos que ao utilizar a primeira versão do padrão Thumb, chamado aqui por nós de “Thumb” apenas(barra azul escuro nos gráficos da imagem acima), temos uma **REDUÇÃO de cerca de 30%**, ou seja, ele passa a ter 70%~ do tamanho original.
- Quando usamos a segunda versão criada pela ARM que é o Thumb-2(barra laranja nos gráficos da imagem acima) vemos que esta não possuí o MESMO NÍVEL de compactação de código da Thumb(primeira versão) e ainda perde-se um pouquinho(tanto que a barra laranja no gráfico do tamanho é MENOR que a barra do ARM mas ainda é MAIOR um pouquinho que a barra do Thumb, passando a ter 75%~ do tamanho original que é a barra do ARM).
- Agora, quando comparamos o gráfico do **DESEMPENHO** entendemos porque que pode ser interessante usar a versão Thumb-2 que mesmo não tendo o “grau de compressão”(redução do tamanho do código original) igual ou maior que a primeira versão Thumb ela possui QUASE O MESMO DESEMPENHO da INSTRUÇÃO ORIGINAL ARM.
- Usando A VERSÃO Thumb-2 temos uma **boa redução do tamanho do código**(mesmo não sendo a máxima possível) e conseguimos praticamente o mesmo desempenho quando se utilizava o código ORIGINAL do processador ARM.
- A maioria das pessoas que trabalham na área hoje tende a utilizar o Thumb-2 a menos que se trabalhem com aplicações onde o DESEMPENHO não seja algo MUITO IMPORTANTE mas a compactação do código(redução do código original) deva ser a **MÁXIMA POSSÍVEL**.

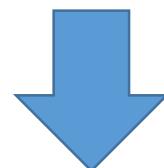
- ASIP:

## ASIP (*Application Specific Instruction set Processor*)



- O conjunto de instruções (ISA) de um ASIP é estabelecido de modo a beneficiar uma determinada aplicação.
- Caso mais conhecido – DSP: *Digital Signal Processor*

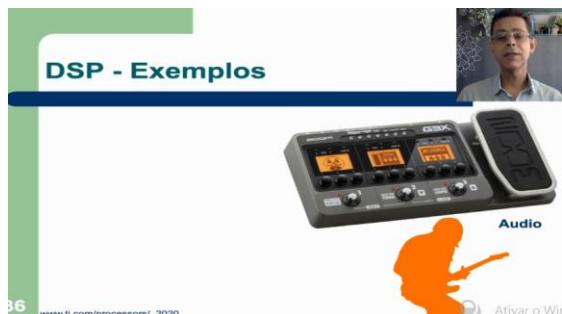
- **ASIP:** É uma forma de elaborar as instruções do processador pensando em um certo domínio de aplicações. Nesse caso o fabricante propõe uma arquitetura de instruções que tenha instruções que favoreçam determinado tipo de algoritmo.



- Sabendo o que determinados tipos de algoritmo precisam para ter um bom desempenho, são criadas instruções e, obviamente, um hardware por trás para executar essas instruções que vai viabilizar um desempenho melhor para aquele grupo de aplicações.

- A implementação MAIS CONHECIDA dessa estratégia ASIP é o DSP que é especializado em PROCESSAMENTO DE SINAIS. Alguns exemplos de DSP para melhor familiarização:

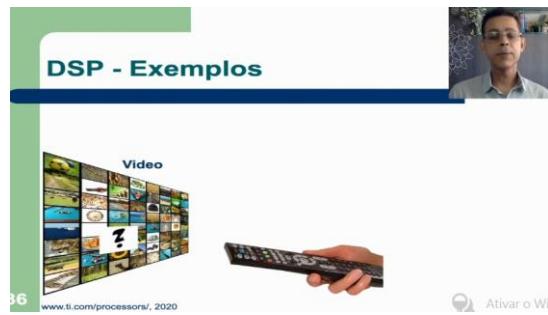
- DSP's(Exemplos-Audio):



- Em muitos shows é comum hoje em dia músicos tocadores de instrumentos(principalmente de baixo e contra-baixo) possuírem em seus pés conjuntos de equipamentos parecidos com o da imagem acima, comumente chamada de “pedaleira”(pois o músico comanda isso com seus pés). Esse equipamento consegue produzir efeitos especiais no som de certos instrumentos, são dos mais diversos tipos esses efeitos que podem ser configurados e acionados no palco em tempo de performance.

- (Uso do DSP na área) Para se fazer essas alterações que a “pedaleira” faz no som do instrumento utiliza-se de algoritmos de processamentos de sinais(já que o som que vem da guitarra é um sinal ☺). Esse equipamento, também, é do tipo digital (como pode-se perceber kkkk) o que quer dizer que dentro dele existem processadores do tipo DSP que fazem essa alteração no som para o guitarrista conseguir o efeito que quer fazer em sua música.

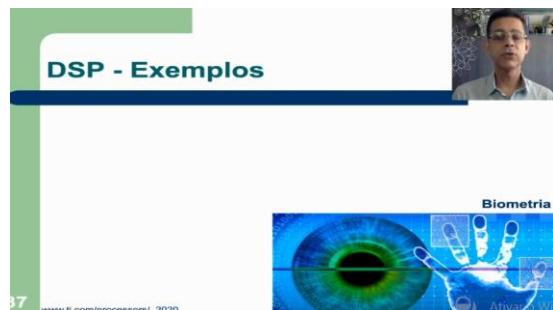
- DSP's(Exemplos-Vídeo):



- (Uso do DSP na área) São aplicados em vídeo também. Quando se assiste vídeos que estão compactados em NPEG por exemplo, é necessário que um algoritmo de “descompressão de vírus” seja aplicado, esse algoritmo utiliza de estratégias de processamento de sinais e para que TV's DIGITAIS consigam fazer isso com BOM DESEMPENHO é necessário o uso DSP's por essas.

- DSP's(Exemplos-Biometria):

- (Uso do DSP na área) Pode ser utilizado para procurar por padrões nas imagens de retina ou de impressão digital por exemplo.



- DSP's(Exemplos-Dispositivos de proteção):



- (Uso do DSP na área) Podem fazer parte de aplicações voltadas para proteção de instalações. Substações elétricas e máquinas industriais podem ser supervisionadas, e os algoritmos que vão detectar eventuais falhas ou o potencial de progredir para uma falha em alguma situação vão utilizar processamento de sinais que podem ser baseados em ÁUDIO, ULTRASONS, VIBRAÇÃO, TEMPERATURAS, IMAGENS e outras informações que podem entrar para esses algoritmos para que eles processem esses sinais e ajudem especialistas a diagnosticar potenciais de certas falhas ocorrerem numa determinada instalação o que faz com que tais algoritmos funcionem como forma de proteção dessas estruturas.

- DSP's(Exemplos-Defesa):

**DSP - Exemplos**




38 [www.ti.com/processors/](http://www.ti.com/processors/), 2020

Ativar o Wii

- (Uso do DSP na área) Há aplicações na área de defesa. Radares baseados em emissão de frequências específicas que são captadas e processadas por esses sistemas.

- DSP's(Exemplos-Carros autônomos):

**DSP - Exemplos**



Carro autônomo

38 [www.ti.com/processors/](http://www.ti.com/processors/), 2020

Ativar o Wii

- (Uso do DSP na área) São aplicações que estão muito na moda hoje em dia. São baseados muito no processamento de imagens, DSP's podem ser úteis para fazer essa computação em tempo mais curto e conseguir dar respostas em tempo apropriado.

- DSP(características):

### DSP - características



- Requer muitas operações matemáticas
- As operações devem ser realizadas em um tempo delimitado

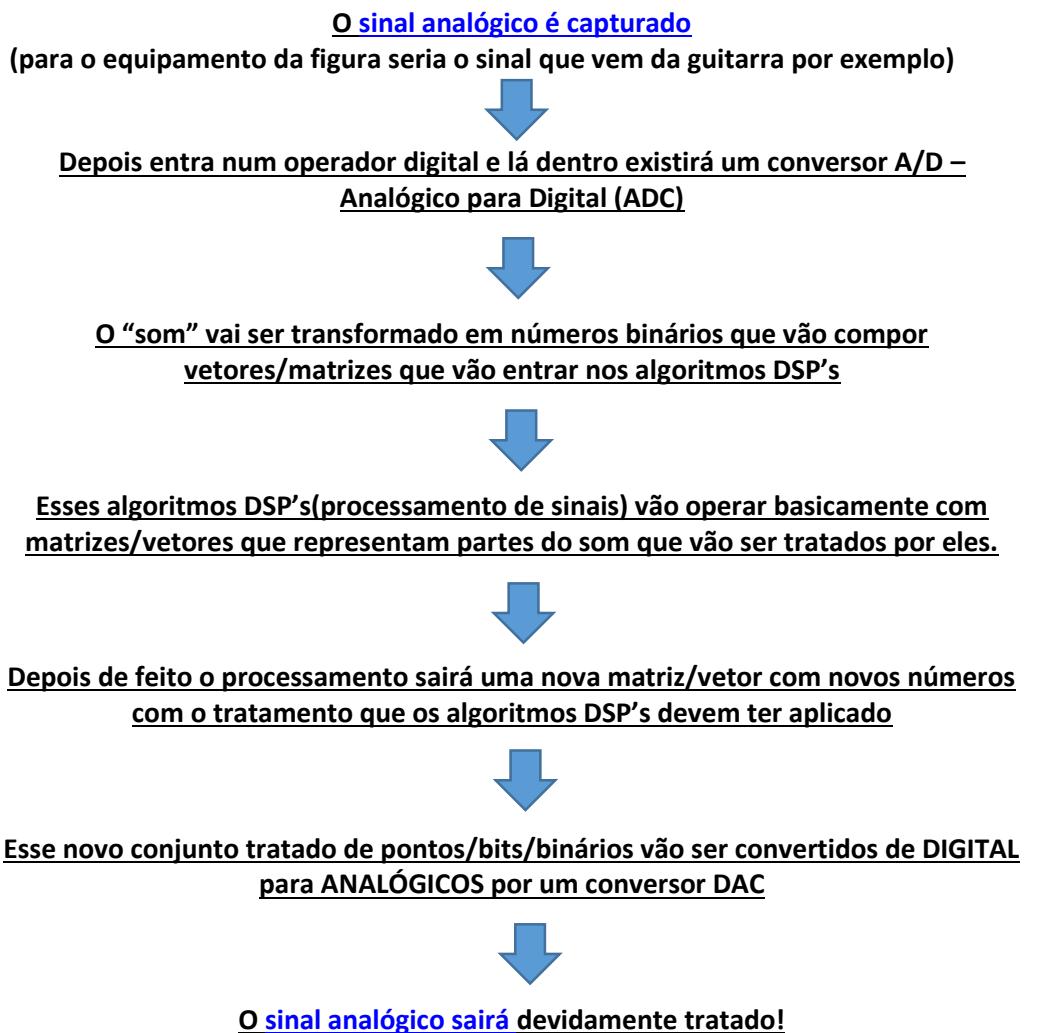


Ativar o Wii

- Quando se fala de processamento de sinais tem toda uma área da engenharia voltada para estudar isso. Qual a importância de se ter um processador que execute esses algoritmos DSP de forma eficiente?

- Tipicamente, aplicações DSP's(de Processamento de sinais) seguem um fluxo parecido com o da imagem acima. O equipamento da figura acima é a "irmã mais velha" da pedaleira mostrada num dos exemplos práticos de DSP's anteriormente tendo um funcionamento parecido com ela porém analógico e mais rústico por assim

dizer. Equipamentos como esse que vêm antes daquela pedaleira digital para serem substituídos pelosmodelos digitais que hoje em dia conhecemos vão passar por um “fluxo” como o apresentado na figura acima:



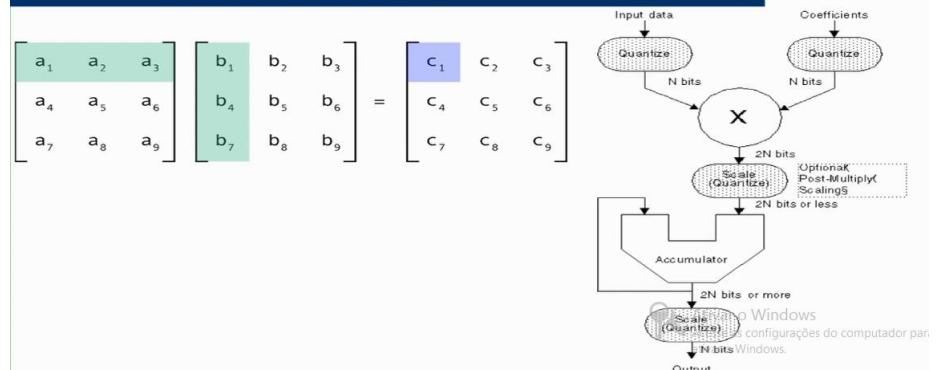
- Eses algoritmos DSP’s requerem uma certa complexidade de computação. Existem grandes quantidades de operações a serem feitas num grande volume de dados e isso requer um esforço computacional significativo.
- Esse grande esforço computacional tem que ser feito num CURTO TEMPO para que NÃO HAJA PERDA DE INFORMAÇÃO:
  - Pegando o exemplo da guitarra e a pisadeira, a guitarra está continuamente produzindo som então a cada instante novas matrizes/vetores estão chegando para serem processadas pelo processador DSP, se ele não conseguir dar conta da matriz/vetor anterior ele não conseguirá processar a próxima e se isso acontece o som que deveria sair na saída do equipamento não chegará lá pois a CPU não TERMINOU de fazer suas OPERAÇÕES, tem-se então problemas de CORTE/FALHAS no som.
  - Funcionamento de DSP’s – Exemplo: “Multiplicação de matrizes”

## Exemplo: Multiplicação de matrizes

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

- Com esse exemplo clássico de multiplicação de matrizes conseguiremos entender o que é que esse processador DSP tem pra fazer que é específico dos algoritmos de processamento de sinais e que terá um desempenho melhor que um processador convencional.

## Exemplo: Multiplicação de matrizes



- Como estamos falando de multiplicar duas matrizes 3x3 começa-se pelo algoritmo clássico da matemática: Multiplicar os elementos da primeira linha da primeira matriz pelos elementos da primeira coluna da segunda matriz, depois SOMAR os resultados de cada uma dessas multiplicações gerando o resultado final numa “matriz solução/resultado”.

- Note que existe um processo de MULTIPLICAR TERMOS e SOMAR o RESULTADO de CADA UM desses TERMOS, isso chama-se “Multiplica e Acumula”. O esquema a direita na imagem acima é um HARDWARE feito exatamente para executar esse processo conhecido como “Multiplica e Acumula”:

- Tem-se DUAS ENTRADAS de DADOS por onde virão os dois vetores que tem de ser multiplicados.
- Tem-se um HARDWARE que fará a a MULTIPLICAÇÃO desses termos(os dois vetores da entrada).
- O primeiro conjunto que é multiplicado cai e chega no “ACUMULADOR”(que nada mais é do que um SOMADOR). A primeira multiplicação desce e é ACUMULADA, quando vem a SEGUNDA multiplicação ela desce e é SOMADA com a MULTIPLICAÇÃO ANTERIOR e assim se segue acumulando essas somas até que no fim desse laço temos a saída que é o valor de um dos elementos da

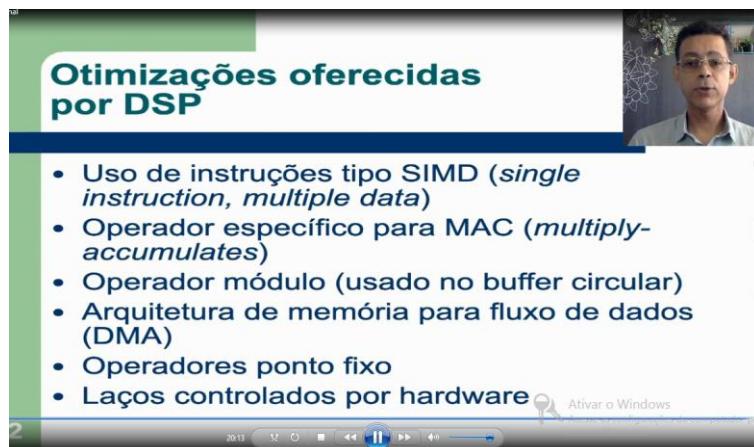
**MATRIZ RESULTADO/SOLUÇÃO** da multiplicação. O processo se repete até achar TODOS OS ELEMENTOS da MATRIZ SOLUÇÃO.

- É exatamente esse hardware(que executa o processo “Multiplica e Acumula”) que é o **DIFERENCIAL do DSP** em relação a um **processador NORMAL**. Já nota-se a possibilidade de se fazer uma MULTIPLICAÇÃO E UMA SOMA tendo um HARDWARE PARA SOMAR e um HARDWARE PARA MULTIPLICA no DSP.

- (**Diferenciável do DSP pro PROCESSADOR COMUM - 01**) O DSP consegue CONTROLAR SOZINHO o LAÇO que PERCORRE TODO O VETOR. O vetor vai chegando nas duas entradas e o próprio hardware controla a quantidade de leituras que ele mesmo vai ter fazer baseado no TAMANHO DA MATRIZ. Esse hardware do DSP consegue ELE MESMO percorrer a primeira linha e a primeira coluna da matriz entrada e já entregar o resultado do elemento resultante na matriz solução PRONTO, tudo isso sem ter que fazer sucessivas operações de leitura na memória, gravação de resultados intermediários até que se finalmente tenha-se o resultado do elemento resultante na matriz solução(isso quem faz é o PROCESSADOR COMUM de forma MUITO DESGASTANTE E REPETITIVA).

- (**Diferenciável do DSP pro PROCESSADOR COMUM - 02**) Numa CPU/PROCESSADOR COMUM tem-se uma ULA, então ela precisa ter que pegar o resultado da multiplicação, guardar em algum lugar para fazer a soma no MESMO HARDWARE em que se fez a multiplicação. Com isso PERDE-SE TEMPO!

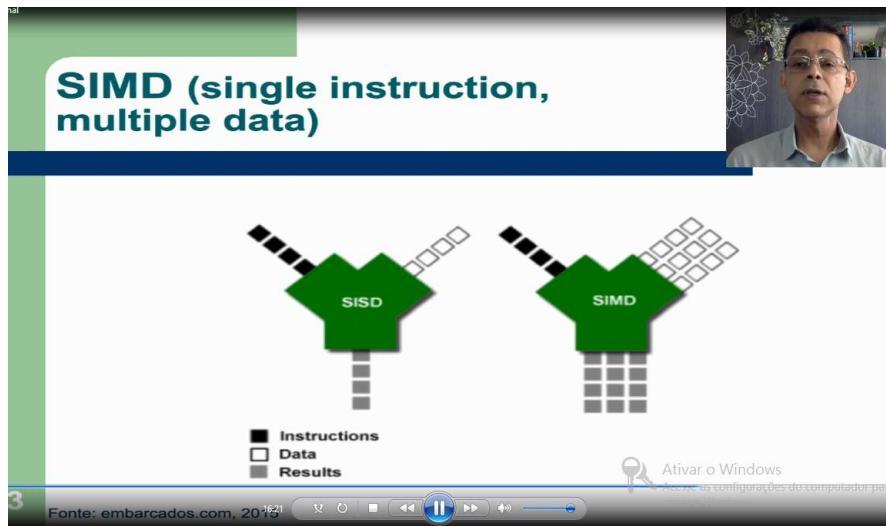
- **DSP(otimizações)**:



- A figura que vimos anteriormente(na multiplicação de matrizes com o esquema do hardware de funcionamento dos DSP's) é o que acontece no hardware do DSP para que ele consiga fazer a EXECUÇÃO da MULTIPLICAÇÃO de MATRIZES de FORMA EFICIENTE.

- Do ponto de vista do software existe essa instrução chamada SIMD e é essa instrução que vai servir de INTERFACE entre nós PROGRAMADORES e o HARDWARE que está lá dentro do DSP que faz a computação acontecer.

- **SIMD(Tipos)**:



- A figura acima compara os DOIS TIPOS de instrução que queremos abordar.

- **(SISD)** Do lado esquerdo da figura acima temos uma instrução NORMAL. Ela é buscada na memória, ela busca o seu dado, ela faz sua computação e ela entrega o seu resultado. É uma instrução do tipo ESCALAR porque UMA INSTRUÇÃO OPERA em CIMA de UM DADO e ENTREGA UM RESULTADO.

- **\*\*\*(SIMD)** Do lado direito da figura acima temos uma instrução VETORIAL. É um TIPO DE INSTRUÇÃO que o PROCESSADOR DSP VAI UTILIZAR. Uma instrução é carregada mas essa instrução é capaz de LER um CONJUNTO DE DADOS e entregar um CONJUNTO DE RESULTADOS. Então, uma instrução do tipo SIMD consegue utilizar aquele hardware(que tem nos DSP's de "Multiplica e Acumula") para fazer TODA A MULTIPLICAÇÃO DE MATRIZ, vamos informar pra ela(instrução SIMD) onde começa na memória uma das matrizes a serem multiplicadas, onde começa na memória a outra matriz e onde começa o lugar onde deve ser colocado o resultado, também deve-se informar o TAMANHO dessas matrizes, feito isso a instrução SIMD de MULTIPLICAR MATRIZES começa a rodar e vai buscando TODOS OS DADOS e vai CONTROLANDO CADA UM DOS LAÇOS e vai entregando CADA UM DOS RESULTADOS, quando essa instrução acabar de executar a MATRIZ RESULTADO já está pronta e a CPU está pronta para buscar a próxima instrução. Isso dá um GANHO DE DESEMPENHO “ASTRONÔMICO” kkkk, porque COMPARADO COM a instrução “for” clássica, utilizada em programação de alto nível, ali(instrução “for”) existe todo um trabalho(que não é requerido na instrução SIMD) de ficar “lendo resultado” e “gravando resultados parciais na memória” até que se chegue no final desse laço [além da incrementação de ponteiros, decrementação de variáveis(que são contadores) e testar se estas chegaramnos limites].

- Além dessas instruções SISD/SIMD feitas para operações vetoriais os processadores DSP vão implementar essa lógica já explicada de “Multiplica e Acumula”-MAC(“multiply-accumulates”). Aquele hardware nos DSP's faz as multiplicações e acumula o resultado, sendo ele a “ESSÊNCIA” do que torna um processador DSP mais eficiente para esses algoritmos de multiplicação de matrizes do que um PROCESSADOR CONVENCIONAL.

- Esse processador DSP, também, vai ter outras funcionalidades que são próprias desse processo de EXECUÇÃO de INSTRUÇÕES EM BLOCO como:

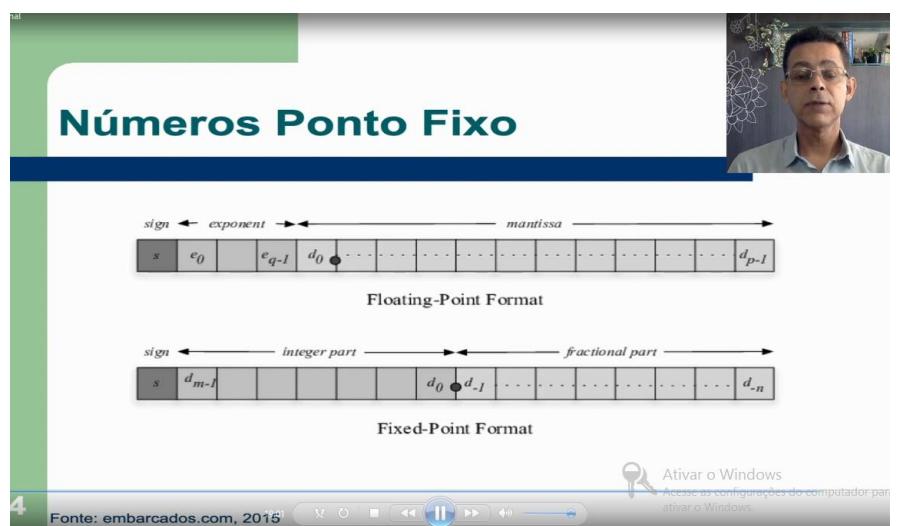
- Operador módulo: Ajuda na implementação da estrutura do buffer circular.

- **Hardware de acesso a memória**: Buscar essas matrizes que vão ser os operandos de uma maneira MAIS EFICIENTE. Então, ao invés de o processador IR NA MEMÓRIA buscar CADA ELEMENTO DA MATRIZ existe um HARDWARE chamado DMA que fará o ACESSO DIRETO A MEMÓRIA(ele é que traz essas matrizes pra entrada do processador e elas são entregues UM BYTE atrás do outro sem que a CPU tenha que executar uma instrução de leitura na memória pra buscar cada um dos operandos, na HORA CERTA eles vão chegando pra que sejam multiplicados pelo OPERADOR MAC - “*Multiplica - Acumula*”).

- **Números de Ponto Fixo**: É outra estratégia que ajuda DSP's a terem um desempenho elevado, a capacidade de operar com pontos fixos:

- Ponto flutuante, Ponto fixo, essas são representações de dados, as duas feitas para representarem NÚMEROS NÃO INTEIROS e o DSP PODE UTILIZAR AS DUAS FORMAS para fazer a sua conta.

- Mesmo podendo usar dessas duas formas, ao se usar PONTO FIXO o DSP conseguirá fazer CONTAS MAIS RAPIDAMENTE do que utilizando PONTO FLUTUANTE:



- A imagem acima mostra uma representação desses dois TIPOS DE DADOS.

- O PONTO FLUTUANTE é aquela ideia de NOTAÇÃO CIENTÍFICA que todos nós estudamos(talvez kkkkk) lá na matemática. Temos uma região para representar a “mantissa” e uma para se “armazenar o expoente”:

- (VANTAGENS) Tem-se a capacidade de representar NÚMEROS MUITO GRANDES por se ter um EXPOENTE MUITO FLEXÍVEL.

- (VANTAGENS) Pode-se representar NÚMEROS MUITO PEQUENOS utilizando-se de EXPOENTES NEGATIVOS.

- (VANTAGENS) Pode-se representar NÚMEROS MUITO GRANDES utilizando-se de EXPOENTES POSITIVOS.

- No PONTO FIXO se estabelece um local onde ficaria o “ponto” ou “vírgula”, sendo a parte a esquerda desse ponto/vírgula será a PARTE INTEIRA e a parte a direita desse ponto/vírgula será a PARTE FRACIONÁRIA:

- (LIMITAÇÕES) O PONTO FIXO tem uma LIMITAÇÃO quanto ao TAMANHO DO NÚMERO que PODE REPRESENTAR.

- (LIMITAÇÕES) NÃO PODE-SE REPRESENTAR NÚMEROS MUITO PEQUENOS, tem-se uma certa LIMITAÇÃO de CASAS DECIMAS para representar o número escolhido. Da mesma forma a PARTE INTEIRA também NÃO PODE SER MUITO GRANDE devidos a limitações que também são impostas.

- (SOLUÇÃO) Na prática o que se utiliza tendo tantas limitações são números entre -1 e 1000. Ou seja, NÃO EXISTE PARTE INTEIRA, todos os BITS são utilizados para a PARTE FRACIONÁRIA.

- Laços controlados por hardware: As intruções SIMD, ELAS PRÓPRIAS CONTROLAM o laço. Elas mesmas que fazem o processo de DECREMENTAR a variável de controle, testar se chegou no zero e prosseguir com o laço ou não.

- TODAS ESSAS CARACTERÍSTICAS COMBINADAS[instruções SIMD(Tipos), lógica “Multiplica e Acumula”-MAC(“multiply-accumulates”), Operador módulo, Hardware de acesso a memória, Números de Ponto Fixo, Laços controlados por hardware] fazem com que o PROCESSADOR DSP consiga um DESEMPENHO MUITO SUPERIOR ao PROCESSADOR CONVENCIONAL.



- [ENERGIA \(em SEMB's\)](#):

- Perguntas importantes:

- Como se pode avaliar a ENERGIA CONSUMIDA por um computador?
- E como podemos modificar um PROJETO de um computador para que ele GASTE MENOS ENERGIA e suas BATERIAS DUREM MAIS?
- Como evitar que sua MÁQUINA esquente TANTO e possa ser mais confortável de se utilizar por seus usuários (chegar mais perto do "corpo" das pessoas - ( ¸◡¸ ) - por exemplo)?

- Eficiência energética:



## Eficiência energética

- Vários processadores são projetados levando em conta o “consumo de energia”.
- Potência versus Energia
  - $P = V \cdot I$  ou  $E/t$  (taxa de transferência de energia)
  - $E = P \cdot t$

- Essas questões ligadas a ENERGIA do COMPUTADOR estão muito ligadas a EFICIÊNCIA ENERGÉTICA (quando escolhemos uma plataforma para fazer nosso projeto de computador pensando em COMO ELE VAI UTILIZAR SUA ENERGIA!).

- Consumo de ENERGIA passou a ser uma importante questão nos projetos de fabricação dos COMPUTADORES hoje em dia.

- É MUITO COMUM se misturar hoje em dia os conceitos de POTÊNCIA e ENERGIA elétricas:

- POTÊNCIA: Está ligada a RELAÇÃO entre CORRENTE( $I$ ) e TENSÃO( $V$ ). É o produto entre as grandezas de tensão que se entrega ao sistema e da corrente que esse mesmo sistema consome. <<<  $[P = V \cdot I]$  ou  $[P = E/t]$

- ENERGIA: Depende da POTÊNCIA porque é sua relação é “a POTÊNCIA no TEMPO”. Fornece-se uma potência ao sistema e ao longo do tempo essa potência pode ser integrada para definir qual foi a energia consumida. <<<  $[E = P \cdot t]$

- Exemplo: Em uma casa é comum ter-se um MEDIDOR DE ENERGIA. Esse medidor MEDE A POTÊNCIA a CADA INSTANTE(QUE É A ENERGIA A CADA INSTANTE) (já que a potência usada numa casa varia de acordo com os momentos do dia com os eletrodomésticos e outros equipamentos ligados), ela vai variando e ao longo do tempo esse MEDIDOR vai INTEGRANDO essa POTÊNCIA NO TEMPO enquanto ACUMULA esse VALOR de maneira que no final do mês a companhia pode ir lá nessa casa olhar esse MEDIDOR e SABER QUAL FOI A ENERGIA que foi utilizada nesse período.

- Potência versus Energia:

- É natural que em sistemas CASEIROS devido aos eletrodomésticos usados ou deixados de usar que a POTÊNCIA VARIE, não sendo constante ao longo do tempo e podendo SIM se modificar (devido a

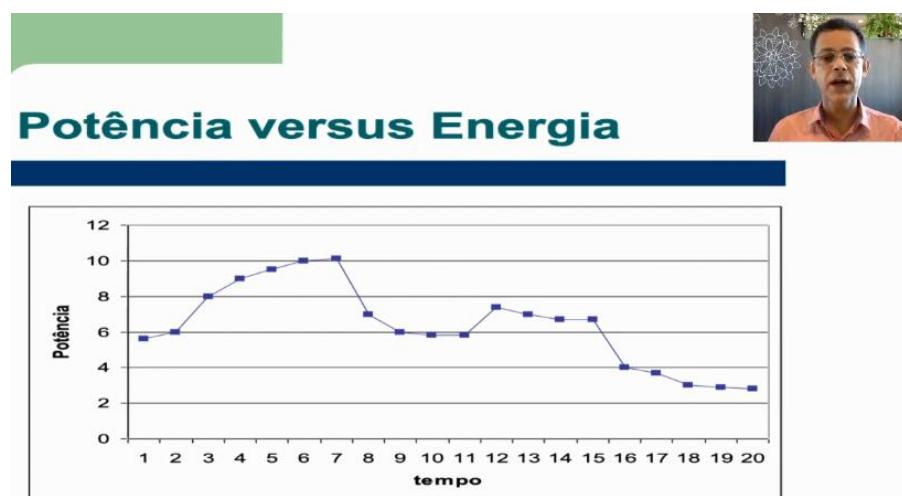
essa inconstância de valores específicos para cada eletrodomésticos e o tempo que ficam gastando eles que também é inconstante!).

- **Exemplo – POTÊNCIA - Na teoria:** Para o caso muito simples de um resistor submetido a uma TENSÃO CONSTANTE a CORRENTE VAI SER CONSTANTE e a **POTÊNCIA** vai ser **CONSTANTE NO TEMPO**.

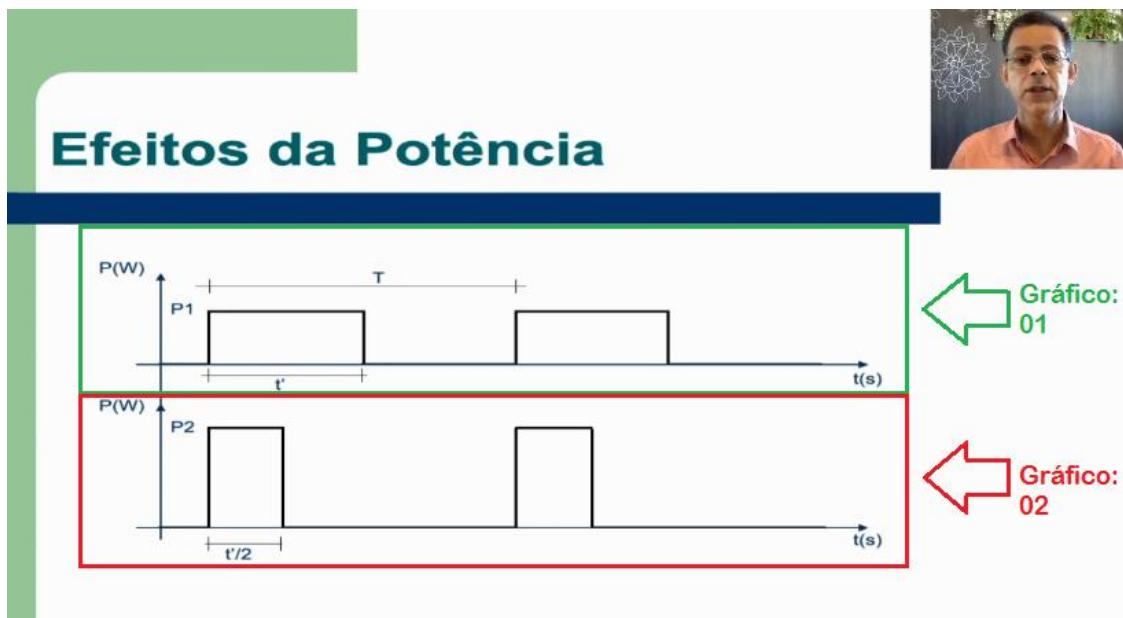
- **Exemplo – POTÊNCIA - Na realidade:** Nos sistemas reais a tendência é que a SOLICITAÇÃO DE **POTÊNCIA** que o SISTEMA FAZ FONTE ela **VARIE COM O TEMPO** (como acontece tanto nos lares de muitas pessoas e TAMBÉM NOS COMPUTADORES de hoje).

- **Nos computadores**, na medida que o tempo passa em função do que estão fazendo vai-se solicitar POTÊNCIAS MAIORES ou MENORES da FONTE.

- Na imagem abaixo que nos mostra um gráfico de “Potência x Tempo”, ao calcular sua área obtém-se a Energia gasta.



- **Efeitos da Potência:**



- Na imagem acima abordamos DUAS situações em linhas do tempo do tipo “Potência(consumida) x Tempo” em sistemas computacionais hipotéticos.

- (**Gráfico:01**) Note que no Gráfico: 01 durante o intervalo  $t'$  o sistema está ligado, o computador está funcionando, e ele está consumindo uma potência admitida nesse momento do

Gráfico:01. O sistema ficará ligado durante esse tempo  $t'$  e depois sua computação a ser feita para e ele “pode ser desligado” e isso vai durar um certo “ciclo” que se repete em tempos “T”. Mais adiante a computação precisa ser executada novamente e o sistema ligado rodando o programa num tempo  $t'$  e assim por diante.

- ([Gráfico:02](#)) No Gráfico: 02 digamos que conseguiu-se AUMENTAR a FREQUÊNCIA do PROCESSADOR do sistema de modo que ele consiga executar o mesmo programa que NO Gráfico:01 na METADE DO TEMPO ( $t'/2$ ), mas no processo a aplicação AINDA É CÍCLICA da MESMA FORMA, então, ele executa a sua tarefa e vai descansar (sendo desligado) e retorna no momento do PRÓXIMO CICLO que pode ser por exemplo de hora em hora.

- ([Comparação dos gráficos](#)) Note que o MESMO PROCESSADOR para executar uma tarefa gastou no PRIMEIRO CASO da imagem acima uma potência  $P_1$  e no segundo uma potência  $P_2$ . Se  $P_2$  for o DOBRO de  $P_1$  e foi executada num tempo que foi METADE do de  $P_1$  então as ÁREAS de AMBOS OS GRÁFICOS 01 e 02 serão IGUAIS! Nesse caso teríamos DUAS SITUAÇÕES em que tal [computador GASTA POTÊNCIAS DIFERENTES](#) mas a [ENERGIA GASTA DA BATERIA](#) do processador [É A MESMA](#) (nos dois casos gasta-se uma MESMA ENERGIA precisando-se de POTÊNCIAS DIFERENTES).

- É importante atentar-se a [essa comparação feita no parágrafo anterior](#) porque [DUAS FONTES de POTÊNCIAS DIFERENTES](#) a [FONTE](#) que [FORNCE UMA ENERGIA MAIOR](#) é [MAIS CARA!](#)

- Toda vez que [fazemos ALTERAÇÕES](#) no [TEMPO DE COMPUTAÇÃO](#) de uma [APLICAÇÃO](#) e a [POTÊNCIA](#) [acompanha essa ALTERAÇÃO PROPORCIONALMENTE](#) teremos um [caso](#) ([estratégia](#)) como o da [imagem acima](#) onde [A POTÊNCIA VARIA mas a ENERGIA NÃO](#). << [Se determinada aplicação NÃO tiver PROBLEMAS com ENERGIA](#) mas sim [PROBLEMAS com POTÊNCIA](#) essa [estratégia](#) ([esse caso](#)) pode ser interessante de [se usar](#).

- ([Conclusões](#)) Na figura nota-se que o DESEMPENHO do computador nos DOIS CASOS em cada gráfico da imagem acima é SUFICIENTE para atender as necessidades da aplicação já que tanto na opção com POTÊNCIA MAIS ALTA([Gráfico:02](#)) e naquela com POTÊNCIA MAIS BAIXA([Gráfico:01](#)) o TEMPO DE COMPUTAÇÃO CABE dentro daquela necessidade que é o período “T”. A computação NÃO PODERIA ultrapassar esse período “T” já que a tarefa tem que ser executada a cada intervalo de tempo “T”. Dentro desse intervalo qualquer tempo de computação seria SATISFATÓRIO para essa aplicação, e nesse caso não precisaria ser “Quanto MENOR o TEMPO MELHOR”. [Se o FOCO da aplicação for ter uma POTÊNCIA BAIXA](#) pode-se pensar que [QUANTO MAIS BAIXA A POTÊNCIA MELHOR](#) ([nunca ZERO é claro](#)).

- [Efeitos da Potência e Energia](#):



**Efeitos de Potência e Energia**

- Aquecimento depende da potência (corrente)
  - Fonte de alimentação, cabos ...
- Duração das baterias depende da energia

- [O AQUECIMENTO](#) produzido pelo computador [DEPENDE DIRETAMENTE](#) da [POTÊNCIA CONSUMIDA. MAIOR A POTÊNCIA, MAIS CALOR EMITIDO](#) pela máquina (mais quente o chip e no pior caso até o circuito). Isso pode gerar a [NECESSIDADE de se usar](#) no hardware [DISSIPADORES, VENTILADORES](#) e muitos outros equipamentos para lidar com o calor gerado o que [ENCARECERÁ O PRODUTO](#) e [AUMENTARÁ O PESO](#) e o [VOLUME](#) do [SISTEMA COMPUTACIONAL](#).

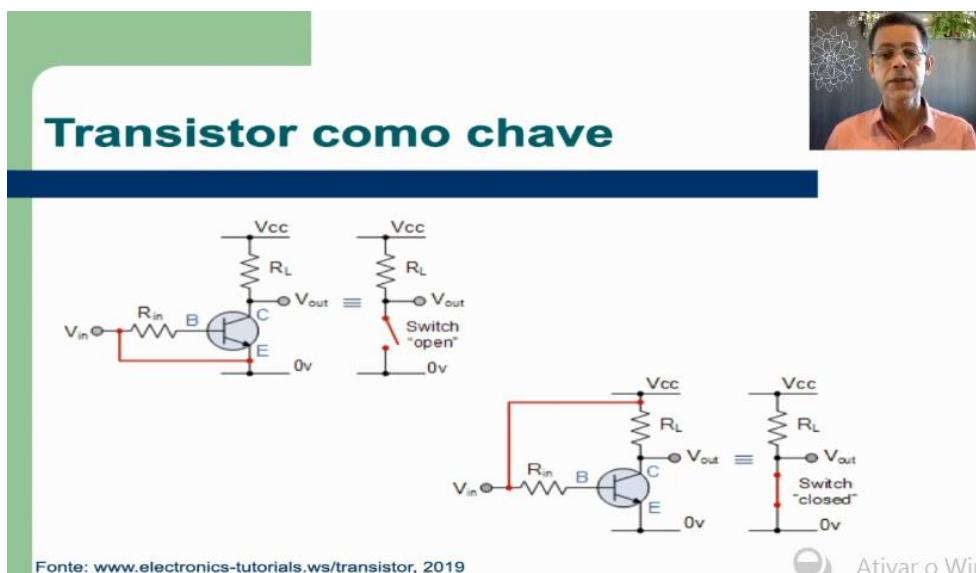
- A FONTE DE ALIMENTAÇÃO é DIRETAMENTE impactada pela POTÊNCIA que o sistema está CONSUMINDO.

- Para sistemas que CONSOMEM MUITA POTÊNCIA necessita-se de FONTES DE ALIMENTAÇÃO MAIS CARAS e CABOS MAIS GROSSOS já que a corrente que é fornecida para o sistema é mais FORTE. FONTES MAIS BARATAS e CABOS MAIS FINOS vão para sistemas consumidores de POUCA POTÊNCIA.

- Em casos de SISTEMAS MANTIDOS POR BATERIAS essas terão uma ENERGIA ARMAZENADA e à medida que o computador vai funcionando ele vai tirando energia dessa bateria e vai utilizando para o seu funcionamento, e nisso a bateria vai se descarregando.

- SISTEMAS que consumem ENERGIAS MENORES para uma MESMA COMPUTAÇÃO PROLONGAM O TEMPO DE USO de suas BATERIAS.

- Transistor como chave:



- Para entender como computadores consomem energia de suas fontes/baterias devemos entender como funcionam os **TRANSISTORES** que compõem os sistemas digitais dessas máquinas, são esses transistores que vão **CONSUMIR ESSA POTÊNCIA**.

- No **COMPUTADOR** transistores funcionam como **CHAVES** (são ligados e desligados para produzir os efeitos de 1's e 0's que tão na maneira como modelamos nosso software e como compreendemos o computador). Dentro da máquina há milhões deles funcionando ou no nível lógico 0 ou no nível lógico 1.

- Transistores se comportam-se assim como inversores (O nível lógico da entrada é o inverso na saída).

- Nos NÍVEIS LÓGICOS 0(CHAVE ABERTA) e 1(CHAVE FECHADA) os TRANSISTORES não precisam praticamente de MUITA POTÊNCIA ELÉTRICA:

- (Transistor como CHAVE FECHADA): Tem CORRENTE PASSANDO pelo lado COLETOR/EMISSOR do transistor mas NÃO HÁ TENSÃO e como a potência é o produto "TENSÃO pela CORRENTE" a POTÊNCIA no TRANSISTOR NESSE MOMENTO vai ser MUITO BAIXA.

- (Transistor como CHAVE ABERTA): Como a chave está ABERTA tem TENSÃO na SAÍDA DO TRANSISTOR mas NÃO HÁ CORRENTE PASSANDO por ele nesse momento o que torna a POTÊNCIA PRATICAMENTE 0.

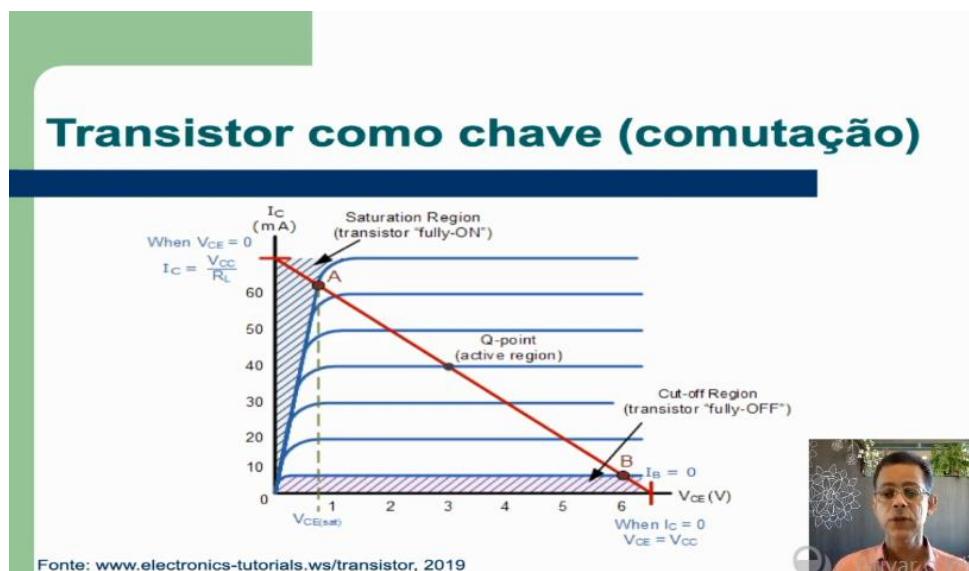
- [LINK para mais detalhes desses DOIS ESTADOS:](#)

<https://blog.render.com.br/elettronica/como-utilizar-o-transistor-como-chaveador/#:~:text=Quando%20um%20transistor%20est%C3%A1%20saturado,com%20tens%C3%A3o%20cont%C3%ADnuo%20ou%20sinais.>

- Transistores podem ser USADOS de DIVERSAS FORMAS (“Eletrônica analógica feelings” kkkkkk ☺) mas em computadores essas duas formas mostradas na imagem acima (transistor como CHAVE ABERTA ou como CHAVE FECHADA) é que vão nos interessar.

- [Comutação:](#)

- O transistor comumente quando está num ESTADO 0 por exemplo e recebe uma entrada diferente ele vai para o ESTADO 1. Esse PROCESSO DE MUDANÇA DE ESTADOS também VAI GASTAR POTÊNCIA.



- No gráfico da imagem acima o TRANSISTOR vai ter que CAMINHAR nessa reta vermelha CADA VEZ que TIVER DE TROCAR de ESTADO (ESSA TROCA chama-se COMUTAÇÃO).
- Se o TRANSISTOR estiver entregando NÍVEL LÓGICO 0 na SAÍDA ele vai estar nesse ponto "A" do gráfico acima (onde sua tensão de saída é 0).
- Se o TRANSISTOR estiver entregando NÍVEL LÓGICO 1 na SAÍDA ele vai estar nesse ponto "B" do gráfico acima (onde sua corrente de saída é 0).
- Quando o transistor COMUTA (muda de estado) ele passa por essa RETA VERMELHA de UM LADO para o OUTRO e nessa PASSAGEM ele vai ter pontos onde o “PRODUTO: TENSÃO x CORRENTE” vai ser BEM MAIS ALTO do que era nos EXTREMOS do gráfico. Esse momento da COMUTAÇÃO vai pedir uma POTÊNCIA MAIS ALTA do TRANSISTOR do que nos momentos em que estiver PARADO ou no ESTADO 0 ou no ESTADO 1.

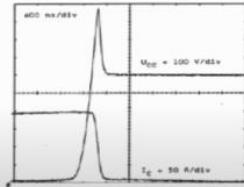
- [Potência dinâmica:](#)



## Potência dinâmica

- Perdas de comutação em transistores
- Função da freqüência e da tensão

$$P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$$



- Essa POTÊNCIA nesse MOMENTO DA COMUTAÇÃO é chamada de POTÊNCIA DINÂMICA.
- A POTÊNCIA DINÂMICA é consumida no momento em que o TRANSISTOR TROCA DE ESTADO.
- O gráfico a direita da imagem acima mostra que DURANTE o processo de QUEDA DA CORRENTE e de ELEVAÇÃO DA TENSÃO tem-se momentos onde esses dois valores (CORRENTE e TENSÃO) NÃO SÃO 0 e o PRODUTO ENTRE ELES VAI DAR UM VALOR SIGNIFICATIVO.
- O gráfico a direita da imagem acima também mostra as PERDAS de COMUTAÇÃO em TRANSISTORES:
  - (Desenho do gráfico da corrente): No gráfico é a linha onde se mostra essa transição de valor alto para valor baixo.
  - (Desenho do gráfico da tensão): No gráfico é a linha onde se mostra essa transição de valor 0 passando por um PICO (SOBRESSALTO) MOMENTÂNEO e depois voltando para um valor de tensão constante.
- (Recapitulando) Nos MOMENTOS ANTERIORES e POSTERIORES a TRANSIÇÃO(COMUTAÇÃO) no começo a CORRENTE tem VALOR ALTO e a TENSÃO é PRATICAMENTE 0 então POTÊNCIA APROXIMADAMENTE 0, e do MESMO JEITO no final dessa TRANSIÇÃO termos uma CORRENTE que é PRATICAMENTE 0 e uma TENSÃO de VALOR ALTO o que resulta numa POTÊNCIA de valor, também, APROXIMADAMENTE 0.
- A área que trabalha com microeletrônica e estuda como os transistores são fabricados e que constrói os chips dos processadores têm dessa fórmula para o semicondutor:

- Função da freqüência e da tensão

$$P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$$

- Chamemos essa fórmula da forma simples de: "Função da Freqüência e da Tensão".

- Essa FÓRMULA É PROPORCIONAL a:

- Freqüência: Maior a FREQUÊNCIA, Maior a POTÊNCIA CONSUMIDA pelo sistema. É uma RELAÇÃO LINEAR (Se uma das grandezas aumentar a outra aumenta proporcionalmente).

- Tensão que alimenta o processador(Vdd): Depende QUADRATICAMENTE dessa tensão Vcc, ou seja, se a TENÇÃO for DOBRADA se QUADRIPLICA a POTÊNCIA:

- (Problema) Por isso deve-se ter MUITO CUIDADO na hora de AUMENTAR A TENSÃO de um PROCESSADOR pois isso impacta de maneira MUITO SIGNIFICATIVA na POTÊNCIA do sistema.

- (Vantagem) Também, é muito interessante DIMINUIR a TENÇÃO DE ALIMENTAÇÃO DO PROCESSADOR (Vcc) porque isso dá um GANHO QUADRÁTICO na POTÊNCIA.

- Capacitância: É um valor da própria CONSTRUÇÃO do transistor sendo muito específico de como este é feito pelo fabricante.

- Taxa de proporcionalidade de uso do transistor( $\alpha$ ): É um valor da própria CONSTRUÇÃO do transistor sendo muito específico de como este é feito pelo fabricante.

- \*\*\*Concluindo, PARA NÓS essa fórmula vai ser simplificada pensando que a POTÊNCIA é PROPORCIONAL a: Tensão ao QUADRADO, a FREQUÊNCIA e a uma CONSTANTE que DEPENDE de COMO o SISTEMA FOI FABRICADO e que TIPO de TECNOLOGIA de FABRICAÇÃO foi usada nesse CHIP (isso, pra nós, só será “uma constante”).

- Potência dissipada por um sistema de computação (e CMOS):



## CMOS e a potência dissipada

- Tensão de alimentação
  - A potência é proporcional a  $V^2$ .
- Comutações
  - Maior parte da potência é dissipada no chaveamento dos transistores de saída.
- Leakage (vazamento)
  - Correntes circulam pelos gates mesmo quando estes estão inativos.

- Quando falamos de Potência dissipada por um sistema de computação essa potência DEPENDE da TECNOLOGIA de FABRICAÇÃO desses CHIPS. Na maioria dos nossos processadores e sistemas do computador essa TECNOLOGIA chama-se CMOS.

- Como visto no tópico anterior essa POTÊNCIA DEPENDE:

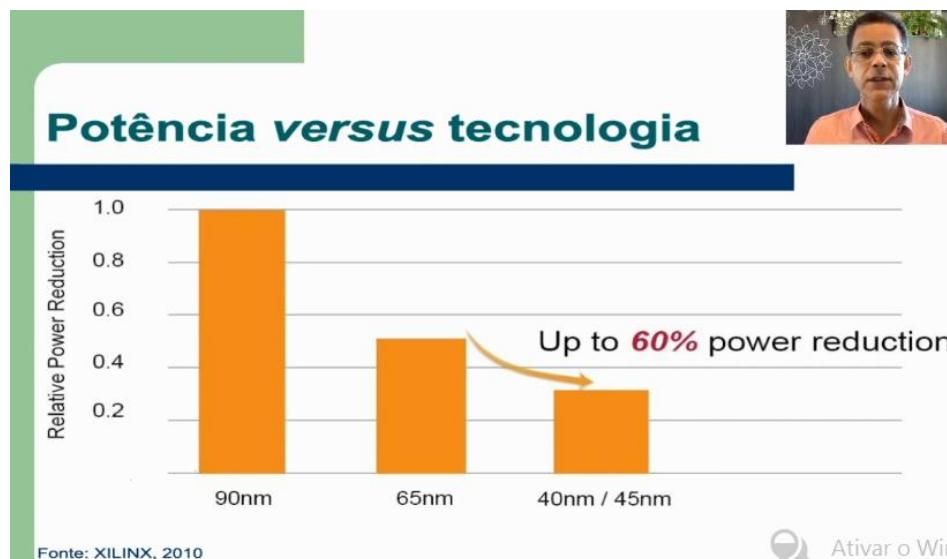
- Da TENSÃO e é PROPORCIONAL ao seu QUADRADO (Ver imagem acima).

- Das COMUTAÇÕES do TRANSISTOR (Ver imagem acima).

- LEAKAGE (“Vazamento”): É outro fator que contribui para a POTÊNCIA TOTAL do computador. Existem CORRENTES chamadas CORRENTES DE FUGA (tópico bastante visto por quem estuda SEMICONDUTORES), mesmo nas junções POLARIZADAS REVERSAMENTE existem CORRENTES e essas CORRENTES vão significar DISSIPAÇÃO DE POTÊNCIA.

- Também existem CIRCUITOS DE POLARIZAÇÃO DOS TRANSISTORES de maneira que mesmos eles estando INATIVOS/PARADOS em um estado lógico (seja 0 ou 1) eles VÃO SIM precisar de ALGUMA POTÊNCIA para isso.

- Potência versus Tecnologia:

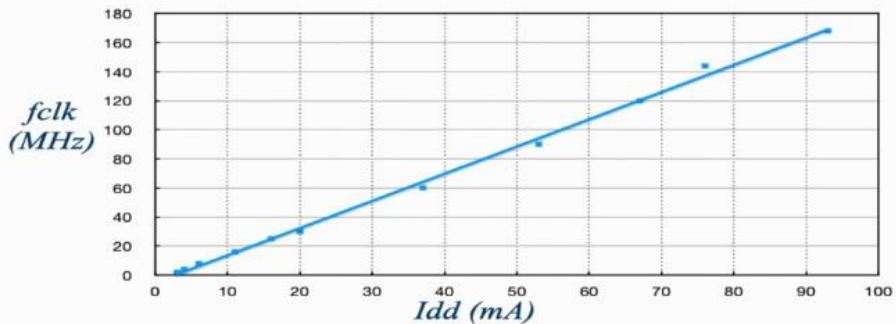


Ativar o Wiz

- A figura acima é uma VARIAÇÃO de como a POTÊNCIA de um SISTEMA COMPUTACIONAL é IMPACTADA pela TÉCNOLOGIA DE FABRICAÇÃO.
- No gráfico da figura temos no EIXO X tecnologias de transistores cada vez mais menores (por isso o “nm” que vem de “nanômetros”) e mais rápidos e no EIXO Y a POTÊNCIA DISSIPADA.
- (Conclusões a partir do gráfico acima): Note que a medida que os anos vão passando os fabricantes vão PROPOUNDO NOVAS TECNOLOGIAS de TRANSISTORES MENORES e MAIS RÁPIDOS, e esses transistores também DISSIPAM MENOS POTÊNCIA. ENTÃO, é uma CARACTERÍSTICA das TECNOLOGIAS MAIS NOVAS dissiparem uma POTÊNCIA MENOR, isso faz com que os FABRICANTES muitas vezes VÃO para essas TECNOLOGIAS (mesmo elas sendo CARAS pra dêdêu kkk) NÃO PORQUE QUEREM COMPUTADORES MAIS RÁPIDOS mas somente pelo fato de quererem computadores com MENOR POTÊNCIA.

- Valores de operação para corrente e frequência:

## Valores de operação para corrente frequência - STM32F407VG



Fonte: www.st.com, 2016

Ativar o WPS

- A imagem do gráfico acima trata-se de um dado de um microcontrolador da família ARM que mostra **números** experimentais tirados do “*datasheet*” do processador.

- **EIXO X:** Corrente consumida (Idd).

- **EIXO Y:** Frequência de clock (fclk).

- Os **PONTOS AZUIS CLARO** no gráfico tem suas coordenadas mostradas em baixo da imagem. Para **CADA FREQUÊNCIA** existe **uma CORRENTE medida** lá **de funcionamento do processador**.

- Perceba que no gráfico há uma relação LINEARMENTE PROPORCIONAL entre a **Corrente e a Frequência (DOBROU A FREQUÊNCIA, DOBROU A CORRENTE)**. Se **DOBROU A CORRENTE e NÃO HOUVE ALTERAÇÃO na TENSÃO DE ALIMENTAÇÃO do processador a POTÊNCIA vai, também, DOBRAR**.

- **Estratégias para DIMINUIR a potência dissipada:**

- Com essas estratégias procuramos **DIMINUIR O AQUECIMENTO** do PROCESSADOR e até mesmo **DIMINUIR a ENERGIA** que ele **PRECISA PARA FUNCIONAR**.

1 - **Uma das estratégias seria REDUZIR a TENSÃO DE ALIMENTAÇÃO do processador.** Como o ganho disso é QUADRÁTICO via de regra o que se procura fazer é primeiro baixar essa TENSÃO DE ALIMENTAÇÃO porque é a ESTRATÉGIA dentre todas as outras aqui que se tem o MAIOR GANHO, se isso for o suficiente para se resolver o problema ótimo porque temos REDUÇÃO NA POTÊNCIA SEM REDUÇÃO DE DESEMPENHO NECESSARIAMENTE. É possível que tenhamos de baixar a frequência também, mas num primeiro momento se tenta BAIXAR A TENSÃO sem ter que BAIXAR A FREQUÊNCIA e ao fazer isso estamos conseguindo BAIXAR A POTÊNCIA e vamos CONSEGUIR BAIXAR também a ENERGIA porque o TEMPO DE COMPUTAÇÃO NÃO VAI AUMENTAR.

2 - **Outra estratégia que também vem daquela fórmula é BAIXAR A FREQUÊNCIA DE OPERAÇÃO DO PROCESSADOR** sendo que nesse caso **não vamos conseguir REDUÇÃO DE ENERGIA** mas **vamos conseguir BAIXAR a POTÊNCIA CONSUMIDA**.

3 - **Outra estratégia tem a ver com o sistema que está dentro do processador** pelo fato de **NEM TODO ELE estar SENDO NECESSÁRIO o TEMPO TODO**. Então uma das estratégias é **NÃO**

DEIXAR o CLOCK CHEGAR a esses SISTEMAS (isso se chama “CLOCK GATING”). Se a FREQUÊNCIA DE CLOCK não chega a esses SISTEMAS os TRANSISTORES DELES NÃO VÃO TROCAR DE ESTADO e não irar-se TER a DISSIPAÇÃO POR COMUTAÇÃO (mas os transistores ainda estarão sendo ALIMENTADOS, aquele GASTO para MANTER eles FUNCIONANDO ainda EXISTIRÁ).

4 – Outra estratégia que DIMINÚI AINDA MAIS A POTÊNCIA que é DESLIGAR PARTES DA CPU ou do SoC (“*System On Chip*”) mesmo.

- (Conclusões sobre as estratégias) ESSES DOIS CAMINHOS de NÃO DEIXAR O CLOCK CHEGAR(3) e DESLIGAR PARTES da CPU(4) são caminhos/estratégias que EXPLORAM o fato de que NEM TUDO no SoC (“*System On Chip*”) ESTÁ SENDO UTILIZADO o TEMPO TODO.

- A imagem abaixo apresenta resumidamente essas “Estratégias para DIMINUIR a potência dissipada”:

**Reducindo a potência dissipada**

- Reduzir a tensão de alimentação da CPU (ganho quadrático).
- Reduzir a frequência de operação da CPU.
- Impedir momentaneamente o clock de chegar a circuitos não ativos.
- Desligar partes da CPU.

Não reduz a energia

Clock gating

- Estratégias para GERENCIAMENTO de potência dissipada:

**Estratégias para gerenciamento de potência**

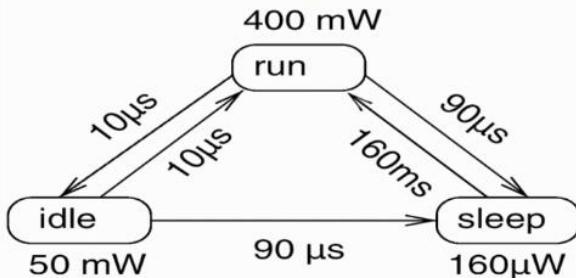
- DPM (Dynamic Power Management)
- DVS (Dynamic Voltage Scaling)

- A imagem acima mostra DUAS ESTRATÉGIAS GLOBAIS implementadas pelos SoC's (“*System On Chip*”) para DIMINUIR a POTÊNCIA GLOBAL GASTA PELO SISTEMA, ela são:

- DPM (“Dynamic Power Management”):

## DPM (Dynamic Power Management)

- A CPU tem alguns níveis de potência



Exemplo do processador StrongArm



- [Essa estratégia é implementada](#) da seguinte maneira:

- Os processadores (SoC's dos nossos SEMB's) vão ter [VÁRIOS NÍVEIS de POTÊNCIAS DE OPERAÇÃO](#). Na [imagem acima](#) tem-se um simples exemplo de processador que tem [3 NÍVEIS de OPERAÇÃO](#):

- [run](#): É o nível no alto da relação triangular onde o processador funcionar plenamente. A POTÊNCIA consumida pelo processador nesse NÍVEL é de 400 mW.

- [idle](#): É outro NÍVEL/ESTADO que o PROCESSADOR pode ASSUMIR. A POTÊNCIA consumida pelo processador nesse NÍVEL é de 50 mW.

- [sleep](#): É outro NÍVEL/ESTADO que o PROCESSADOR pode ASSUMIR. A POTÊNCIA consumida pelo processador nesse NÍVEL é de 160 µW.

- ([análise-01](#)) Existem COMANDOS que podem ser passados para esse processador. Essas palavras de comando que são mandadas para ele FAZ COM QUE ESSE PROCESSADOR MUDE UM ESTADO PARA OUTRO e ao mudar de estado teremos algumas coisas desligadas (sem receber clocks) de maneira que agora naquele novo estado tem-se um NÍVEL DE POTÊNCIA MAIS BAIXO.

- ([análise-02](#)) Veja que a transição do estado "[run](#)" para o estado "[idle](#)" nesse processador DURA 10 µs, tanto para entrar no modo "[idle](#)" quanto para retornar dele.

- No modo "[sleep](#)" a entrada custa 90 µs mas a volta custa 160 ms (que é MUITO MAIS TEMPO) o que é TÍPICO de PROCESSOS QUE DESLIGAM PARTES DO PROCESSADOR.

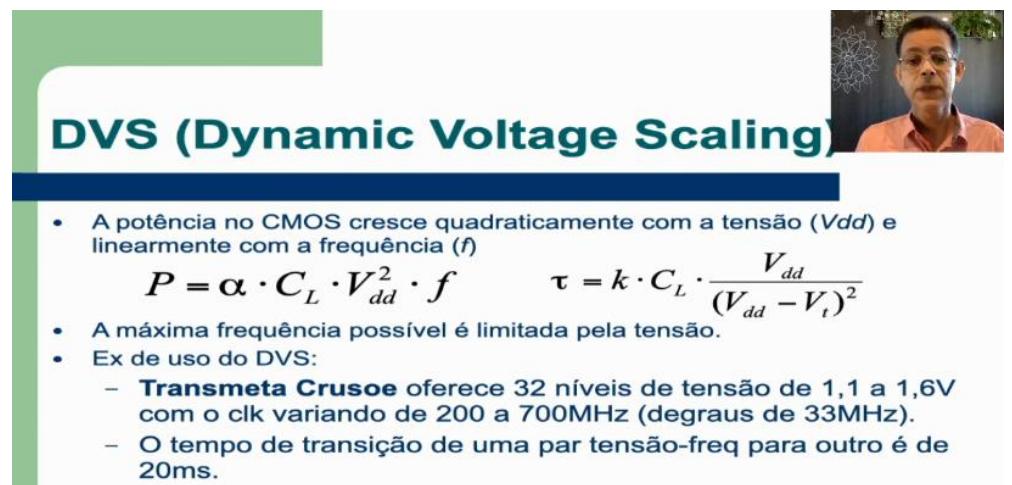
- Quando se [REMOVE a FREQUÊNCIA DE CLOCK](#) de um [SISTEMA DIGITAL](#) ele [fica ÁPTO](#) a [RETORNAR AO FUNCIONAMENTO](#) de [uma MANEIRA MUITO RÁPIDA](#).

- Quando se REMOVE a ALIMENTAÇÃO de um BLOCO DIGITAL DENTRO de um SISTEMA ele DEMORA MAIS PRA RETORNAR A ATIVA.

- (Níveis diferentes de potência – Em Datasheets) Se olharmos os datasheets dos processadores comerciais hoje em dia veremos que eles possuem VÁRIOS NÍVEIS quanto aos seus tópicos de GERENCIAMENTO DE ENERGIA. Sua grande maioria tem 4, 5 ou mais níveis de energia cada um tendo indicado pelo datasheet que blocos estão desligados em cada um desses níveis (pode-se ter PARTES DAS MEMÓRIAS DESLIGADAS, bem como alguns DISPOSITIVOS E/S também, podemos ter o dispositivo operando a uma frequência mais baixa).

- \*\*\*(Conclusão) De acordo com o MOMENTO de EXECUÇÃO da nossa APLICAÇÃO podemos ESCOLHER em QUAIS DESSES NÍVEIS sob os quais nossa aplicação pode estar dividida queremos operar. Isso nos dá então, vários níveis diferentes de potência para o nosso dispositivo trabalhar.

- DVS (“Dynamic Voltage Scaling”):



- A potência no CMOS cresce quadraticamente com a tensão ( $V_{dd}$ ) e linearmente com a frequência ( $f$ )  
$$P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \quad \tau = k \cdot C_L \cdot \frac{V_{dd}}{(V_{dd} - V_t)^2}$$
- A máxima frequência possível é limitada pela tensão.
- Ex de uso do DVS:
  - **Transmeta Crusoe** oferece 32 níveis de tensão de 1,1 a 1,6V com o clk variando de 200 a 700MHz (degraus de 33MHz).
  - O tempo de transição de uma par tensão-freq para outro é de 20ms.

- Falar sobre DVS, Escalonamento de Tensão Dinâmico, significa também falar sobre Escalonamento de Frequência Dinâmico.

- A técnica DVS consiste em TROCAR A FREQUÊNCIA DE OPERAÇÃO do PROCESSADOR com ELE FUNCIONANDO.

- (Caso-01) Uma coisa é você no projeto do seu microcontrolador ESCOLHER qual a FREQUÊNCIA DE OPERAÇÃO, colocá-lo para funcionar dentro da sua “caixa/cápsula”, entregá-lo para o cliente e ele irá ficar funcionando lá naquela FREQUÊNCIA FIXA.

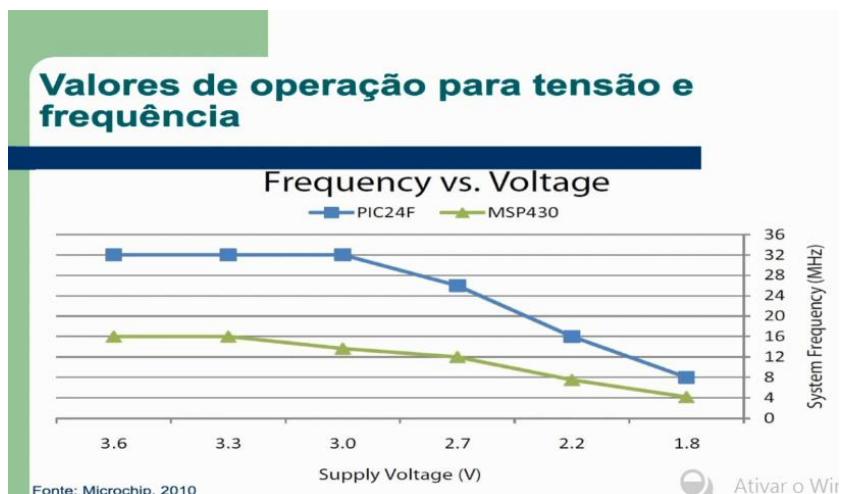
- (Caso-02) Outra coisa é NA DINÂMICA de FUNCIONAMENTO o SISTEMA identificar algum MOMENTO DE OCIOSIDADE e REDUZIR a FREQUÊNCIA de OPERAÇÃO do SISTEMA para poder BAIXAR a sua POTÊNCIA DE OPERAÇÃO.

- É do “Caso-02” que estamos falando quando falamos de DVS!

- E nesse caso do DVS sempre procura-se **OPERAR A MENOR TENSÃO POSSÍVEL** que uma CERTA FREQUÊNCIA **PERMITE** pois estamos **tentando LEVAR O PROCESSADOR** para o estado de **MÍNIMA POTÊNCIA POSSÍVEL** para manter a **APLICAÇÃO FUNCIONANDO**.

- (**Análise - Tempo de troca de níveis de operação de tensão/frequência**) Note que no exemplo da imagem acima (“*Transmeta Crusoe*”) o **TEMPO NECESSÁRIO** para se FAZER A COMUTAÇÃO(TROCA) entre um NÍVEL DE OPERAÇÃO DE TENSÃO E FREQUÊNCIA para OUTRO é de 20 ms o que **não é um tempo tão curto assim**, é interessante que o software utilize isso **mas não COM MUITA FREQUÊNCIA**, afinal, essa **espera de 20 ms** pra se fazer essa **TROCA** pode **IMPACTAR NEGATIVAMENTE** o **DESEMPENHO DA APLICAÇÃO** tanto em **termos de COMPUTAÇÃO** quanto em **termos de POTÊNCIA** o que seria péssimo.

- Acontece que a **MÁXIMA FREQUÊNCIA DE OPERAÇÃO POSSÍVEL** pra um SISTEMA COMPUTACIONAL é LIMITADA PELA TENSÃO, veja o exemplo gráfico abaixo:



- A imagem acima mostra um exemplo de DOIS PROCESSADORES COMERCIAIS nessa situação onde tanto a FREQUÊNCIA quanto a TENSÃO tem que caminhar juntas.

- **EIXO X:** Tensão de alimentação (V).

- **EIXO Y:** Frequências de operação (MHz).

- (**Análise do gráfico – segmento verde**) A linha em tom verde representa um determinado processador. Agora veja, esse processador verde é capaz de operar com tensões entre 1,8 V e 3,6 V e com frequências entre 4 e 12 MHz, só que não se pode ESCOLHER LIVREMENTE QUALQUER COMBINAÇÃO nessas duas FAIXAS DE VALORES (TENSÃO e FREQUÊNCIA). Se escolhermos operar com uma alimentação de 1,8 V por exemplo, estaremos limitados a uma FREQUÊNCIA DE 4 MHz. Se quiser operar a 8 ou 12 MHz terá que SUBIR A ALIMENTAÇÃO(TENSÃO) para os valores que o gráfico mostra pro processador que estamos aqui abordando (dá linha verde no caso). Note que se quiséssemos operar em frequências maiores que 16 MHz não seria possível pois o MÁXIMO possível PARA ESSE PROCESSADOR É SÓ ATÉ 16 MHz.

- \*\*\***Na PRÁTICA, FREQUÊNCIAS DE OPERAÇÃO mais altas IMPLICAM em operar NAS TENSÕES MAIORES do PROCESSADOR** e se PUDERMOS **OPERAR** a FREQUÊNCIAS MAIS BAIXAS poderemos nos **BENEFICIAR** de TENSÕES MAIS BAIXAS.

- (**Conclusões**) Note que quando quisermos REDUZIR a FREQUÊNCIA, NÃO SOMOS OBRIGADOS a REDUZIR a TENSÃO. No gráfico acima por exemplo, se quisermos operar a 3,6 V (tensão FIXA nesse processador) podemos ESCOLHER LIVREMENTE entre 12 e 4 MHz a FREQUÊNCIA que for CONVENIENTE para nossa aplicação, estaríamos GANHANDO algo na POTÊNCIA se baixássemos de 12 para 4 MHz por exemplo mas NÃO ESTARÍAMOS usufruindo desse benefício em relação a tensão.



- Lógica Reconfigurável (em SEMB's):

**Hardware programável?**

A slide featuring a video of a man speaking in the top right corner. Below him is a blue and green graphic with the number '2'. To the left is a photograph of a Zynq SoC development board with a USB connection. To the right is an illustration of a person sitting at a desk with multiple computer monitors displaying code snippets like '</>', 'Js', 'C++', 'CSS', 'CMS', 'PHP', and 'HTML'.

- É possível um hardware programável? Posso fazer um programa pra esse hardware mesmo estando o chip já fabricado? **Sim!!!** ☺

- Estamos falando de **LINGUAGENS DE DESCRIÇÃO DE HARDWARE**, existem linguagens apropriadas pra esse propósito, que **são linguagens para programar/configurar esse chip** (após compra-lo é claro).

- Veja a imagem abaixo para entender a **ideia básica** de um **HARDWARE PROGRAMÁVEL**(ou PLD's):

**Hardware programável!**

A slide featuring a video of a man speaking in the top right corner. Below him is a green graphic with the number '3'. On the left, there is a logic diagram showing two parallel paths. The top path has inputs A1 and A0, and output S. It contains a memory block with a 2x2 grid of values (00, 01, 10, 11) and a truth table below it:

A1	A0	S
0	0	0
0	1	1
1	0	1
1	1	1

The bottom path has inputs A1 and A0, and output S. It contains a logic gate symbol (an AND gate).

**Hardware programável!**

A slide featuring a video of a man speaking in the top right corner. Below him is a green graphic with the number '4'. On the left, there is a logic diagram similar to the one above, but with a more complex internal structure. It includes a memory block, a RAM block, a LUT (look-up table), and a D flip-flop. The truth table remains the same:

A1	A0	S
0	0	0
0	1	0
1	0	0
1	1	1

- Suponha DUAS caixas com um CIRCUITO DIFERENTE em cada, não sabemos que circuitos são até testá-los. Digamos que na caixa mais acima da imagem haja uma memória como 4 endereços

(00,01,10 e 11). A tabela lógica a direita da imagem acima resume os resultados devolvidos pela memória baseados na configuração das duas entradas lógicas (que só podem assumir 1's e 0's) do circuito(então se A1 e A0 na entrada forem ambos 0 teremos o endereço 00 da memória e segundo sua tabela esse endereço devolverá na saída S o valor 0).

- Note que essa tabela lógica a direita da imagem acima não se encaixa nos resultados que SOMENTE A MEMÓRIA da CAIXA mais acima devolve, na CAIXA mais abaixo temos a porta lógica "OR" e sua tabela lógica é a mesma que a da memória! Portanto temos então nas DUAS CAIXAS [ **circuitos DIFERENTES** que REAGEM de forma **DIFERENTE** mas entregando as **MESMAS SAÍDAS** quando recebem **ENTRADAS IGUAIS**. ]-----> Essa é a ideia que é explorada num HARDWARE PROGRAMÁVEL!

- Então, para o exemplo a direita na imagem acima podemos concluir que ao invés de termos uma porta lógica "Or" que SEMPRE vai manifestar o **MESMO COMPORTAMENTO**, vamos ter uma **ESTRUTURA DE MEMÓRIA** que tem que ter sido **PREVIAMENTE PROGRAMADA** com a **LÓGICA** que seu programador quis como **RESULTADO** do **COMPORTAMENTO** de DETERMINADO CIRCUITO.

- Para a memória no exemplo a direita da figura acima programou-se a LÓGICA da porta "Or" e essa MEÓRIA "SIMULA" o COMPORTAMENTO dessa PORTA "Or" com a diferença de que pode-se MUDAR O CONTEÚDO dessa MEMÓRIA de modo que ela passe a ter um outro COMPORTAMENTO (Na verdade **QUALQUER CIRCUITO combinacional** de **DUAS ENTRADAS e UMA SAÍDA** pode ser **EMULADO por essa MEMÓRIA!**). Na imagem acima no exemplo da esquerda temos agora uma memória que simula o mesmo comportamento de uma porta lógica "AND".

- Vemos que só por esses dois exemplos iniciais hardwares programáveis tem algumas **DIFERENÇAS IMPORTANTES**:

1 - **A complexidade do circuito.** Uma memória de 4 bits como a apresentada nos exemplos da imagem acima é um circuito eletrônico BEM MAIS complexo do que as portas "Or" e "And" e isso torna a EMULAÇÃO que procura-se configurar nesse circuito difícil de se fazer. Por isso as vezes escolhe-se trocar esse HARDWARE MAIS COMPLEXO e FLEXÍVEL, como a MEMÓRIA RECONFIGURÁVEL, por um HADWARE MAIS SIMPLES E RÍGIDO, como as PORTAS "Or" e "And".

2 - Se precisarmos de **FLEXIBILIDADE** optamos pelo modelo da MEMÓRIA RECONFIGURÁVEL.

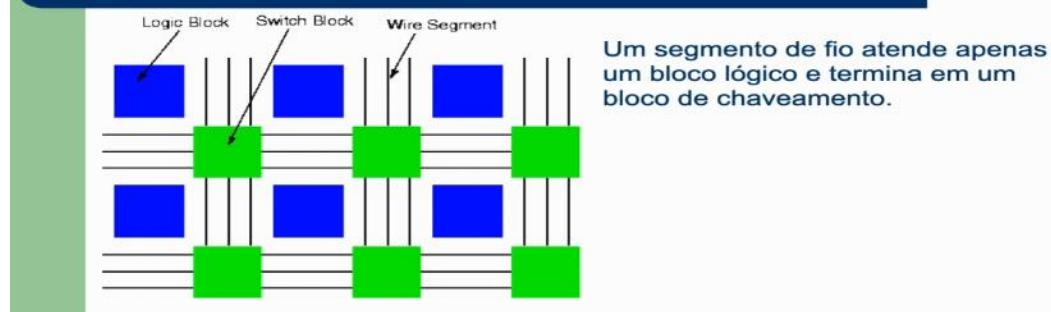
- Se NÃO precisarmos de FLEXIBILIDADE optamos pelo modelo RÍGIDO das PORTAS LÓGICAS "Or" e "And".

- (**Análise do exemplo acima para FPGA's**) Nos casos dos **FPGA's** que são **CIRCUITOS PROGRAMÁVEIS BASTANTE POPULARES** essa "**MEMÓRIA RECONFIGURÁVEL**" que apresentamos é **IMPLEMENTADA** utilizando **TECNOLOGIA RAM**, essa memória também é conhecida como "**LUT: look-up table**" e recebe esse nome pois na verdade **REPRESENTA** uma "**TABELA DE DADOS**".

- Arquitetura - **FPGA**:



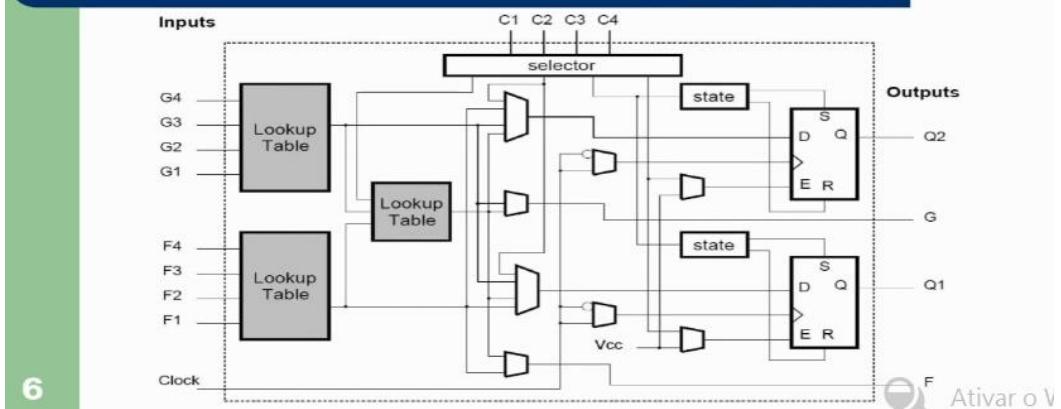
## Arquitetura FPGA – conexões



- Essa arquitetura pode ser simplificada na figura acima, onde esse “blocos azuis” são BLOCOS LÓGICOS (circuitos que implementam aquelas PORTAS LÓGICAS):



## Arquitetura FPGA - Blocos XILINX – CLB (Configurable Logic Block)



- A imagem acima mostra a ESTRUTURA INTERNA de um desses BLOCOS LÓGICOS. Veja que é um sistema BEM MAIS COMPEXO do que foi usado para se explicar os conceitos iniciais do assunto. Temos vários blocos lógicos várias “Lookup Tables” que nós falamos anteriormente, tudo isso para representar 1 dos BLOCOS AZUIS da figura anterior a essa.

- Aliado a esses “blocos azuis” (BLOCOS LÓGICOS) tem-se blocos que representam as CONEXÕES entre eles, assim, num FPGA temos:

- A LÓGICA que é REPROGRAMÁVEL (“blocos azuis” na figura, um conjunto muito grande de blocos como esse está disponível dentro do CHIP).
- Conjuntos de blocos que fazem a CONEXÃO ENTRE ESSES BLOCOS LÓGICOS (“blocos verdes”).
- Então, num FPGA TUDO É FLEXÍVEL e PROGRAMÁVEL: A LÓGICA de cada um desses BLOCOS e a MANEIRA COMO ELES VÃO SE CONECTAR.
- A TECNOLOGIA DE MEMÓRIA utilizadas pelos FPGA's é a tecnologia RAM. A tecnologia RAM permite que se GRAVE DADOS naquela memória que vão DEFINIR o COMPORTAMENTO LÓGICO do

**CHIP RECONFIGURÁVEL.** Temos algumas **VANTAGENS** e outras **DESVANTAGENS** ao se UTILIZAR essa TECNOLOGIA RAM, vejamos elas na figura abaixo:



## Tecnologia SRAM (FPGA)

- Células de RAM estática são usadas para:
  - Implementação de lógica (LUT)
  - Como block RAM (BRAM)
  - Controle de chaves de roteamento e configuração
- Vantagens:
  - Facilmente modificável
  - Boa densidade
  - Flexível (bom para FSM e aritméticos)
- Desvantagens:
  - Volatilidade
  - Alta potência (geralmente)

- Elas são UTILIZADAS para IMPLEMENTAR aquelas TABELAS de lógica LUT.
- Além dos sistemas lógicos(LUT's) temos sistemas que vão funcionar como MEMÓRIA RAM dentro dos FPGA's chamados de BRAM (também são implementados por MEMÓRIA RAM).
- As LÓGICAS DE CONTROLE (os “blocos verdes” mostrados anteriormente que podem ser reprogramáveis) que servem para CONECTAR os BLOCOS LÓGICOS DENTRO DO FPGA também vão ter associada a si MEMÓRIA RAM para definir o TIPO DE CONEXÃO que essas lógicas vão IMPLEMENTAR.

- Vantagens de se utilizar RAM em FPGA's:

- A FACILIDADE com que se MODIFICA a MEMÓRIA RAM(flexibilidade).
- O fato ser uma MEMÓRIA que OCUPA POUCO ESPAÇO([ajuda a manter os chips de FPGA's dentro de um TAMANHO LIMITADO](#)).
- Pela sua FLEXIBILIDADE elas podem ser UTILIZADAS para IMPLEMENTAR DIFERENTES TIPOS DE CIRCUITOS LÓGICOS (isso é ÓTIMO para emulação de circuitos FSM's e ARITMÉTICOS).

- Desvantagens de se utilizar RAM em FPGA's:

- São VOLÁTEIS. Enquanto a MEMÓRIA estiver ALIMENTADA ela mantém a configuração que FOI GRAVADA. Ao DESLIGAR a ALIMENTAÇÃO do FPGA a configuração que foi CARREGADA na MEMÓRIA pelo FPGA é APAGADA. Naturalmente, existem [medidas para se evitar isso](#) que seria o [FPGA “CARREGAR a sua CONFIGURAÇÃO numa MEMÓRIA NÃO VOLÁTIL EXTERNA a ele”](#). Geralmente quando se LIGA um chip FPGA ao seu lado existirá uma memória NÃO-VOLÁTIL, tipicamente uma FLASH, de onde ele CARREGA a sua CONFIGURAÇÃO.
- Memórias RAM/SRAM consomem uma POTÊNCIA MUITO ELEVADA (principalmente nos FPGA's MAIS MODERNOS que utilizam SRAM de ALTA VELOCIDADE para poder oferecer um ALTO DESEMPENHO).

- Lógica reconfigurável versus ASIC (Vantagens dos FPGA'S/ Vantagens dos RECONFIGURÁVEIS):



## Vantagens dos reconfiguráveis (contra ASIC)

- Prototipação rápida (menor *time-to-market*)
- Fácil de corrigir 'bugs'
- Menor custo de projeto
- Adequado a aplicações de baixo volume

- O principal competidor da LÓGICA RECONFIGURÁVEL é o ASIC (chip fabricado para LÓGICAS DEDICADAS).

- A principal APLICAÇÃO de FPGA's também será em CONSTRUÇÃO de LÓGICA DEDICADA.

- Como ambas as TECNOLOGIAS estão voltadas para o mercado de LÓGICA DEDICADA é nesse ponto que elas vão COMPETIR:

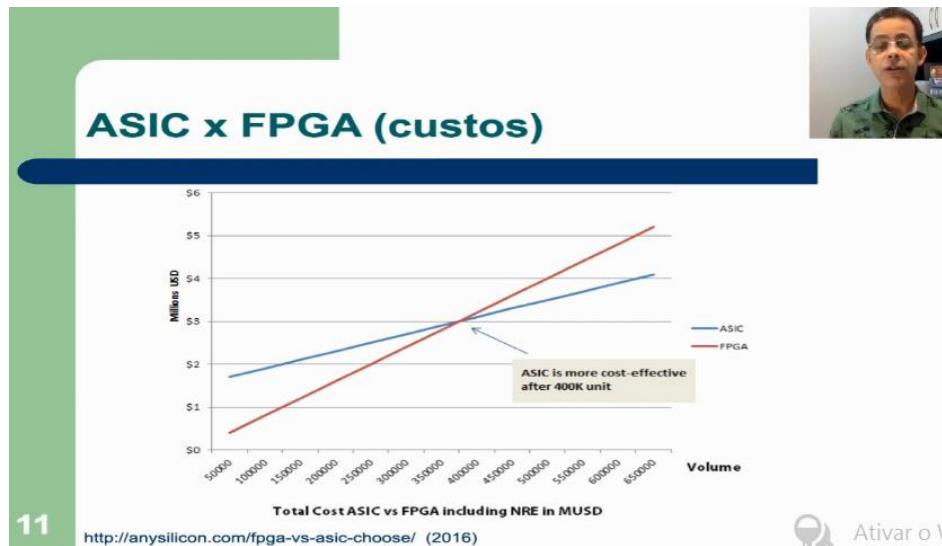
- É MAIS RÁPIDO fazer uma PROTOTIPAÇÃO utilizando FPGA do que utilizando ASIC's. Você pode comprar um FPGA numa loja sem esperar que ele seja FABRICADO (como é com um ASIC), faz sua programação, grava nele e já tem para si um HARDWARE DEDICADO para implementar sua aplicação (com FPGA's tem-se MENOR "time-to-market" que com ASIC's).

- É MAIS RÁPIDO de se corrigir DEFEITOS em FPGA's (se um chip com FPGA estiver de fácil acesso e for notado comportamento inadequado pode-se facilmente programa-lo novamente para corrigir seus erros e gravá-lo de novo para obtenção de um novo HARDWARE DEDICADO que satisfaça suas demandas). Em ASIC's dependendo da FASE em que o PROJETO esteja pode ser MUITO CARO fazer uma CORREÇÃO é preciso MAIS CUIDADO e ORIENTAÇÃO com a EQUIPE DE FABRICAÇÃO (PRINCIPALMENTE O ASIC já estando pronto como produto).

- É MUITO MAIS BARATO fabricar uma APLICAÇÃO usando FPGA (dado ao tempo MAIS CURTO do projeto principalmente) do que ASIC.

- FPGA's são MAIS ADEQUADOS para APLICAÇÕES DE BAIXO VOLUME o que vai ser outro ponto de REDUÇÃO DE CUSTOS comparativamente o FPGA com o ASIC. Quando tem-se uma aplicação que não tem uma GRANDE ESCALA (é de BAIXO VOLUME) é INVIÁVEL mandar FABRICAR um CHIP (ASIC) para sua aplicação o que dá chance para a utilização de uma LÓGICA DEDICADA que dê um DESEMPENHO MAIS ELEVADO.

- A imagem abaixo mostra essa comparação entre os CUSTOS de aplicações com FPGA's e com ASIC's:



Ativar o V

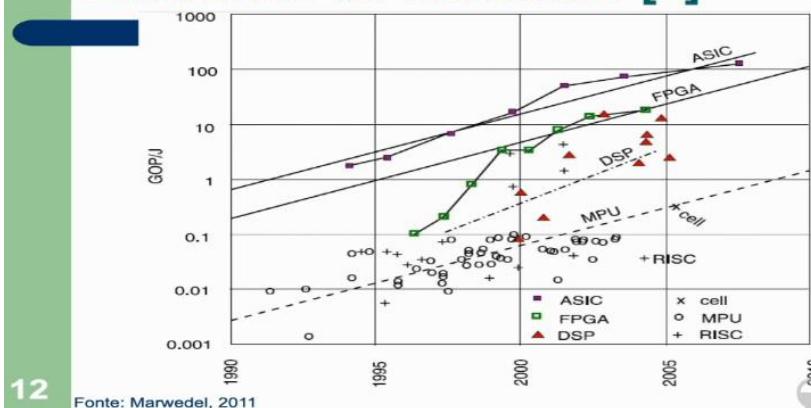
- A imagem mostra que EXISTE UM PONTO relacionado a uma CERTA QUANTIDADE de CHIPS que precisamos em nosso PROJETO onde DESSA QUANTIDADE PARA FRENTE caso precisemos de MAIS CHIPS o CUSTO FPGA passará a ser MAIS CARO do que o CUSTO do ASIC.
- Mas se a EXPECTATIVA de USO de CHIPS no produto for MAIS BAIXA, então o FPGA passa a ser MAIS COMPETITIVO com os ASIC's.

#### - Eficiência dos HARDWARES RECONFIGURÁVEIS:

- Traz-se DE VOLTA a figura abaixo que mostra esse gráfico comparativo entre as várias FORMAS de se IMPLEMENTAR O PROCESSAMENTO nas APLICAÇÕES EMBARCADAS.
- A LINHA TRACEJADA são os PROCESSADORES COMUNS.
- Note que TANTO o FPGA quanto o ASIC estão MUITO PRÓXIMOS, isso porque são DUAS ALTERNATIVAS muito EFICIENTES para se IMPLEMENTAR aplicações embarcadas.
- Porque no gráfico abaixo o ASIC é colocado como MAIS EFICIENTE que o FPGA?
  - Resposta: Porque de modo GERAL, o ASIC implementa a LÓGICA DE HARDWARE da maneira MAIS SIMPLES. Para se fazer uma SIMPLES LÓGICA de um FPGA se GASTA muito mais TRANSISTORES do que pra se fazer uma LÓGICA num ASIC. Essa LÓGICA IMPLEMENTADA a BASE DE MEMÓRIAS utilizada no FPGA cobra uma “CONTA MAIS ALTA” na ENERGIA fazendo com que sua EFICIÊNCIA seja MAIS BAIXA se comparado com o ASIC por outro lado se ele for comparado com o PROCESSADOR (GERAL ou ESPECÍFICO) o FPGA se torna uma ALTERNATIVA ATRATIVA.
- É IMPORTANTE quando se fala de VANTAGENS/DESVANTAGENS de FPGA's que saibamos COM QUEM estamos COMPARANDO ele e em que TIPO DE SITUAÇÃO:
  - Se compararmos ele com o ASIC, ele terá VANTAGENS em relação a FLEXIBILIDADE e o TEMPO DE PROJETO e DESVANTAGENS em relação a EFICIÊNCIA ENERGÉTICA.
  - Se compararmos o FPGA com o PROCESSADOR, seja ele de USO GERAL ou ESPECIALIZADO, o FPGA vai ter como DESVANTAGEM ser MENOS FLEXÍVEL do que o PROCESSADOR e como VANTAGEM ser MAIS EFICIENTE ENERGETICAMENTE falando.



## Eficiência do hardware [2]



Ativar o V

- Aplicações dos HARDWARES RECONFIGURÁVEIS (FPGA's):

## Aplicações



- Lógica complementar (*glue logic*)
- Sistemas completos – SoC (*System on Chip*)
- Co-processadores (funções matemáticas complexas)
- Computação de alto desempenho
- Etapa intermediária para um projeto ASIC

- São candidatos a UTILIZAR o FPGA praticamente as MESMAS APLICAÇÕES que são candidatas a USAR o ASIC já ambos são formas de se implementar CIRCUITO DEDICADOS (SoC's, glue logic, Co-processadores e outros na imagem acima).

- Agora tente-se para o ÚLTIMO EXEMPLO da IMAGEM ACIMA: “Etapa intermediária para um projeto ASIC”. Então, quando se tem um projeto que vai um CHIP DEDICADO (que nós temos CERTEZA de que ele vai ser utilizado) uma alternativa para se TESTAR esse produto ANTES de efetivamente COTRAR a FABRICAÇÃO DO CHIP (o ASIC) é utilizar o FPGA nesse processo, determinada EQUIPE DE DESENVOLVIMENTO vai utilizar o FPGA para VALIDAR o RESTO DO PRODUTO conectado aquele CHIP que faz a lógica dedicada. A medida que o projeto for ficando mais maduro o fabricante PASSA A CONTRATAR um CHIP ESPECIALIZADO para aquele produto e na placa SUBSTITUI o FPGA por um NOVO CHIP passando a USUFRUIR dos BENEFÍCIOS DE ESCALA do ASIC sem correr MUITOS RISCOS NO DESENVOLVIMENTO pelo fato de estar IMPLEMENTANDO/AMADURECENDO o PRODUTO utilizando de toda a FLEXIBILIDADE que o FPGA usado OFERECE nessa sua FUNÇÃO mais focada em “SUPORTE COMERCIAL”.

\*\*\***(Obs.)** Para saber mais sobre CPLD's (e FPGA's):

[https://edisciplinas.usp.br/pluginfile.php/530826/mod\\_resource/content/1/DISPOSITIVOS%20L%C3%93GICOS%20PROGRAM%C3%81VEIS\\_2014.pdf#:~:text=CPLD%20Complex%20Programmable%20Logic%20Devices,outro%20atrav%C3%A9s%20de%20interconex%C3%B5es%20program%C3%A1veis.](https://edisciplinas.usp.br/pluginfile.php/530826/mod_resource/content/1/DISPOSITIVOS%20L%C3%93GICOS%20PROGRAM%C3%81VEIS_2014.pdf#:~:text=CPLD%20Complex%20Programmable%20Logic%20Devices,outro%20atrav%C3%A9s%20de%20interconex%C3%B5es%20program%C3%A1veis.)

- [Exemplos de FABRICANTES de FPGA's e CPLD's](#) (CPLD é uma outra sigla para se representar LÓGICA RECONFIGURÁVEL muito parecida com os FPGA's):

**Fabricantes (FPGA e CPLD)**

- Xilinx (1985)
- Altera
- Lattice, Atmel, Actel, Quicklogic e SiliconBlue

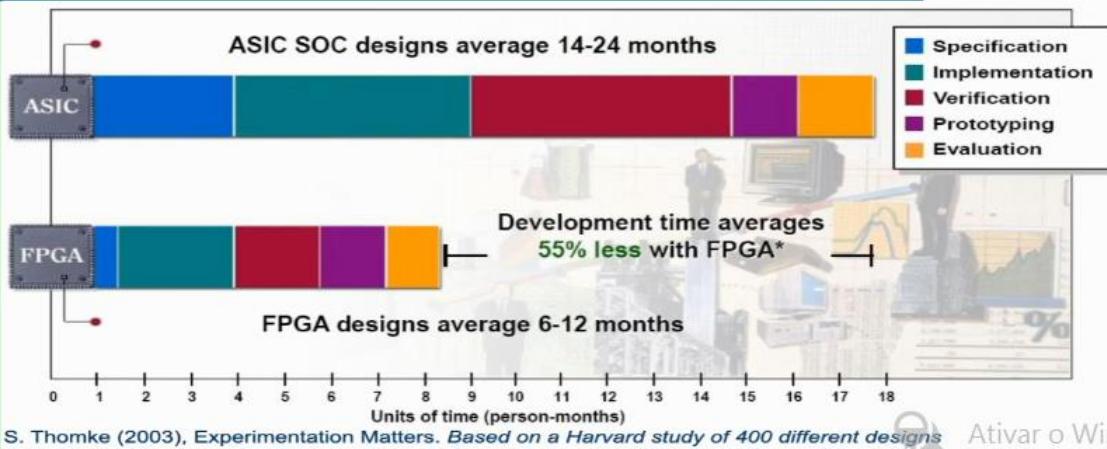
- [Na imagem abaixo](#) veja que tanto a Xilinx como a Altera DIVIDEM QUASE TODO o MERCADO de DISPOSITIVOS PROGRAMÁVEIS hoje em dia:



- Apenas um adendo, “PLD” é a SIGLA GENÉRICA dada para “DISPOSITIVOS PROGRAMÁVEIS” (FPGA e CPLD são TIPOS de PLD's).

- [Na imagem abaixo](#) é interessante ver o TEMPO DE DESENVOLVIMENTO de um PROJETO quando ele é DESENVOLVIDO sob plataforma FPGA e ou sobre uma plataforma ASIC:

## Redução no ciclo de desenvolvimento



- Tem-se mais ou menos os MESMOS TIPOS de ETAPAS para ambos os projetos (divididas em cores diferentes) mas note que com FPGA's o TEMPO GASTO em cada uma delas é BEM MENOR que com ASIC's.
- Vamos agora descobrir como combinar a "bem sucedida" estratégia do SoC com HARDWARE RECONFIGURÁVEL(FPGA/PLD):

- Uma das formas de se fazer isso é chamada de **PROCESSADOR SOFT-CORE**:

### Processadores Soft-Core

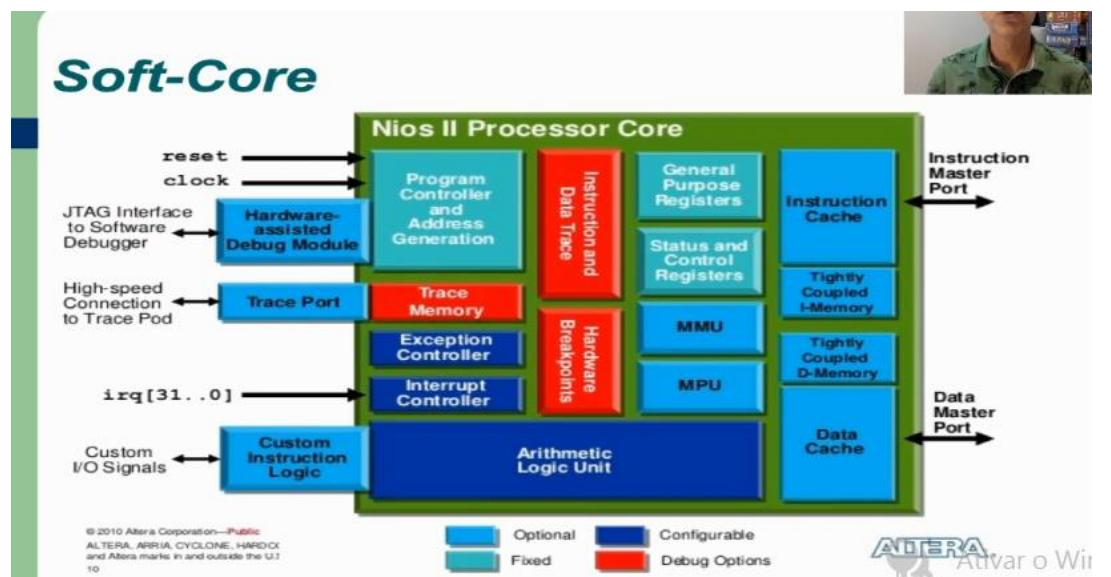
- MicroBlaze – XILINX
- Nios II – ALTERA
- OpenRISC – open source
- LEON3 – open source
  - Implementar modificações no processador
  - Usar o processador pra cuidar de atividades simples (interface)

18:13 / 27:13

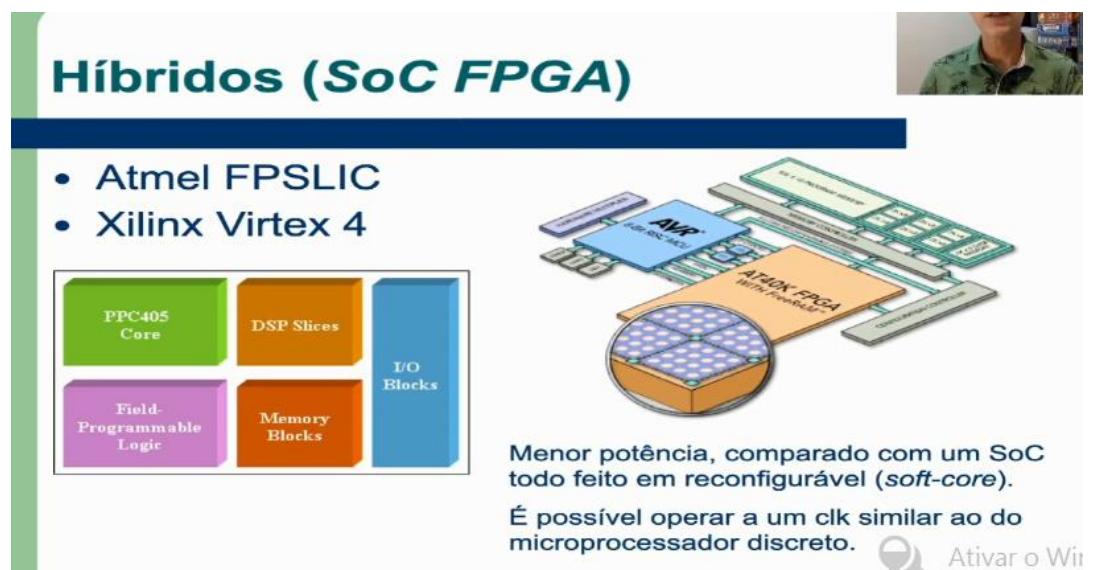
- A imagem acima apresenta alguns exemplos de processadores desse tipo.
- A ideia de "Soft" aí é que o processador vai ser descrito numa LINGUAGEM DE DESCRIÇÃO DE HARDWARE.
- (Como é o processador numa “Linguagem de Descrição de Hardware”) Então, existe um chip FPGA que pode ASSUMIR qualquer CIRCUITO LÓGICO COMBINACIONAL ou SEQUENCIAL ali, como o PROCESSADOR não deixa de ser um SISTEMA DIGITAL a ideia aqui é que nós temos uma BIBLIOTECA de SOFTWARE que nós ENCORPORAMOS ao nosso PROJETO e essa biblioteca REPRESENTARÁ o PROCESSADOR e SEUS DEMAIS COMPONENTES no mesmo. Montar o processador nesse caso significaria ABRIR o SOFTWARE de MONTAGEM do PROCESSADOR (nesse software se tem disponível várias bibliotecas onde se tem o processador, muitas vezes em diversas configurações DIFERENTES, VÁRIAS FORMAS de MEMÓRIA que se deseja utilizar e VÁRIOS DISPOSITIVOS DE E/S).

- (Funcionamento do softcore) Basta ABRIR A FERRAMENTA e COMBINAR as PARTES que se DESEJA utilizar e aquela ferramenta GERA uma DESCRIÇÃO do que É seu SoC (nisso contém-se informações como a versão do processador escolhido, as memórias escolhidas e os dispositivos E/S que foram escolhidos para se usar no projeto). A partir de então a FERRAMENTA GERA uma DESCRIÇÃO do PROCESSADOR e esse processador É GRAVADO no FPGA, depois desse processo é só programar tal SoC da mesma forma que se programa qualquer outro apenas com a diferença de que foi NÓS que CONFIGURAMOS o SoC com os COMPONENTES que NOS INTERESSAVAM.

- A imagem abaixo traz o que o FABRICANTE disponibiliza na FERRAMENTA DE CONFIGURAÇÃO de SoC do processador **Nios II**:



- Outra forma de se fazer isso é chamada de **HÍBRIDA (Sistemas SoC e FPGA)**:



- Nome “HÍBRIDO” é como o professor Elias chama kkkk.

- A imagem acima traz dois exemplos de CHIPS HÍBRIDOS no mercado.

- Os “HÍBRIDOS” são uma solução DIFERENTE da “SOFTCORE” apresentada anteriormente porque naquela solução “SOFTCORE” TUDO É PROGRAMÁVEL (ou seja, existirá um CHIP FPGA completamente PROGRAMÁVEL, poderia-se chamá-lo de “COMPLETAMENTE LIMPO”),

e nesse CHIP vai-se GRAVAR o PROCESSADOR, a MEMÓRIA, os DISPOSITIVOS de E/S e vamos gravar também o SOFTWARE de APLICAÇÃO nas MEMÓRIAS desse CHIP).

- Nos HÍBRIDOS vamos ter UMA PARTE DO CHIP feita de LÓGICA RECONFIGURÁVEL (uma “malha” de FPGA que pode ser programada com HARDWARES ADICIONAIS que nos interesse de colocar na aplicação que desencolvermos) e a OUTRA PARTE vai ser o PRÓPRIO PROCESSADOR feito num MODELO DE LÓGICA NÃO-CONFIGURÁVEL, ou seja, ali no CHIP existirá um PROCESSADOR que NÃO É FLEXÍVEL (ele é o PROCESSADOR que JÁ ESTÁ DEFINIDO PELO FABRICANTE).

- Nos “SOFTCORE” qualquer PROCESSADOR poder ser montado na aplicação contanto que se tenha a SUA DESCRIÇÃO e a SUA BIBLIOTECA, com isso pode-se montar seu SoC no FPGA de escolha.

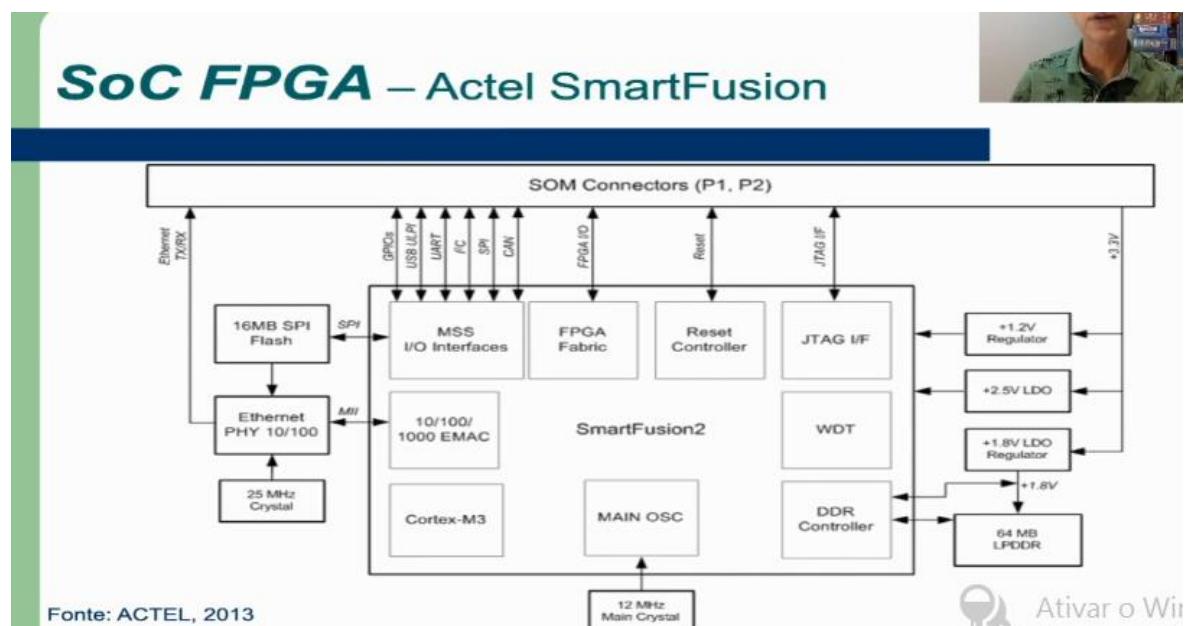
- Nos “HÍBRIDOS” já se tem um PROCESSADOR PRONTO tendo-se FLEXIBILIDADE apenas para ACRESCENTAR HARDWARES ao LADO DESSE PROCESSADOR.

- CHIPS HÍBRIDOS oferecem como VANTAGEM:

- POTÊNCIAS MAIS BAIIXAS em relação a estratégia SOFTCORE, pois o processador NÃO É IMPLEMENTADO utilizando LÓGICA RECONFIGURÁVEL estando, assim, no MESMO NÍVEL que os ASIC's (hardwares construídos a base de PORTAS LÓGICAS). Portanto, a parte do chip que é o PROCESSADOR consegue ter a MESMA EFICIÊNCIA ENERGÉTICA de um ASIC só perdendo EFICIÊNCIA ENERGÉTICA na parte do chip que é um FPGA (NOS HÍBRIDOS HÁ UMA PARTE COM EFICIÊNCIA ENERGÉTICA COMPARÁVEL A UM ASIC E OUTRA PARTE COMPARÁVEL A UM FPGA).

- Conseguir operar a um CLK similar ao do microprocessador discreto.

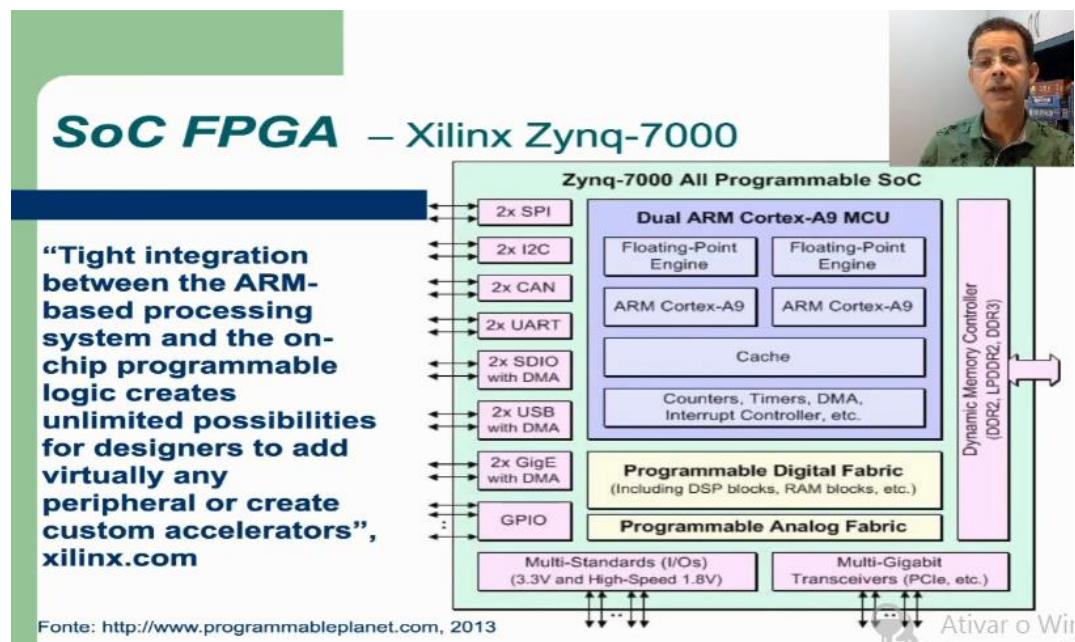
- A imagem abaixo mostra um exemplo “COMERCIÁVEL” de um SoC baseado nessa COMBINAÇÃO de PROCESSADORES FPGA (é um exemplo de HÍBRIDO comercial):



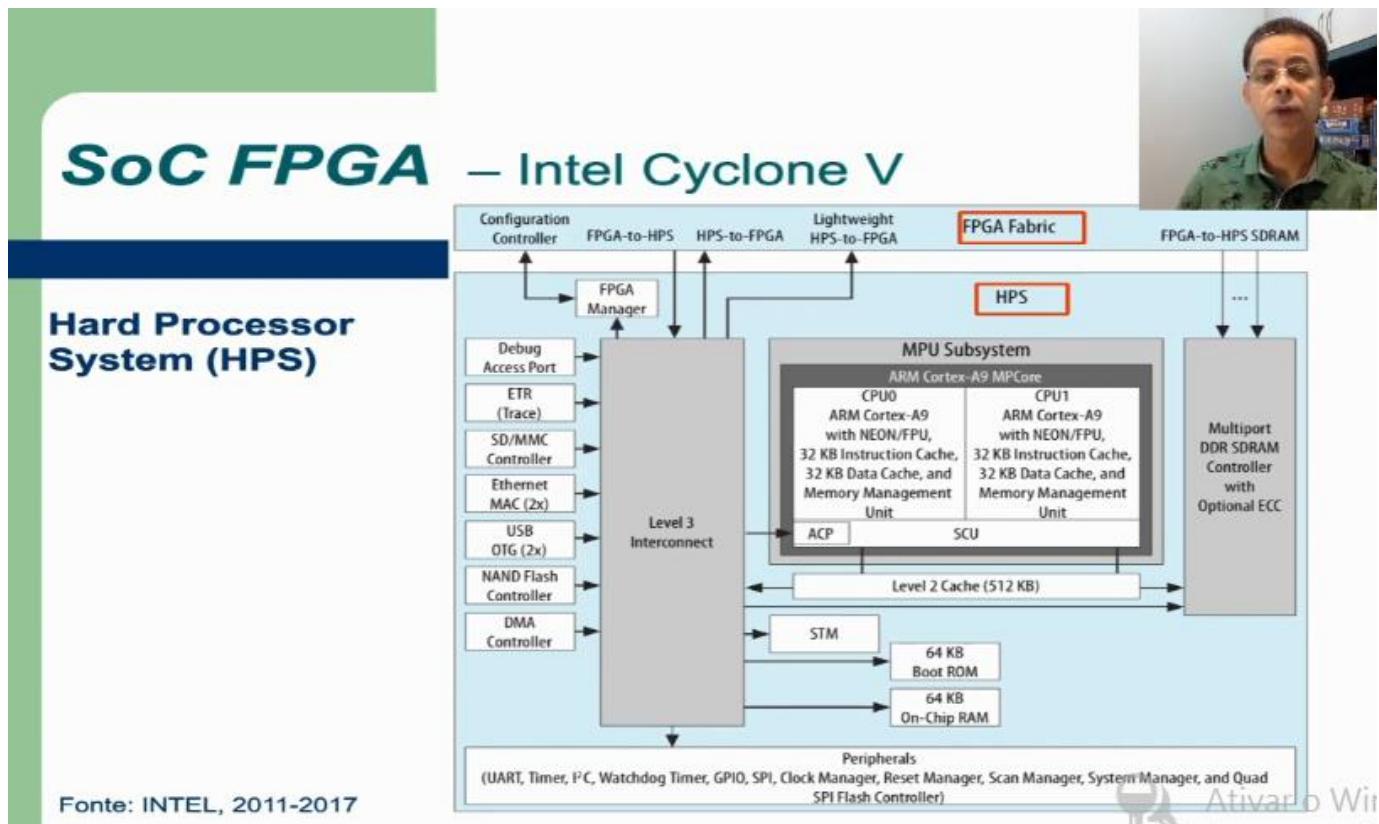
- Note pela imagem que essa forma HÍBRIDA é relativamente NOVA hoje em dia, na imagem acima se vê PRATICAMENTE EM QUASE TODA A FIGURA configurações de SoC's e pouco de FPGA's (no caso da figura só há uma caixinha onde se espera do

desenvolvedor adicionar o que lhe for mais útil pois todo o resto do sistema já está devidamente implementado e da forma MAIS EFICIENTE POSSÍVEL).

- A seguir, outro exemplo na imagem abaixo usando de um PROCESSADOR de MAIS ALTO DESEMPENHO:



- A seguir, outro exemplo na imagem abaixo, agora da Intel com PROCESSADOR DUAL CORE:



- Resumindo:

## Resumo



- Reconfigurável pode ser melhor que ASIC:
  - Prototipação rápida e baixo volume de fabricação
- FPGA oferece maior densidade e flexibilidade que CPLD
- Uso de HDL (*Hardware Description Language*) ainda é limitante para time-to-market
- Sistemas híbridos são *novidade* para reduzir tempo de projeto e potência

- Os **RECONFIGURÁVEIS** são **ALTERNATIVAS INTERESSANTES** aos **ASIC's**:

- Pois **PERMITEM** uma **PROTOTIPAÇÃO MAIS RÁPIDA**.

- São **ADEQUADOS** para **APLICAÇÕES** de **BAIXO VOLUME DE FABRICAÇÃO**.

- Os **FPGA** são a **PRINCIPAL TECNOLOGIA** de **RECONFIGURÁVEIS DISPONÍVEL**. Existem ainda os CPLD que são uma tecnologia **MAIS SIMPLES** QUE TAMBÉM OFERECE a POSSIBILIDADE de SER RECONFIGURÁVEL.

- **A PRINCIPAL LIMITAÇÃO** no **USO** de **FPGA's** são as **LINGUAGENS DE DESCRIÇÃO DE HARDWARE (HDL's)** já que EXISTE um NÚMERO LIMITADO de PESSOAS QUE TRABALHAM COM ELAS, além do fato de que as **FERRAMENTAS** que TRABALHAM COM ELAS tem seus "TIME-TO-MARKET" AUMENTADOS se comparado com PROCESSADORES.

- **O que se tem de MAIS NOVO** hoje em dia para **UTILIZAÇÃO** de **FPGA's** em **SEMB's** é essa **TECNOLOGIA** que foi chamada de "**HÍBRIDOS**" onde se tem num **ENCAPSULAMENTO TODO O CONJUNTO** de **COMPONENTES** de um SoC construído da maneira **MAIS EFICIENTE ENERGETICAMENTE** e além disso um **PEQUENO BLOCO** **FPGA** que poder ser **PROGRAMADO** para **ACRESCENTAR FUNÇÕES NOVAS** que não estão previstas inicialmente naquele SoC.

- Tipicamente, esses **BLOCOS DE FPGA's** existentes nesses **HÍBRIDOS** são **UTILIZADOS** para **IMPLEMENTAR ALGORITMOS COMPLEXOS** que demandariam um **ALTO TEMPO DE COMPUTAÇÃO DO PROCESSADOR** mas que **IMPLEMENTADOS** com **HARDWARE DEDICADO** oferecem uma **BOA EFICIÊNCIA ENERGÉTICA**.

//////////

- **Linguagens** (em **Sistemas Computacionais Embarcados**):

- Vamos falar aqui tanto de **LINGUAGENS** como de **COMPILEDORES** (dois elementos **BASTANTE** próximos).

- **LINGUAGENS**:

## Linguagens



- Objetivo: especificar a aplicação
- Linguagem = expressividade = abstração
- Linguagem = guia de implementação

- O **OBJETIVO** de qualquer **LINGUAGEM DE PROGRAMAÇÃO** é **ESPECIFICAR** a sua **APLICAÇÃO**.

- Através da **LINGUAGEM** pode-se **EXPRESSAR** o **COMPORTAMENTO** que se **DESEJA** para determinada **APLICAÇÃO**.

- AS LINGUAGENS são MECANISMOS de EXPRESSIVIDADE e também são uma FORMA DE ABSTRAÇÃO do COMPORTAMENTO que vai ser EXECUTADO pelo HADWARE nos níveis mais baixos da aplicação.

- A LINGUAGEM é quem COMANDA o PROCESSADOR e é uma forma de ABSTRAÇÃO do que ELE VAI FAZER quando estiver ACIONANDO seus CIRCUITOS ELETRÔNICOS.

- A LINGUAGEM também servirá para nós como um GUIA DE IMPLEMENTAÇÃO. As DIFERENTES LINGUAGENS possuem ESTRUTURAS DIFERENTES e elas acabam por NORTEAR a MANEIRA como o DESENVOLVEDOR vai EXPRESSAR o COMPORTAMENTO DA APLICAÇÃO o que pode, também, LIMITAR no que será POSSÍVEL FAZER em determinada LINGUAGEM.

- \*\*\*(Discussão) Quando discutir sobre LINGUAGENS procure SEMPRE PENSAR em QUE SITUAÇÃO ela é MAIS APROPRIADA e em QUE SITUAÇÃO ela NÃO É (NÃO FOQUE NA DISCUSSÃO DA LINGUAGEM EM SI).

- A medida que a COMPLEXIDADE DAS APLICAÇÕES vai CRESCENDO o PESO do ESFORÇO DE DESENVOLVIMENTO DE SOFTWARE vai AUMENTANDO PROPORCIONALMENTE em relação ao que se tinha em momentos ANTERIORES (onde o HARDWARE PREDOMINAVA em termos de ESFORÇO DE ENGENHARIA).

- A medida que as APLICAÇÕES ficam MAIS COMPLEXAS há necessidade de LINGUAGENS QUE FACILITEM O DESENVOLVIMENTO tornando o TEMPO DE DESENVOLVIMENTO MENOR e que FACILITEM A MANUTENÇÃO e o REAPROVEITAMENTO desse CÓDIGO em APLICAÇÕES FUTURAS.

- No contexto de APLICAÇÕES EMBARCADAS existem algumas LINGUAGENS que são as MAIS COMUNS de se usar, nosso professor optou por DIVIDÍ-LAS em 3 GRANDE GRUPOS (mostrados na imagem abaixo):

## Linguagens

- Assembly
  - Usado nos primórdios, quando o código era pequeno
- C
  - “Assembly estruturado”
- Java/Kotlin/Python (POO)
  - O aumento da complexidade requer elevação do nível das linguagens

- Assembly: É o GRUPO das LINGUAGENS DE MONTAGEM (montagem do código Assembly no caso). Eram muito usadas nos primórdios quando não existiam compiladores e se tinha CÓDIGOS MUITO PEQUENOS. >>>

>>> (Explicação) Maior parte do esforço de desenvolvimento dos produtos, como já foi dito antes, estava no HARDWARE e o que se fazia de linguagem de desenvolvimento de software era muito menos, então, nesse contexto programar em ASSEMBLY fazia sentido, sem falar que não haviam tantas linguagens como hoje e não havia compiladores das linguagens disponíveis para os microcontroladores que se usavam nessa época.

- C: É o GRUPO das LINGUAGENS ESTRUTURADAS. São costumeiramente chamadas de “Assembly estruturado” pois acabam por ser LINGUAGENS que FAZEM MUITA COISA que o

ASSEMBLY PERMITIA FAZER (no sentido de “LIBERDADE” do programador fazer coisas sem uma LINGUAGEM que IMPONHA MUITAS REGRAS) e são também LINGUAGENS, como o próprio nome diz, “ESTRUTURADAS” (o que não acontecia com o ASSEMBLY). >>>

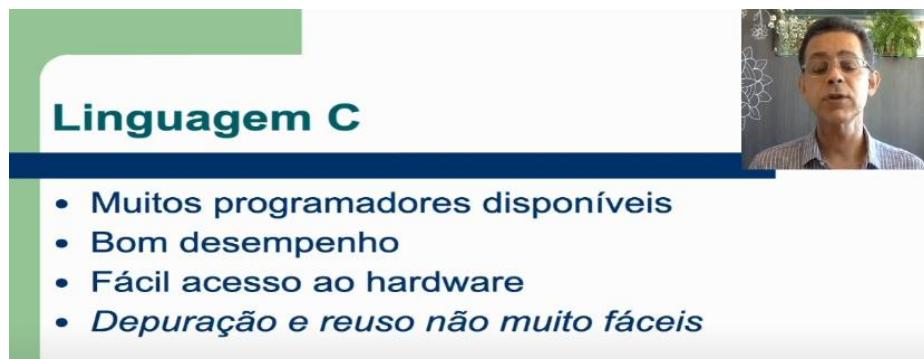
>>> (Explicação) Com o advento das linguagens ESTRUTURADAS, C e outras linguagens foram SURGINDO. Elas acabaram por se tornar OPÇÕES INTERESSANTES o que fez SURGIR um grande número GRANDE de compiladores para os MICROCONTROLADORES que são utilizados em APLICAÇÕES EMBARCADAS.

- Java/Kotlin/Python: É o GRUPO das LINGUAGENS DE ALTO NÍVEL (POO) (esse grupo é o que TEMOS HOJE EM DIA, em sua grande parte composto por LIGUAGENS POO). Nos dão MUITO MAIS RECURSOS para ter uma APLICAÇÃO DESENVOLVIDA da MANEIRA CERTA usando as MELHORES PRÁTICAS de DESENVOLVIMENTO DE SOFTWARE. Essas LINGUAGENS SÃO FUNDAMENTAIS com o CRESCIMENTO da COMPLEXIDADE das APLICAÇÕES (principalmente no nível a que se chegou hoje em dia) >>>

>>> (Explicação) Com o desenvolvimento das tecnologias de programação e o surgimento de novas ferramentas OUTRAS LINGUAGENS SURGIRAM oferecendo MUITO MAIS RECURSOS e AJUDANDO AINDA MAIS o programador no desenvolvimento das suas aplicações.

- Vamos então, FALAR de ALGUMAS DESSAS LINGUAGENS:

- Linguagem C:



- Muitos programadores disponíveis
- Bom desempenho
- Fácil acesso ao hardware
- Depuração e reuso não muito fáceis

- O USO DE PONTEIROS em C é o que o nosso professor chama de “ASSEMBLY ESTRUTURADO” já que em ASSEMBLY se TRABALHA MUITO com a ideia de PONTEIROS, o C OFERECE ISSO porque ELE SE PROPÓE a SER UMA LINGUAGEM para SUBSTITUIR O ASSEMBLY em DESENVOLVIMENTO de MUITO BAIXO NÍVEL.

- Acontece que PONTEIROS se tornam MUITAS VEZES elementos DIFÍCILS de se GERENCIAR pela maioria dos PROGRAMADORES.

- VANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:

- É a GRANDE DISPONIBILIDADE de PROGRAMADORES que USAM dessa LINGUAGEM (o que não deixa de ser um FATOR de REDUÇÃO DE CUSTOS e do TEMPO DE DESENVOLVIMENTO DA APLICAÇÃO).

- Se CONSEGUIR um BOM DESEMPENH (pelo fato de ser a LINGUAGEM MAIS PRÓXIMA da LINGUAGEM ASSEMBLY do que as demais outras).

- Ela NÃO IMPÕE BARREIRAS de ACESSO AO HARDWARE (diferente de outras LINGUAGENS, principalmente as que UTILIZAM MÁQUINA VIRTUAL que IMÕE RESTRIÇÕES por questões de SEGURANÇA o que causa desvantagens nesse meio).

- DESVANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:

- A DEPURAÇÃO e o REUSO do CÓDIGO NÃO SÃO FÁCEIS. Como a LINGUAGEM não utiliza de muitos FILTROS/RESTRIÇÕES e ainda DÁ ao programador MUITA LIBERDADE para fazer o que bem quiser isso acaba tornando-se um DIFICULTADOR DO REAPROVEITAMENTO desse CÓDIGO (essa LIBERDADE acaba por DIFICULTAR esse REAPROVEITAMENTO uma):

- O código acaba ficando MENOS LEGÍVEL.

- O código acaba ficando MENOS ESTRUTURADO.

- Torna o PROCESSO DE MANUTENÇÃO DO CÓDIGO mais TRABALHOSO.

- Linguagem Assembly (de montagem):



## Linguagem de montagem (Assembly)

- É voltada para controlar diretamente o circuito eletrônico do processador
- Fácil acesso ao hardware
- Desempenho ótimo
- Poucos programadores disponíveis
- Depuração difícil
- Reuso não muito fácil



- É uma LINGUAGEM NÃO ESTRUTURADA.

- Basicamente, as INSTRUÇÕES EM ASSEMBLY UTILIZAM REGISTRADORES como ESPAÇO DE MANIPULAÇÃO DE DADOS.

- NÃO EXISTE UMA LINGUAGEM ASSEMBLY “PADRÃO”! (CADA PROCESSADOR tem a sua PRÓPRIA LINGUAGEM DE MONSTAGEM, ou ASSEMBLY se assim preferir.)

- Essa LINGUAGEM EXISTE para FACILITAR o TRABALHO DO PROGRAMADOR daquela época de não ter que escrever o seu código UTILIZANDO SOMENTE NÚMEROS (que são o que a MÁQUINA ENTENDE).

- Os TERMOS das linguagens ASSEMBLY são chamados de MNEMÔNICOS: Palavras para AJUDAR O DESENVOLVEDOR a REPRESENTAR aqueles NÚMEROS BINÁRIOS que são as próprias INSTRUÇÕES DOPROCESSADOR.

- Foi FEITA para CONTROLAR DIRETAMENTE o CIRCUITO ELETRÔNICO DO PROCESSADOR.
- Ela não é algo muito intuitivo o que acaba por fazer o seu processo de programação ter um NÍVEL DE ABSTRAÇÃO MUITO BAIXO. Por isso, o processo “PENSAR” (do programador) nessa LINGUAGEM torna-se BEM COMPLICADO.
- Naturalmente seus códigos no começo eram MUITO PEQUENOS, e as funções das aplicações feitas por essas linhas eram MUITO SIMPLES se comparado ao que se pode fazer hoje (a linguagem ainda se mantém “simples” em aparência até hoje mas sua interpretação permanece complexa, ainda mais nos tempos de hoje em que esta tem estruturas de código **BEM MAIS COMPLEXAS** do que tinha em seus primórdios).
- VANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:
  - Oferece um ACESSO AO HARDWARE MAIS FÁCIL.
  - Oferece a MELHOR POSSIBILIDADE DE DESEMPENHO (pois estamos acionando os RECURSOS DO PROCESSADOR DIRETAMENTE e sem NENHUMA CAMADA DE ABSTRAÇÃO sobre isso).
- DESVANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:
  - A existência de POCOS PROGRAMADORES da LINGUAGEM no meio (não só são poucos os seus programadores como grande parte deles trabalha com LINGUAGENS ASSEMBLY DIFERENTES! Nem todos esses programadores portanto estão aptos a, muitas vezes, se trabalhar num mesmo projeto isso dependerá do processador abordado).
  - Por ser NÃO ESTRUTURADA ela torna o PROCESSO DE DEPURAÇÃO de uma APLICAÇÃO DIFÍCIL (aqui, não estamos falando de MILHARES DE LINHAS de CÓDIGO ASSEMBLY mas sim de um CÓDIGO COM POUCAS LINHAS que mesmo assim é de DIFÍCIL IDENTIFICAÇÃO e CORREÇÃO de ERROS em sua estrutura).
  - Por ser NÃO ESTRUTURADA ela, também, torna o PROCESSO DE REUSO DO CÓDIGO não MUITO FÁCIL (afinal, se o código ainda que pequeno é difícil de ser interpretado ele precisaria ser MUITO BEM ESCRITO seguido de ÓTIMA DISCIPLINA DE PROGRAMAÇÃO para que possa SER REAPROVEITADO em OUTROS PROJETOS).
  - A imagem abaixo traz dois exemplos de mnemônicos em ASSEMBLY de DOIS PROCESSADORES BASTANTE CONHECIDOS (Intel e ARM):

## Linguagem de montagem (Assembly)

- Intel x86

```
IMUL BX,12h  
MOV eax,5
```

- ARM

```
ADDS R1,R1,#1  
MOVS R2,#0
```



- Note a SIMPLICIDADE e ESTRUTURAÇÃO dessa LINGUAGEM.

- É essa SIMPLICIDADE que “DAR PODER” a essa LINGUAGEM pois com um CONJUNTO DE PEQUENAS INSTRUÇÕES que SÃO SIMPLES um programador ou compilador já consegue ATENDER as NECESSIDADES de QUALQUER APLICAÇÃO.

- [Linguagem Java \(Como EXEMPLO de LINGUAGEM ORIENTADA A OBJETO - POO\):](#)

## Linguagem Java



- Número de programadores elevado
- Depuração fácil
- Reuso fácil
- Portabilidade
- Desempenho bom (sem uso de JVM)
- Acesso ao hardware através de JNI (não direto)

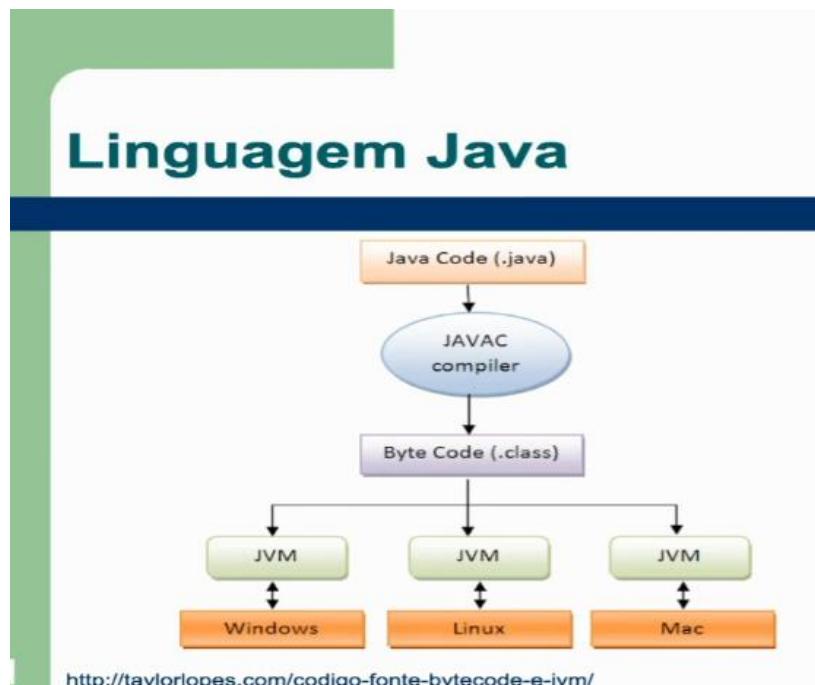
- Somente [A MÁQUINA VIRTUAL \(JVM\)](#) tem [ACESSO AO HARDWARE](#) (o programador JAVA não possui ACESSO DIRETO ao HARDWARE, ele sempre faz isso através de SERVIÇOS PRESTADOS PELA MÁQUINA VIRTUAL).

- [VANTAGENS](#) de [se programar](#) nessa [LINGUAGEM](#) para [APLICAÇÕES EMBARCADAS](#):

- Tem GRANDE DISPONIBILIDADE de PROGRAMADORES que USAM dessa LINGUAGEM.
- A DEPURAÇÃO nela é BEM MAIS FÁCIL (a DISCIPLINA que a LINGUAGEM impõe torna a LEGIBILIDADE DO CÓDIGO bem MAIOR além da própria POO).
- É de FÁCIL REUSO (É uma das CONSEQUÊNCIAS do ENCAPSULAMENTO imposto pela ORIENTAÇÃO A OBJETO).
- Tem PORTABILIDADE (É uma PARTICULARIDADE da LINGUAGEM JAVA, quando foi desenvolvida seus criadores tinham em mente que a linguagem ia ser utilizada para compilar o código e uma vez esse código compilado ele poderia ser executado em qualquer plataforma que tivesse COMPATIBILIDADE).
- Tem BOM DESEMPENHO (sem usar da JVM que quando utilizada pode se tornar um LIMITADOR DE DESEMPENHO).
- DESVANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:
  - A PORTABILIDADE (veja que isso é uma característica da linguagem de REALIDADE MUITO MAIS PRESENTE em MÁQUINAS DE MAIOR PORTE, em SISTEMAS DE MENOR PORTE EMBARCADOS essa PORTABILIDADE é MAIS DIFÍCIL de COLOCAR EM PRÁTICA). >>> Exemplo: É difícil ter um código que foi compilado para um servidor e que possa ser executado num microcontrolador, esse tipo de portabilidade fica MAIS no CAMPO TEÓRICO, no PRÁTICO a PORTABILIDADE existirá ENTRE PLATAFORMAS da MESMA CARACTERÍSTICA (como plataformas de desktop, se escrevermos um código num SO de DESKTOP WINDOWS ele vai rodar num SO de DESKTOP LINUX por exemplo).
  - JVM (É a MÁQUINA VIRTUAL que a LINGUAGEM JAVA foi desenvolvida para FUNCIONAR e costuma-se poder tornar-se um LIMITADOR DE DESEMPENHO já que a sua APLICAÇÃO precisa SER TRADUZIDA dos CÓDIGOS DE JAVA para a LINGUAGEM NATIVA DO PROCESSADOR onde aquela MÁQUINA VIRTUAL está FUNCIONANDO):
    - Existe uma SITUAÇÃO MUITO COMUM em APLICAÇÕES EMBARCADAS que USAM JAVA que é a TRADUÇÃO BINÁRIA do JAVA diretamente para o ASSEMBLY NATIVO. Então nós, \*\*\*por exemplo, programaríamos em JAVA e o RESULTADO DA COMPILAÇÃO não seriam BYTECODES que FUNCIONAM sob uma MÁQUINA VIRTUAL mas sim o ASSEMBLY NATIVO do PROCESSADOR ALVO, com isso, perde-se a PORTABILIDADE da LINGUAGEM JAVA mas ganha-se o DESEMPENHO de PASSAR DIRETO pela MÁQUINA VIRTUAL(JVM).
    - O fato do programador JAVA não possuir ACESSO DIRETO ao HARDWARE e sempre fazer isso através de SERVIÇOS PRESTADOS PELA MÁQUINA VIRTUAL pode TORNAR-SE uma LIMITAÇÃO quando a APLICAÇÃO EMBARCADA estiver sendo desenvolvida UTILIZANDO JAVA. Mas existem MECANISMOS para lidar com esse cenário:
      - JNI: É um MECANISMO, definido pelo PADRÃO JAVA, de ACESSAR o HARDWARE através de uma OUTRA LINGUAGEM. \*\*\*Por exemplo,

se escreveria o código que VAI ACESSAR O HARDWARE em C e esse código em C é ACESSADO pelo PROGRAMA JAVA o que faz com que o SEMB em questão passe a conseguir tanto controlar o HARDWARE como ainda assim continuar PROGRAMANDO EM JAVA.

- A figura abaixo traz um resumo DAS PRINCIPAIS CARACTERÍSTICAS da LINGUAGEM JAVA:



- No topo tem-se o CÓDIGO ESCRITO EM JAVA.
- O CÓDIGO no topo é TRADUZIDO pelo COMPILADOR (aqui seria o JAVAC) para uma LINGUAGEM “BYTE CODE” (que não deixam de ser um TIPO DE ASSEMBLY) que são uma espécie de “CÓDIGO BINÁRIO” entendido pela JVM.
- Cada PLATAFORMA que PRETENDE SER COMPATÍVEL com a LINGUAGEM JAVA (no exemplo acima temos Windows, Linux e Mac) precisam IMPLEMENTAR A JVM.
- É a JVM que se ENCARREGA de TRADUZIR dos “BYTE CODES” para a LINGUAGEM NATIVA de cada um desses ambientes, inclusive para os diferentes processadores que possam estar embaixo deles.
- \*\*\*(Dificuldade da JVM) Essa MÁQUINA VIRTUAL ONERA/DIFICULTA o DESEMPENHO de uma aplicação JAVA porque acaba introduzindo uma “CAMADA INTERMEDIÁRIA” no meio de todo esse esquema que abrange a LINGUAGEM e sua APLICAÇÃO:
  - \*\*\*(Solução) Porém, é muito comum ter MÁQUINAS VIRTUAIS JAVA que são INCORPORADAS DIRETAMENTE ao CÓDIGO TRADUZIDO (isso significa dizer que o compilador para aquela plataforma TRADUZ DE JAVA para a ARQUITETURA e NÃO PARA “BYTE CODES” o que ELIMINA o uso da JVM e passa a dar a linguagem DESEMPENHOS BEM MELHORES, como os vistos em C).

- Java versus C:



## Linguagem Java [2]

- Tipos de dados mais fortes que C
- Sem uso de ponteiros (referência a objetos)
- Garbage Collector

- Outra característica interessante de JAVA comparado com C é a TIPAGEM DE DADOS que é MAIS FORTE em JAVA do que em C (os TIPOS DE DADOS em C são FLEXÍVEIS):

- \*\*\*(exemplo) Em C consegue-se COPIAR uma VARIÁVEL do TIPO INTEIRO numa VARIÁVEL TIPO FLOAT, e o máximo que o compilador faz é nos dar um aviso mas ele ainda compila e gera um código executável pra isso, em JAVA esse tipo de coisa não é POSSÍVEL (só pode-se COPIAR VARIÁVEIS de um TIPO em OUTRAS do MESMO TIPO).

- Diferente de como é com C a linguagem JAVA NÃO USA PONTEIROS, só consegue-se ter REFERÊNCIAS a OBJETOS e não a POSIÇÕES DE MEMÓRIA o que AJUDA O PROGRAMADOR a COMETER MENOS ERROS e a PERCEBÊ-LOS MAIS FACILMENTE com a ajuda do COMPILADOR.

- O “Gabage Collector” (Coletor de Lixo) oriundo da LINGUAGEM JAVA é uma TAREFA da MÁQUINA VIRTUAL JAVA que se ATIVA AUTOMATICAMENTE sem dar ao PROGRAMADOR NOÇÃO e CONTROLE de QUANDO esse CÓDIGO vai ENTRAR EM AÇÃO e POR QUANTO TEMPO ele IRÁ OCUPAR O PROCESSADOR:

- Vai DIFICULTAR a IMPLEMENTAÇÃO de APLICAÇÕES que possuem RESTRIÇÕES DE TEMPO sobre o AMBIENTE DA MÁQUINA VIRTUAL JAVA.

### Linguagem Python:



## Linguagem Python

- Número de programadores elevado e crescente
- Tipos de dados fortes
- Depuração fácil
- *MicroPython*
- *PyMite*

- É outra linguagem na mesma linha de JAVA que oferece MUITOS RECURSOS para o PROGRAMADOR.

- VANTAGENS de se programar nessa LINGUAGEM para APLICAÇÕES EMBARCADAS:

- É a GRANDE DISPONIBILIDADE de PROGRAMADORES que USAM dessa LINGUAGEM (\*Obs. Muitas vezes ao colocarmos uma pessoa conhecida da programação dessa linguagem, e outras semelhantes, na prática para desenvolver uma aplicação embarcada pode acabar por vir a ser uma experiência DIFÍCIL no primeiro momento, isso porque ao se programar PYTHON/JAVA para máquinas de GRANDE PODER COMPUTACIONAL dá a vantagem de muitas vezes dispor-se de conjuntos de

bibliotecas para disposição do programador o que provavelmente não será encontrado nos ambientes embarcados de desenvolvimento necessitando que o programador se “adapte” e “re-eduque” seu uso dos recursos disponíveis).

- Possui FORTE TIPAGEM de DADOS (o que ajuda na DEPURAÇÃO do CÓDIGO e AUMENTA a PRODUTIVIDADE dos PROGRAMADORES).

- Seu código é de FÁCIL DEPURAÇÃO.

- Como RECURSOS para se DESENVOLVER EM PYTHON em APLICAÇÕES EMBARCADAS temos na imagem acima os exemplos do MICROPYTHON e o PYMITE que aparentemente foi descontinuado):

- MicroPython:

The screenshot shows the MicroPython homepage. At the top right is a portrait of a man. Below it, the title "MicroPython" is displayed. To the right of the title is a small image of a person's face. The main content area lists features of MicroPython, followed by a table of supported boards and their memory specifications, and links to the official site and a GitHub repository.

**Features:**

- Implementação enxuta de Python 3;
- Subconjunto da biblioteca padrão do Python;
- Otimizado para rodar em microcontroladores e em ambientes restritos (256K de memória de código 16k de RAM)
- Placas que suportam:
  - PyBoard
  - ESP8266
  - ESP32
  - outras ...

	Flash	RAM
<b>PIC18F4550</b>	32KB	2KB
<b>Arduino MEGA</b>	256KB	8KB

<https://micropython.org>

- É UMA PLATAFORMA ABERTA de comunidade e ambiente BASTANTE POPULARES.

- Feito pra SER COMPATÍVEL COM PYTHON 3, porém tendo algumas coisas LIMITADAS em sua aplicação, por isso diz-se que é uma “IMPLEMENTAÇÃO ENXUTA” pois tiraram-se algumas coisas da sua linguagem para viabilizar essa ferramenta no microcontrolador escolhido como arquitetura alvo.

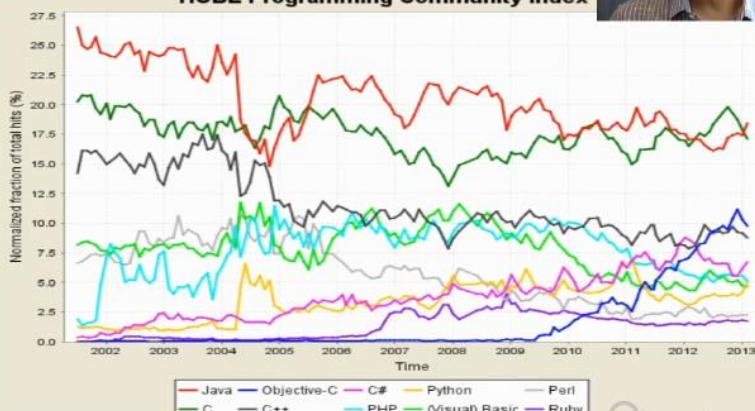
- A imagem ACIMA apresenta os pontos importantes da linguagem!

- Evolução da POPULARIDADE dessas e OUTRAS LINGUAGENS em vários ESTUDOS DIFERENTES:

- GRÁFICO [01]:

## Evolução do uso das linguagens programação [2]

TIOBE Programming Community Index



<http://www.tiobe.com>

Ativar o Wiz

- **GRÁFICO [02]:**

## Evolução do uso das linguagens programação [3]

Language Types (click to hide)



Mobile

Enterprise

Embedded

Language Rank	Types	Custom Ranking
1.	C	99.2
2.	C++	95.6
3.	Assembly	72.2
4.	Arduino	60.4
5.	Haskell	50.0
6.	D	32.6
7.	Ada	27.5
8.	VHDL	26.3

<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017>

Ativar o Wiz

- **GRÁFICO [03]:**

## Evolução do uso das linguagens programação [4]

Language Types (click to hide)



Mobile

Enterprise

Embedded

Language Rank Types Spectrum Ranking

1.	Python	100.0
2.	C++	99.7
3.	C	96.7
4.	Assembly	74.1
5.	Arduino	69.0
6.	Haskell	48.6
7.	VHDL	45.4
8.	Verilog	41.2

<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>

Ativar o Wiz

- TABELA COMPARATIVA: “Assembly” vs “C” vs “JAVA/PYTHON”:



	Assembly	C	Java
Memória	++	+	+/-
Time-to-market	--	+/-	+
Tempo de execução	++	+	+/-

- Para interpretação da tabela acima entende-se que:

- “++”: **ÓTIMO.**
- “+/-”: **BOM.**
- “--”: **RUIM.**

- Conclusões a partir do observado na tabela:

- MEMÓRIA:

- ASSEMBLY é a que GASTA MENOS MEMÓRIA (código fica muito enxuto, as variáveis representadas da maneira mais simples possível).
- C consegue ser BOA no uso de memória não sendo TÃO EFICIENTE quanto a LINGUAGEM ASSEMBLY (basicamente pelo processo de compilação que acaba introduzindo alguns custos adicionais).
- As LINGUAGENS DE ALTO NÍVEL DE ABSTRAÇÃO (JAVA/PYTHON), principalmente pela POO, acabam criando estruturas adicionais de controle que vão fazer com que elas GASTEM AINDA MAIS MEMÓRIA do que linguagens como C e ASSEMBLY.

- TIME-TO-MARKET:

- Se uma APLICAÇÃO é feita TODA EM ASSEMBLY isso levaria um TEMPO MUITO GRANDE e ATRASARIA DEMAIS o PROJETO.
- Se uma APLICAÇÃO é feita TODA em uma LINGUAGEM DE ALTO NÍVEL termina-se o PROJETO em MUITO MENOS TEMPO do que em LINGUAGENS C e MAIS AINDA em ASSEMBLY.

- TEMPO DE EXECUÇÃO:

- Naturalmente a LINGUAGEM que consegue fazer o MELHOR USO dos RECURSOS DOS PROCESSADORES é a ASSEMBLY que acaba por nos dar o MELHOR TEMPO DE EXECUÇÃO.

- As LINGUAGENS DE ALTO NÍVEL DE ABSTRAÇÃO (JAVA/PYTHON) que oferecem mais RECURSOS DE ABSTRAÇÃO, HERANÇA, HIERARQUIAS e etc, ABSTRAÇÕES que além de CONSUMIREM MAIS MEMÓRIA também levam o PROCESSADOR a FAZER MAIS CHAMADAS e CONSUMIR MAIS TEMPO DE PROCESSAMENTO, o que dá a essas linguagens o PIOR TEMPO DE EXECUÇÃO DOS 3 GRUPOS QUE DIVIDIMOS.

- TABELA COMPARATIVA em relação ao DESENVOLVIMENTO HISTÓRICO de CARACTERÍSTICAS que MUITO INTERESSAM quando se produzem PRODUTOS EMBARCADOS:



- (Desafio - 01)\_A cada ano se consegue fazer CHIPS MAIS RÁPIDOS, com MAIOR DENSIDADE DE INTEGRAÇÃO e MAIOR EFICIÊNCIA ENERGÉTICA, entretanto, nossa CAPACIDADE DE DESENVOLVER SOFTWARES de MANEIRA PRODUTIVA acaba NÃO COMPANHANDO essa VELOCIDADE. Ou seja, temos CHIPS cada vez MAIS RÁPIDOS mas temos DIFICULDADES em ESCREVER SOFTWARES que consigam USAR TODA A CAPACIDADE DESSES CHIPS.

- (Desafio - 02)\_Com a CAPACIDADE DAS BATERIAS acontece o mesmo problema que descrito no parágrafo acima, a CAPACIDADE DE ACUMULAÇÃO DE ENERGIA das nossas BATERIAS NÃO COMPANHA a nossa PRODUTIVIDADE DE SOFTWARE que, também, NÃO ACOMPANHA nossa CAPACIDADE DE FAZER CIRCUITOS INTEGRADOS cada vez MAIS RÁPIDOS.

- Ambos os dois parágrafos acima tratam de um DESAFIO que a INDÚSTRIA ENFRENTA HOJE EM DIA.

- COMPILEDORES:



## Compiladores

- Porque otimizar neste nível?
  - Arquiteturas dos processadores oferecem recursos especiais
  - Velocidade de compilação não é tão importante quanto otimização do código

- Porque os sistemas procuram introduzir OTIMIZAÇÕES no NÍVEL DA COMPILAÇÃO?

- Os processadores oferecem RECURSOS DIFERENTES e muitas vezes INTERESSANTES para tornar o SOFTWARE MAIS RÁPIDO, ou tornar o GASTO DE MEMÓRIA MENOR, ou até mesmo para AUMENTAR a EFICIÊNCIA ENERGÉTICA. TER NOCOMPILEADOR a capacidade de se EXPLORAR ESSES RECURSOS torna a VIDA DO DESENVOLVEDOR MAIS FÁCIL (ele vai fazer o programa dele e o compilador vai procurar gerar um código que aproveite os recursos que a plataforma de hardware tem disponível).
- Outra coisa interessante é que se o COMPILEADOR pra conseguir fazer “isso” (“isso” = procurar gerar um código que aproveite os recursos que a plataforma de hardware tem disponível) tiver que DEMORAR UM POUCO MAIS na sua COMPUTAÇÃO não acaba sendo um PROBLEMA GRAVE já que a compilação VAI ACONTECER SOMENTE UMA VEZ, o IMPORTANTE é que o CÓDIGO FINAL GERADO seja OTIMIZADO pra que a APLICAÇÃO que vai ser executada MILHÕES VEZES seja MAIS EFICIENTE.
- Outra CARACTERÍSTICA de OTIMIZAÇÃO importante está associada ao conceito já falado de ASIP, a imagem abaixo resume essa ideia:



## Outras otimizações (ASIP)

- Compilação para DSP
  - Registradores e operações especiais, organização da memória
- Compilação para processadores multimídia
  - “Paralelização” de laços
- Compilação para VLIW
- Processadores de rede

- O ASIP é um processador com um CONJUNTO DE INSTRUÇÕES ESPECIALIZADAS em um DETERMINADO GRUPO DE APLICAÇÃO.

- Falamos bastante em DSP's (é o exemplo MAIS POPULAR de ASIP) que oferecem em si otimizações para tornar o PROCESSO DE COMPUTAÇÃO mais RÁPIDO na aplicação e o COMPILEADOR é um FORTE CANDIDATO a SABER UTILIZAR essas INSTRUÇÕES.

- Tendo um COMPILEADOR que consiga explorar o PODER DO DSP teremos uma aplicação que FUNCIONE de FORMA MAIS EFICIENTE (tanto no

APROVEITAMENTO DA MEMÓRIA quanto na REDUÇÃO do TEMPO DE COMPUTAÇÃO).

- Estes são outros exemplos de processadores onde ter-se um compilador para explorá-los vale a pena (vistos na imagem acima): Multimídia, VLIW, e PROCESSADORES DE REDE.

- Ocorre que o COMPILADOR não consegue GERAR O CÓDIGO com a MESMA EFICIÊNCIA que um PROGRAMADOR EM ASSEMBLY conseguiria, afinal, o OBJETIVO DO COMPILADOR é PERMITIR EXPRESSAR a APLICAÇÃO numa LINGUAGEM DE ALTO NÍVEL e ele TRADUZIR AUTOMATICAMENTE dessa linguagem para a LINGUAGEM DE MONTAGEM NATIVA do PRÓPRIO PROCESSADOR, mas ele vai introduzir PERCAS DE DESEMPENHO SIM (a imagem abaixo apresenta algumas das principais percas):

## Perda de desempenho (compiladores C)

- Reportados tamanhos de código até 10x maiores
- Melhor fabricante reporta fator 2 (20% seria o desejável)
- Resultado: muito código em assembly (75% MCU, 90% DSP)



- Infelizmente, há caso em que o DESEMPENHO do código gerado pelo compilador pode ser 10 VEZES PIOR do que um CÓDIGO ESCRITO NATIVAMENTE na LINGUAGEM DO PROCESSADOR.

- Outros compiladores MELHORES conseguem um “FATOR 2” (que quer dizer que o código compilado é DUAS VEZES PIOR do que o CÓDIGO ESCRITO NATIVAMENTE na LINGUAGEM DO PROCESSADOR). Pegando “20%” como uma referência desejável, vemos que ainda estamos MUITO LONGE, até mesmo nos MELHORES CASOS, daí que se espera de um COMPILADOR quanto a CONSEGUIR PRODUZIR um código quem um BOM PROGRAMADOR ASSEMBLY CONSEGUIRIA escrevendo o código na LINGUAGEM NATIVA daquele processador.

- o QUE SE FAZER ENTÃO? (Baseado no apresentado pelos DOIS PARÁGRAFOS ACIMA)

- O resultado disso é que MUITO DO CÓDIGO que a gente USA em nossas aplicações foi escrito ORIGINALMENTE em ASSEMBLY (sendo 75% para MCU'S e 90% para DSP's):

- Quer dizer que se usarmos MCU's/DSP's vamos ter que programar em ASSEMBLY?

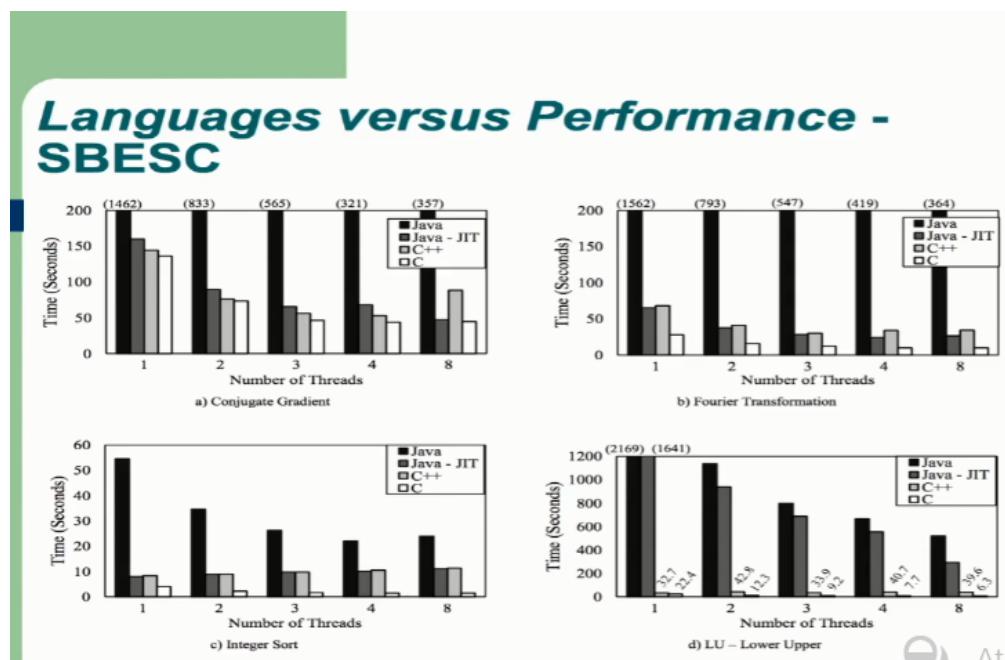
- Isso existe para CONTORNAR a LIMITAÇÃO DO COMPILADOR em GERAR esses CÓDIGOS de MANEIRA EFICIENTE.

- Não, o que estamos dizendo é que o código que está lá na aplicação no final 75% dele FOI ESCRITO EM ASSEMBLY, mas não quer dizer que você que fez a aplicação escreveu o código em ASSEMBLY. Veja o seguinte exemplo:

- Quando escrevemos uma aplicação para um microcontrolador como o PIC18 que você usa a função "Math.h" para implementação de um SENO/TANGENTE essas funções foram escritas em ASSEMBLY e estão lá na BIBLIOTECA "Math.h". Se sua aplicação chama a função SENO e usa o resultado para fazer alguma coisa uma parte da sua aplicação agora terá parte dela escrita em C e a outra parte escrita em ASSEMBLY já que a função SENO está sendo usada também.

- É isso que ACONTECE NA REALIDADE, TODAS AS APLICAÇÕES recorrem a uma GRANDE QUANTIDADE de BIBLIOTECAS por questões de EFICIÊNCIA NO PROCESSO DE DESENVOLVIMENTO (para se ganhar tempo já que a função está pronta na biblioteca) e por questões de EFICIÊNCIA DE DESEMPENHO (essas bibliotecas foram otimizadas e escritas em linguagem do mais próximo que o processador pode oferecer).

- Abaixo um estudo onde mediu-se o TEMPO DE COMPUTAÇÃO num MESMO COMPUTADOR para 4 FORMAS DIFERENTES de se AVALIAR os BENCHMARKS (BENCHMARKS aqui são as várias LINGUAGENS DE PROGRAMAÇÃO comparadas nos gráficos abaixo):



- \*\*\*(**IMPORTANTE!**) A melhor solução em vez de sempre optar por UM TIPO SÓ DE LINGUAGEM ao se fazer sua APLICAÇÃO EMBARCADA é COMBINÁ-LAS (uma parte de sua aplicação pode ser feita em ASEMBLY, talvez OUTRA em C, e talvez outra em JAVA/PYTHON)! COMBINAR ESSAS LINGUAGENS para g=chegar no resultado mais EFICIENTE para sua aplicação em vez de tentar chegar a isso usando somente UMA

DELAS é a melhor escolha que um programador de aplicações embarcadas pode fazer. A combinação DESSAS DIVERSAS FERRAMENTAS é que nos dá o RESULTADO ÓTIMO, e é preciso saber trabalhar com CADA UMA DELAS e saber COMBINÁ-LAS de forma a se FAZER A MELHOR APLICAÇÃO POSSÍVEL.



- \*\*\*ANÁLISE e OTIMIZAÇÃO de CÓDIGO em APLICAÇÕES EMBARCADAS:



- FINALMENTE, chegamos na última compilação do que foi dito pelo professor ELIAS sobre SISTEMAS COMPUTACIONAIS EMBARCADOS!!!

- Como ESCOLHER a PLATAFORMA ALVO para nossa APLICAÇÃO EMBARCADA?

- Pra isso temos que avaliar a QUANTIDADE DE MEMÓRIA que nossa APLICAÇÃO PRECISA e o TEMPO DE COMPUTAÇÃO, também, vai ser afetado por essa escolha.

- E a BATERIA? Como que se DEFINE A CAPACIDADE DE CARGA que a BATERIA TEM QUE TER pra manter nossa APLICAÇÃO FUNCIONANDO?

- Vamos procurar responder esses questionamentos acima! Abaixo um resumo do que vai ser bordado:

**Roteiro**

- Tempo de execução } Parte:01
- Energia } Parte:02
- Tamanho do código } Parte:03

- Como que a gente mede o TEMPO DE EXECUÇÃO de nossa APLICAÇÃO EMBARCADA:

**Medindo o tempo de execução**

**Não Invasivo**

- Simulador – programa que roda em uma *workstation* usando binário como entrada
  - Dificuldade/imprecisão pra código intensivo em I/O
- Medida em uma CPU real
  - Leituras do Relógio de Tempo Real (RTC)
  - Timer – Ativado em determinados trechos do código e lido depois (pode ser ativado por hw ou sw)
  - Analisador lógico ou Osciloscópio
    - Código precisa gerar eventos identificáveis (nos pinos)



- Pode-se fazer isso por meio de dois TIPOS de MÉTODOS:

- INVASIVO:

- É aquele que requer que se INSTRUMENTE determinado CÓDIGO para FAZER A MEDIÇÃO (precisa-se pegar o FONTE do código, modifica-lo e colocar algumas funções adicionais nele para que assim se possa medir o tempo de computação).

- NÃO- INVASIVO:

- Não precisam de ALTERAÇÃO DO CÓDIGO. O SIMULADOR permite que MONITORE-SE determinado código sem precisar alterá-lo (basta-se colocar “break-points”, por exemplo, no início e no final do código sob observação).

- Uso de SIMULADORES(Não-Invasivo):

- Uma das estratégias seria UTILIZAR UM SIMULADOR. Existem vários simuladores para SEMB's e esses simuladores são programas que vão funcionar em determinada estação de trabalho que o desenvolvedor utilize (como o próprio PC) recebendo o CÓDIGO BINÁRIO da APLICAÇÃO que FUNCIONARÁ em uma OUTRA PLATAFORMA e conseguindo EXECUTAR tal APLICAÇÃO em determinada estação.

- Através desses SIMULADORES consegue-se MEDIR o TEMPO DE COMPUTAÇÃO de determinada APLICAÇÃO de MANEIRA INDIRETA (já que esses simuladores irão possuir informações úteis para essas medições como o CLOCK e o TEMPO DE EXECUÇÃO das INSTRUÇÕES da PLATAFORMA ALVO).

- (Vantagens do método): Esse método de UTILIZAR SIMULADORES é muito bom quando se está focado EM UM ALGORITMO EM PARTICULAR.

- (Desvantagens do método): Quando se precisar fazer medições que envolvam ACESSO A DISPOSITIVOS DE E/S esse método não é tão bom porque em geral os simuladores NÃO TEM uma boa PRECISÃO na hora de REPRESENTAR esses dispositivos (o TEMPO DE REAÇÃO deles).

- Uso de medidas em UMA CPU REAL(Invasivo):

- Outra estratégia é dispor de um protótipo com a PRÓPRIA CPU onde determinado código vai ser executado.

- Esse método pode-se BASEAR em algumas ESTRATÉGIAS:

1a- Pode-se ter uma plataforma que tenha um RELÓGIO DE TEMPO REAL (RTC) de precisão que nos permita LER O TEMPO que esse RELÓGIO ESTÁ MARCANDO antes do CÓDIGO COMEÇAR A FUNCIONAR e novamente LER O TEMPO NO FINAL e FAZER A DIFERENÇA entre os DOIS TEMPOS nos dando, assim, o TEMPO DA COMPUTAÇÃO desejado.

2a- Algumas plataformas embarcadas NÃO DISPÕEM de RTC's mas DISPÕEM de “TIMERS” e pode-se utilizá-los para MEDIR o TEMPO DE COMPUTAÇÃO desejado.

3a- Outro RECURSO disponível para a MEDIÇÃO DO TEMPO DE COMPUTAÇÃO é utilizando OSCILOSCÓPIOS ou ANALISADORES LÓGICOS. Esses equipamentos VÃO PRECISAR que ativemos algum pino em específico (tipicamente as PORTAS de E/S dos micros podem ser utilizadas pra isso) e com esses equipamentos (como o OSCILOSCÓPIO por exemplo) vai-se MONITORAR o ESTADO desse pino (ele pode ser por exemplo colocado em nível lógico 1 no início de determinado código e em nível lógico 0 ao final e com o OSCILOSCÓPIO consegue-se CAPTURAR O TEMPO desse PULSO sendo esse tempo a DURAÇÃO do código em questão).

- Métricas/Medições de INTERESSE:



## Métricas de interesse

- Caso médio – comportamento típico
  - Desafio: dados de entrada típicos
- Pior caso (ex. WCET – importante para aplicações de tempo-real)
- Melhor caso

- Dependendo de COMO O DADO CHEGA pra determinado algoritmo ele pode ter um COMPORTAMENTO DIFERENTE em termos de TEMPO DE EXECUÇÃO.

\*Obs. (dependendo do algoritmo ele nem sempre vai ter o mesmo tempo de execução a cada vez que ele é executado! Por exemplo: Algoritmos de ordenação mais simples, os seus tempos de computação vão depender se a estrutura a ser organizada estava previamente em ordem ou parcialmente em ordem).

- Quando falamos de “MEDIR TEMPO DE COMPUTAÇÃO” de um código existem ALGUMAS MÉTRICAS/MEDIÇÕES DE INTERESSE ([a imagem acima mostra um resumo dessas métricas](#)):

- CASO MÉDIO:

- É o “COMPORTAMENTO TÍPICO”.

- Para fazer essa MEDIDA ter-se-ia que conhecer QUAL é o TIPO DE SITUAÇÃO que acontece com MAIS FREQUÊNCIA para aquele algoritmo e aí se MEDIRIA O TEMPO DE COMPUTAÇÃO nesse caso.

- PIOR CASO:

- SE CONSIDERARIA as PIORES CONDIÇÕES.

- Tipicamente APLICAÇÕES DE TEMPO REAL estão MUITO INTERESSADAS em MEDIR ESSE TEMPO pois seus SISTEMAS precisam ser ROBUSTOS O SUFICIENTE para que mesmo que as condições de entradas sejam as piores a computação AINDA ACONTEÇA DENTRO DE UM CERTO TEMPO que seria o “LIMITE” para AQUELA COMPUTAÇÃO.

- (Exemplo-algoritmos de ordenação) Para alguns deles se a ENTRADA vem COMPLETAMENTE INVERTIDA EM ORDEM esse é o PIOR CASO do TEMPO DE COMPUTAÇÃO então pra medir esse tempo se usaria essa ESTRUTURA COMO ENTRADA.

- MELHOR CASO:

- A situação MAIS FAVORÁVEL ao ALGORITMO.

- Sabendo qual situação é a mais favorável ao algoritmo basta simulá-la sob a condição certa e MEDIR O TEMPO DE COMPUTAÇÃO.

- Quando o ALGORITMO cai numa situação dos CASOS acima é RECOMENDÁVEL se fazer as 3 MEDIÇÕES (PIOR, MELHOR e CASO MÉDIO) e apresente como resultado do tempo de

computação do código em análise. Dependendo da FORMA de como essa MÉTRICA DE TEMPO vai ser utilizada a pessoa que recebe essa informação pode escolher qual a métrica de tempo que mais lhe interessa.

- **ALGORITMOS DETERMINÍSTICOS**: Quando o ALGORITMO é COMPLETAMENTE DETERMINÍSTICO, ou seja, é um algoritmo no qual não se importando as condições externas aquele trecho do seu código vai durar sempre o mesmo tempo, os 3 tempos de cada uma da MÉTRICAS ACIMA (PIOR, MELHOR e CASO MÉDIO) vão ser coincidentes não precisando se preocupar com a medição de CADA UMA).

- Estimando o tempo de execução:

**Estimando o tempo de execução**

Porque estimativa:

- Tempo de computação depende dos dados de entrada
- Cache – seu comportamento é (em parte) dependente dos dados de entrada
- Variações no nível de instruções (pipeline e superescalar – dependência de dados)

- Quando falamos de MEDIR O TEMPO DE EXECUÇÃO falamos da possibilidade de se usar um simulador pra isso e da possibilidade de se pegar um hardware que contenha nosso processador e fazer o código funcionar ali para medir o tempo.

- Existem VÁRIAS SITUAÇÕES que podem contribuir para ALTERAÇÕES no TEMPO DE COMPUTAÇÃO/EXECUÇÃO do ALGORITMO

- Existem outras estratégias chamadas de “ESTIMADORES DO TEMPO DE EXECUÇÃO” que também conseguem, como o nome diz, estimar o tempo de computação do algoritmo utilizando para isso de VÁRIOS ARTIFÍCIOS. O nome “ESTIMADOR” dá-se pelas seguintes situações que geram imprevisibilidade nas medições dessas estratégias (a imagem acima mostra quais são abordados):

- QUANDO as ENTRADAS VARIAM e PROVOCAM ALTERAÇÕES (falado agora a pouco) temos o caso em que a COMPUTAÇÃO DEPENDE DOS DADOS DE ENTRADA.

- A COMPUTAÇÃO pode SER IMPACTADA pela PRESENÇA DA CACHE. O comportamento da CACHE não é TOTALMENTE DETERMINÍSTICO, logo, há a possibilidade de um DADO ESTAR na CACHE ou NÃO ESTAR nela. Então, isso pode gerar situações onde numa PRIMEIRA EXECUÇÃO o CÓDIGO DEMORE MAIS e numa SEGUNDA EXECUÇÃO o CÓDIGO DEMORE MENOS pois PARTE DOS DADOS que ele precisa JÁ ESTEJAM DISPONÍVEIS na CACHE por exemplo.

- Outro fator que pode IMPACTAR o TEMPO DE COMPUTAÇÃO que TAMBÉM é uma CARACTERÍSTICA DA ARQUITETURA é o TIPO DE PROCESSADOR (se ele é PIPELINE ou se é SUPERESCALAR por exemplo, as CONDIÇÕES DE EXECUÇÃO são VARIÁVEIS e podem fazer o processador conseguir um PIOR ou MELHOR DESEMPENHO a cada vez que o código é executado).

- Os ESTIMADORES tentam PREVER essas informações, vindas das situações mostradas no parágrafo acima, e nos dar um NÚMERO MÉDIO ESTIMADO junto a uma FAIXA DE PROBABILIDADES.

- Mas, podemos também **NÓS MESMOS** fazer **ESSAS MEDIÇÕES** que os **ESTIMADORES** fazem de uma forma até mais “**ESTATISTICAMENTE CORRETA**”, se conhecermos o processador e soubermos que ele tem esse tipo de característica (Ser PIPELINE ou SUPERESCALAR no caso) devemos fazer VÁRIAS MEDIÇÕES do TEMPO DE COMPUTAÇÃO procurando criar situações onde esses fatores possam influenciar na medição feita nos dando ao invés de apenas um número (que seria o TEMPO DE COMPUTAÇÃO) um CONJUNTO de VALORES e DADOS como sua MÉDIA e seu DESVIO PADRÃO por exemplo.

- MÉTODO SIMPLISTA:

## Tempo das instruções (1)



- Método simplista
  - Todas instruções usam o mesmo número de ciclos de clock
  - Contar a quantidade de instruções e multiplicar pelo custo por instrução

Verdade para processadores RISC simples

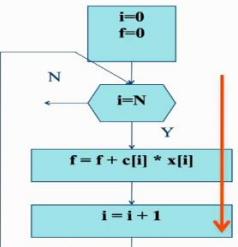
- É o MÉTODO MAIS SIMPLES de se fazer essas MEDIÇÕES DE TEMPO DE COMPUTAÇÃO utilizando SIMULADORES/ESTIMADORES.
- Esse método se baseia somente no CONHECIMENTO DO TEMPO DE EXECUÇÃO de cada instrução e a ferramenta pode analisar o FLUXO DE EXECUÇÃO DO CÓDIGO, contar QUANTAS INSTRUÇÕES seriam EXECUTADAS naquele fluxo e a partir daí fazer uma MEDIDA SIMPLES entre o TEMPO DE COMPUTAÇÃO de CADA INSTRUÇÃO e a QUANTIDADE DE INSTRUÇÕES que foi realizada.
- Esse método ADMITE QUE TODAS AS INSTRUÇÕES tem o MESMO TEMPO DE COMPUTAÇÃO que é o que acontece com a maioria dos processadores hoje em dia porque são de ARQUITETURA RISC.
- Na imagem abaixo um exemplo de fluxo de código e como se faz isso:

### Exemplo de fluxo de código (1)

```
for (i=0, f=0; i<N; i++)
  f = f + c[i] * x[i];
```

Custos a considerar:
 

- Preparação do laço
- Corpo do laço

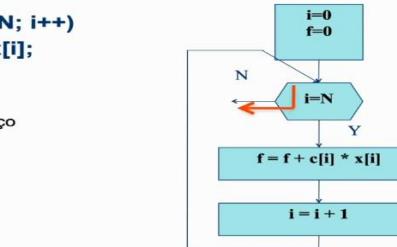


### Exemplo de fluxo de código (1)

```
for (i=0, f=0; i<N; i++)
  f = f + c[i] * x[i];
```

Custos a considerar:
 

- Preparação do laço
- Corpo do laço
- Teste



- Vemos a DIREITA um FLUXOGRAMA de um código.

- Nesse FLUXOGRAMA há um LAÇO DE REPETIÇÃO e estão definidas quais são as instruções que serão executadas em cada FASE desse pequeno algoritmo. Essas fases tem TEMPOS CARACTERÍSTICOS e quando essa simulação é feita conta-se a FASE DE PREPARAÇÃO de

INÍCIO DO LAÇO, a FASE REPETITIVA (que é o CORPO DO LAÇO) e a VARIÁVEL “N” precisa ser conhecida para se fazer esse tipo de análise.

- Existe uma PEQUENA DIFERENÇA na ÚLTIMA ITERAÇÃO COM O LAÇO quando o teste é feito e o resultado dá positivo é uma forma de introduzir um pouco mais de precisão nessa conta.

- **REDUZIR o TEMPO DE COMPUTAÇÃO** de um **PROJETO**:

The screenshot shows a video player interface. At the top, there's a green header bar with the title 'Como reduzir o tempo de computação?'. Below the title, there's a video thumbnail of a man with glasses and a yellow shirt. The main content area contains a bulleted list:

- No software
  - Substituindo o algoritmo
  - Otimizando os acessos à memória
- No hardware
  - Implementações de algoritmo em hardware (ASIC, ASSP)
  - Uso de instruções especializadas (ASIP)

At the bottom of the slide, there's a progress bar indicating the video is at 11:04 / 13:31. The video player has standard controls like play/pause, volume, and a full-screen button.

- Como podemos trabalhar nosso projeto para diminuir o tempo de computação se isso for necessário?

- **(Trabalhar) No SOFTWARE**:

**\*\*\*Obs.** (A MEMÓRIA é um PONTO CHAVE na execução de um CÓDIGO em QUALQUER PROCESSADOR é o ACESSO A MEMÓRIA pois é aí onde se costuma sentir um certo “GARGALO DE TEMPO DE COMPUTAÇÃO”).

- Podemos buscar um ALGORITMO DIFERENCIADO, dependendo de como é a aplicação sob análise podem existir VÁRIOS ALGORITMOS que IMPLEMENTAM aquele mesmo tipo de solução e naturalmente alguns serão MAIS RÁPIDOS que OUTROS, existem ainda METODOLOGIAS DE OTIMIZAÇÃO para que tais ALGORITMOS POSSAM SE TORNAR MAIS RÁPIDOS.

- Olhar a MANEIRA como o ALGORITMO ACESSA A MEMÓRIA e BUSCAR MECANISMOS que explorem propriedades da sua arquitetura e TORNEM o ACESSO A MEMÓRIA MAIS RÁPIDO na execução do código.

- **(Trabalhar) No HARDWARE**:

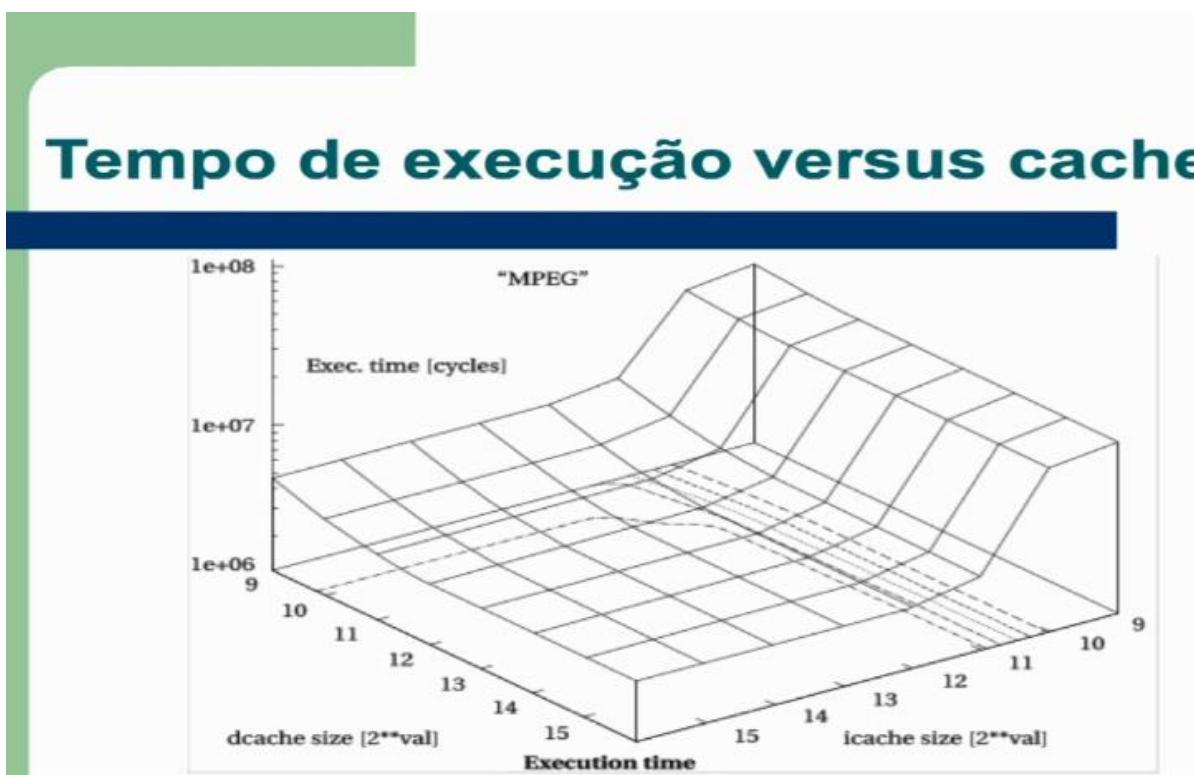
**\*\*\*Obs.** (Já vimos que algoritmos podem ser implementados em hardware então essa também é uma estratégia que podemos usar)

- Deslocar uma PARTE DA COMPUTAÇÃO para o HARDWARE ESPECIALIZADO seja ele um ASIC ou ASSP. Ou ainda, MOVER TODA A APLICAÇÃO para ESSE HARDWARE ESPECIALIZADO se isso for POSSÍVEL ou NECESSÁRIO.

- Utilizar um PROCESSADOR que TENHA INSTRUÇÕES ESPECIALIZADAS para determinado GRUPO DE APLICAÇÃO, que no caso seria um ASIP (Veja que essa estratégia não envolve um HARDWARE ESPECIALIZADO mais é “quase isso”):

\*\*\*([Exemplo](#)) Se tivéssemos uma aplicação especializada em PROCESSAMENTO DE SINAIS por exemplo, ela, claramente, seria uma candidata a utilizar um DSP que é o TIPO DE ASIC MAIS CONHECIDO.

- TEMPO DE EXECUÇÃO versus CACHE:



- A figura acima é uma ANÁLISE de DESEMPENHO DE CACHE em um ALGORITMO ESPECÍFICO (no caso é o MPEG), onde se foi feito um estudo sob os TEMPOS DE COMPUTAÇÃO para DIFERENTES TAMANHOS DE CACHE.

- Note que a “icache” (“CACHE DE INSTRUÇÕES”) de menor tamanho ( $2^{**\text{VAL}} = 9$ ) tem o TEMPO DE COMPUTAÇÃO que é o MAIS ALTO POSSÍVEL. Quando passamos pra uma cache de tamanho um pouco maior ( $2^{**\text{VAL}} = 10$ ) já tem-se um certo ganho mas claramente quando se passa para uma cache de tamanho  $2^{**\text{VAL}} = 11$  é que se tem um GANHO EXPRESSIVO no TEMPO DE EXECUÇÃO (Isso já é SABIDO, CACHES MAIORES permitem DIMINUIR O TEMPO DE COMPUTAÇÃO DE UM ALGORITMO).

- Na “dcache” (“CACHE DE DADOS”) temos a mesma coisa, quando vamos aumentando os valores de “ $2^{**\text{VAL}}$ ” vamos tendo TEMPOS DE COMPUTAÇÃO menores até que se chega num ponto em que continuar AUMENTANDO O TAMANHO DA CACHE torna-se irrelevante pois a diminuição de TAL ELEMENTO se torna quase que constante e mínima (o mesmo é visto na “icache” quando se atinge  $2^{**\text{VAL}} = 12$ , desse valor pra frente o gráfico é quase uma reta).

- Na linha de que “SEMB’s gastam o MÍNIMO DE RECURSOS POSSÍVEIS” para o exemplo acima o ideal seria ficar com uma dcache e um ichace de tamanhos  $2^{**12}$  (pois só aí tem-se o mínimo tempo de execução pro MPEG).

/////////// [Parte: 02](#)

- ENERGIA & POTÊNCIA (em APLICAÇÕES EMBARCADAS):

- LEMBRE que ENERGIA e POTÊNCIA NÃO SÃO A MESMA COISA!
- Dependendo do interesse da aplicação iremos ou precisar DIMINUIR A ENERGIA ou DIMINUIR A POTÊNCIA ou até mesmo DIMINUIR AS DUAS!

## Energia e potência



- **Energia:** habilidade para terminar o serviço.
  - Mais importante em dispositivos alimentados por baterias
- **Potência:** suprimento solicitado da fonte.
  - Importante mesmo em dispositivos ligados na tomada – potência vira calor

- **ENERGIA:** Quando falamos de ENERGIA estamos preocupados na HABILIDADE que um SISTEMA vai ter de TERMINAR UM SERVIÇO ([um exemplo](#) seria a habilidade de um sistema de manter um celular funcionando por 24 hora):

- (“HABILIDADE que um SISTEMA vai ter de TERMINAR UM SERVIÇO”) << Essa é uma CARACTERÍSTICA IMPORTANTE para SISTEMAS ALIMENTADOS POR BATERIA.
- [Mais um exemplo](#) disso seria um DRONE que tem de decolar, ir até um local, bater certas fotos, decolar novamente e voltar, a ENERGIA É FUNDAMENTAL para garantir o cumprimento dessa missão!

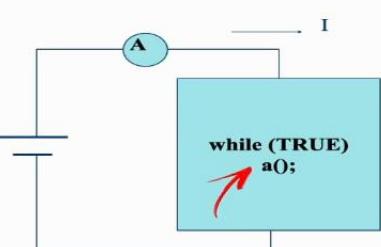
- **POTÊNCIA:** Está associada ao SUPRIMENTO que o equipamento no computador vai solicitar da fonte:

- A fonte estará ali para FORNECER A POTÊNCIA que o SISTEMA PRECISAR.
- Tanto os dispositivos LIGADOS NA TOMADA quanto os MOVIDOS A BATERIA precisam estar atentos a POTÊNCIA que a FONTE é capaz de OFERECER.
- A CAPACIDADE DE CORRENTE que a FONTE PODE FORNECER sempre será LIMITADA. O COMPUTADOR precisa estar atento para RESPEITAR ESSES LIMITES ou, de uma outra forma, estabelecer com clareza QUAIS SÃO esses LIMITES para que a FONTE que vai alimentar o computador esteja apta a mantê-lo funcionando sem que OCORRA NENHUM ACIDENTE.
- Tipicamente a POTÊNCIA está associada a CALOR. Um sistema que demanda uma POTÊNCIA MAIOR da FONTE provavelmente vai produzir MAIS CALOR seja NELE MESMO (a própria CPU esquentar enquanto trabalha) ou na própria FONTE (produzirá calor no momento de funcionamento).

- [Medindo o CONSUMO DE ENERGIA \(em APLICAÇÕES EMBARCADAS\):](#)

## Medindo o consumo de energia

Execute um pequeno laço e meça a corrente...




- Para fazer isso é necessário chegar mais perto da área “eletrônica” já que iremos medir CORRENTES, VOLTAGENS e outros elementos.

- A imagem acima mostra um exemplo bem simples de uma medição:

- A esquerda mostra-se uma BATERIA.
- Em cima um amperímetro.
- A direita tem-se um bloco que representa o COMPUTADOR com um programa executando um “while (TRUE)”.
- O “a();” seria o método que o SEMB está executando, é nessa situação que quer-se medir a POTÊNCIA/ENERGIA CONSUMIDA PELO SISTEMA.

- Por fim, para se **fazer** essas **MEDIÇÕES** existem **ALGUMAS METODOLOGIAS**:

- **Medir a POTÊNCIA COM RESISTOR “SHUNT”** (**método indireto de medição**):



- É um **MÉTODO DE MEDAÇÃO indireto** (não se mede o valor desejado DIRETAMENTE, mas sim usando-se de outras medições feitas no sistema para se chegar a esse valor).

- Utiliza um **RESISTOR EM SÉRIE** com o **SISTEMA** chamado de “**RESISTOR SHUNT**” (**Rshunt**).

- O **Rshunt** é tipicamente um **RESISTOR de VALOR MUITO BAIXO**.

- Se o **Rshunt** tiver uma **RESISTÊNCIA de VALOR GRANDE** a queda de tensão nele pode ser suficiente pra fazer o **COMPUTADOR PARAR DE FUNCIONAR** \*\*\*(**O que seria uma desvantagem desse sistema**).

- Nossa computadora é representada pelo “**load**” (essa carga) na figura acima.

- A **FONTE** que **ALIMENTA** o **SISTEMA** é representada acima na esquerda da figura como “**source**”.

- Esse método, por **INTERROMPER** o **SISTEMA DE ALIMENTAÇÃO** para colocar um **RESISTOR EM SÉRIE**, pode ser chamado de **INVASIVO** \*\*\*(**O que seria uma desvantagem desse sistema**).

- (**Explicação- Rshunt**): Imagine a figura acima como um computador que funciona com 5V e a FONTE é quem os está fornecendo para ele. Quando a corrente de alimentação passar pelo **Rshunt** ele terá uma certa queda de tensão, essa queda não pode ser o suficiente para parar o sistema computacional, se o sistema é alimentado

com 5V existe uma tensão mínima de operação, talvez 4,5V, então esse Rshunt tem que ser dimensionado de maneira que a tensão nele não passe de 0,5V sob pena de o computador acabar sendo desligado ou ele apresentar mal funcionamento o que nos faria não fazer mais as medições de nossa vontade.

- (Explicação- Medição da potência): Há um OSCILOSCÓPIO no lado direito da figura acima (que poderia ser um VOLTÍMETRO também) para medir a TENSÃO no Rshunt e na CARGA “source”, tendo-se essas duas tensões pode-se CALCULAR A POTÊNCIA porque a TENSÃO no Rshunt me permite calcular sua corrente (i Rshunt), pois sabe-se o valor de sua resistência, e com essa corrente multiplica-se com a TENSÃO no computador dando o VALOR DA POTÊNCIA que queremos medir.

- Medir a POTÊNCIA COM PONTA DE CORRENTE (método indireto de medição):



- Não utiliza Rshunt.

- É um MÉTODO DE MEDAÇÃO indireto (não se mede o valor desejado DIRETAMENTE, mas sim usando-se de outras medições feitas no sistema para se chegar a esse valor).

- Utiliza uma “PONTA DE PROVA” de OSCILOSCÓPIO que também pode ser utilizada em VOLTÍMETROS que é para MEDIR A CORRENTE num CERTO FIO utilizando do EFEITO MAGNÉTICO dessa CORRENTE.

- (Explicação- Ponta de prova): Quando a corrente passa pelo fio ela induz um campo magnético em torno desse fio, essa PONTA DE PROVA capta através de um dispositivo seu esse campo magnético em torno do fio e converte isso na CORRENTE EQUIVALENTE, então, essa medição vai nos dar o VALOR INDIRETO DA CORRENTE (igual como no método de medição anterior, com a diferença de que nesse outro método tinha-se a CORRENTE INDIRETA fazendo uma relação com a TENSÃO enquanto nesse método tem-se a CORRENTE INDIRETA fazendo relação com o CAMPO MAGNÉTICO).

- Esse método, por NÃO INTRODUZIR NADA no MEIO DO CIRCUITO, pode ser chamado de NÃO-INVASIVO \*\*\*(O que seria uma vantagem desse sistema).

- Esse método, por NÃO INTRODUZIR NADA no MEIO DO CIRCUITO, não corre risco de que elementos, como um resistor por exemplo, façam o sistema parar de

funcionar por defeitos como o súbito abaixamento de tensão. \*\*\*(O que seria outra vantagem desse sistema).

- Esse é um MÉTODO MAIS CARO pelo fato de que essa “PONTA DE PROVA” utilizada não se um instrumento BARATO.

- Apesar de ser MAIS CARO, esse método entrega uma medição melhor!

- Falou-se de dois métodos de medição INDIRETOS (um pela TENSÃO e outro pelo CAMPO MAGNÉTICO), por que, então, não usar de um AMPERÍMETRO diretamente nos circuitos anteriores e obter uma MEDIÇÃO DIRETA?

- Resposta: Pois o problema do AMPERÍMETRO se parece com o problema do Rshunt (Na verdade todos os AMPERÍMETROS são “Rshunt’s” colocados em série com o circuito onde se faz a conversão da TENSÃO nesse resistor para CORRENTE). Os AMPERÍMETROS NORMAIS têm Rshunt’s MUITO ALTOS e esses valores de resistência alta acabam por comprometer quaisquer medições que sejam feitas no sistema.

- Conclusão: É melhor que nós mesmos ou dimensionemos esses Rshunt’s para evitar que o COMPUTADOR PARE por TENSÃO BAIXA por causa do uso mal configurado desses ou que usemos de uma PONTA DE PROVA DE CORRENTE se tivermos acesso a isso.

- Fontes de consumo de energia – EM SISTEMAS COMPUTACIONAIS (guia geral):

**Fontes de consumo de energia**

Energia relativa por operações:

- Transferências de memória: 33
- Operações de I/O: 10
- Escrita na SRAM (cache): 9
- Leitura na SRAM (cache): 4,4
- Multiplicação: 3,6
- Soma: 1

7:05 / 13:12

- A imagem mostra que tipos de coisa costumam gastar mais POTÊNCIA ou ENERGIA num SISTEMA COMPUTACIONAL.

- A SOMA é a operação que gasta MENOS ENERGIA pois é uma das MAIS SIMPLES que se tem. É feita pela ULA dentro do processador envolvendo processadores. Na lista da imagem acima é usada como “REFERÊNCIA” para as outras operações.

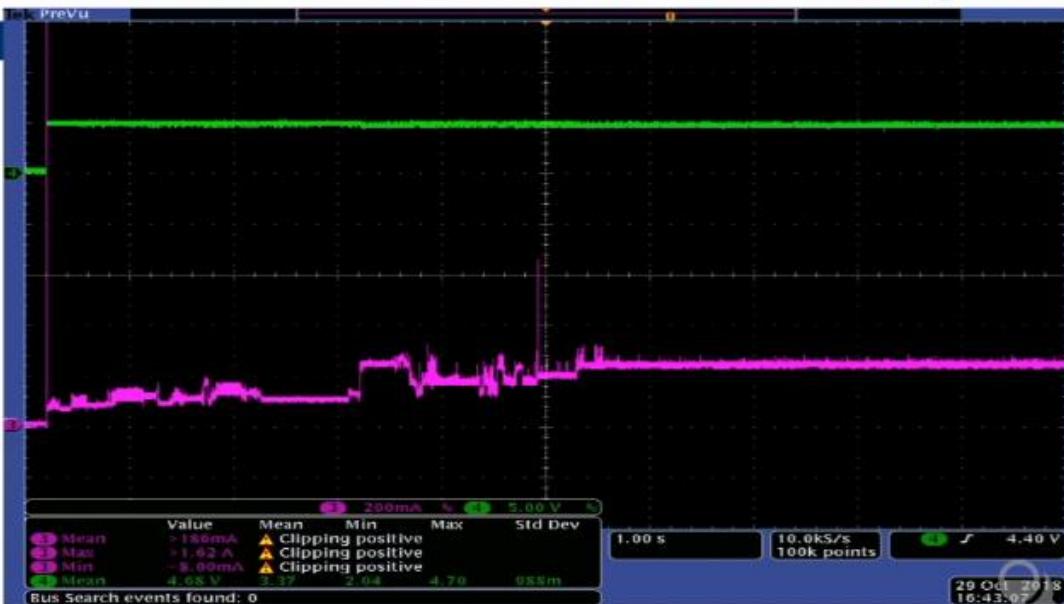
- Multiplicação ainda é considerada “SIMPLES” mas por executar mais etapas que a SOMA passa a GASTAR MAIS ENERGIA que ela.

- Acessar a MEMÓRIA CACHE (seja por LEITURA ou ESCRITA) ainda é MAIS BARATO do que ACESSAR A MEMÓRIA PRINCIPAL (que na lista seria a operação de “Transferências de memória”).

- O NÚMERO do GASTO ENERGÉTICO das OPERAÇÕES de I/O é “APROXIMADO” porque é um valor que DEPENDE MUITO de qual que é o DISPOSITIVO DE E/S que se está referenciando e esse consumo apresentado na imagem acima é o CONSUMO que diz RESPEITO AO PROCESSADOR já que lá o próprio DISPOSITIVO DE E/S tem seu CONSUMO TÍPICO (e precisa ser colocado nessa conta).

- Medidas de corrente na \*Raspberry (interpretação gráfica):

# Medidas de Corrente na Raspberry



- A imagem acima é uma CAPTURA DE TELA feita pelo professor da imagem do OSCILOSCOPIO mostrando a EVOLUÇÃO DA CORRENTE de uma PLACA RASPBERRY do momento em que ela é LIGADA ao momento em que o SISTEMA DESSA PLACA entra em OPERAÇÃO (que é quando podemos utilizar a RASPBERRY através de sua linha de comando).
- A linha verde é o momento em que a TENSÃO. Ela começa em ZERO e quando a PLACA É ACIONADA essa TENSÃO AUTOMATICAMENTE tem seu VALOR ATINGIDO (O gráfico da TENSÃO é uma LINHA porque a TENSÃO DA FONTE de um computador fica SEMPRE CONSTANTE sempre SUPRINDO O SISTEMA).
- O GRÁFICO ROSA mostra a CORRENTE que a RASPBERRY está CONSUMINDO A CADA INSTANTE. Note que no começo essa corrente está longe de se estar CONSTANTE dando SALTOS e SALTOS, isso acontece dependendo de quais dos dispositivos estão sendo inicializados, se estão utilizando memória do cartão, se estão utilizando a RAM e, assim, a corrente vai tendo seus valores alterados a medida do tempo em que é medida até que fique “CONSTANTE/ESTABILIZE” que é quando já se tem o PROCESSADOR BUSCANDO INSTRUÇÕES DA MEMÓRIA RAM que é um TIPO DE OPERAÇÃO que tem basicamente o MESMO CONSUMO (mesmo que ainda hajam pequenas variações na corrente elas são BEM MENOS perceptíveis para nós).

- [Formas de redução de POTÊNCIA/ENERGIA em SISTEMAS COMPUTACIONAIS:](#)

# Como reduzir a potência/energia?

- No software
  - Substituindo o algoritmo
  - Otimizando os acessos à memória
- No hardware
  - Desligando componentes ociosos
  - Chips *Low-power*
  - ...

$$P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$$



11:49 / 13:12

- Como podemos trabalhar nosso projeto para diminuir a POTÊNCIA/ENERGIA se isso for necessário?

- ([Trabalhar](#)) No SOFTWARE:

- SUBSTITUINDO O ALGORITMO nós podemos DIMINUIR O TEMPO DE COMPUTAÇÃO (como já discutido antes) e DIMINUIDO esse tempo temos a possibilidade de DIMINUIR A ENERGIA TOTAL QUE O SISTEMA CONSOME (como já foi falado antes, diminuir o tempo de computação significa a possibilidade de aumentar o tempo em que a máquina ficará desligada).
- MELHORAR O ACESSO A MEMÓRIA (que significa diminuir os casos de acesso a memória) é uma forma de se DIMINUIR a ENERGIA GASTA do SISTEMA COMPUTACIONAL (porque, como já vimos antes, as memórias são responsáveis por uma parte importante dos gastos de potência dos sistemas computacionais).

- ([Trabalhar](#)) No HARDWARE:

- Pode-se DESLIGAR COMPONENTES OCIOSOS (a maioria dos SoC's oferecem mecanismos para DESLIGAR DISPOSITIVOS DE E/S, por exemplo, que NÃO estão SENDO UTILIZADOS pela APLICAÇÃO em DETERMINADOS MOMENTOS), ao fazer isso vamos DIMINUIR A POTÊNCIA, pois determinado dispositivo vai deixar de consumir energia por estar “DESLIGADO”, e a ENERGIA (que diminuirá ao longo do tempo).
- ([Uso de “Chips Low-power”](#)) Somente por se TROCAR um CHIP NORMAL pela sua versão “LOW-POWER” já temos um GANHO DE POTÊNCIA muitas vezes suficiente para resolver os problemas de determinado projeto:

\*\*\*([Obs.](#))\_O termo “[Low-power](#)” é utilizado pelos fabricantes para indicar que eles fabricaram o semi-condutor utilizando mecanismos para que os transistores daquele chip GASTEM MENOS POTÊNCIA. Por isso, é COMUM que os FBRICANTES OFEREÇAM as versões [NORMAL](#) e “[Low-power](#)” de determinado PROCESSADOR.

-----> [A VERSÃO LOW-POWER tipicamente tem POTÊNCIAS MAIS BAIIXAS do que a VERSÃO NORMAL do processador.](#)

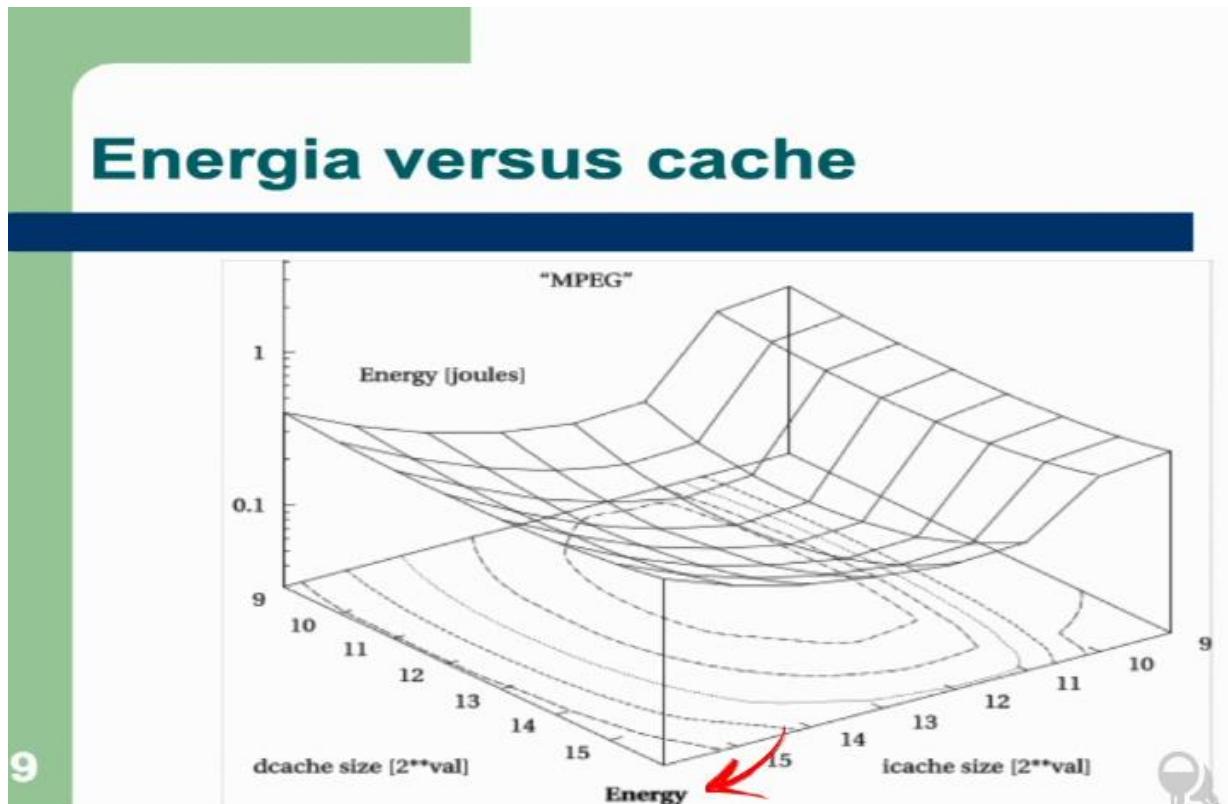
- Mesmo com todos os MECANISMOS DE TRABALHO em SOFTWARE/HARDWARE que falamos para se procurar MELHORAR A POTÊNCIA/ENERGIA de um SISTEMA COMPUTACIONAL devemos ter sempre em mente a seguinte fórmula já abordada:

$$P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$$

- Então, na hora de se pensar em OTIMIZAR POTÊNCIA/ENERGIA deve-se estar olhando para essa fórmula para se buscar na plataforma de desenvolvimento em uso onde for melhor de mexer (em tensão, em frequência ou nas duas coisas) para MELHORAR/ADEQUAR a POTÊNCIA ao requerido ótimo.

- Lembre, também, que a ENERGIA é o produto da POTÊNCIA PELO TEMPO quando for mexer nessas duas grandezas (ENERGIA e POTÊNCIA).

- **ENERGIA versus CACHE:**



- A figura acima mostra COMO SE COMPORTA o GASTO ENERGÉTICO DO ALGORITMO “MPEG” em CACHES DE DIFERENTES TAMANHOS.

- No caso da ENERGIA É UM POUCO MAIS CRÍTICO pois existe um “PONTO ÓTIMO”, ou seja, a medida que vamos aumentando o TAMANHO DA CACHE e a ENERGIA GASTA vai DIMINUINDO tendemos a chegar em um ponto onde se tem o MELHOR NÍVEL DE ENERGIA. Caso continue-se a AUMENTAR A CACHE já estando nesse PONTO ÓTIMO PROBLEMAS (como a tendência da energia de voltar a aumentar seu gasto no sistema novamente, o que seria uma PERDA ENERGÉTICA DESNECESSÁRIA) PODEM VIR A SURGIR NO SISTEMA E EM SUA ADMINISTRAÇÃO ENERGÉTICA.

////////// [Parte: 03](#)

- **MEMÓRIA - TAMANHO DO CÓDIGO (em APLICAÇÕES EMBARCADAS):**

- Aqui vamos falar das MEMÓRIAS num código e quando falamos de “MEMÓRIA” estamos interessados em AVALIAR o TAMANHO DO CÓDIGO (tanto do TAMANHO DO PROGRAMA EXECUTÁVEL quanto da MEMÓRIA DE DADOS QUE O CÓDIGO VAI PRECISAR UTILIZAR).

- MEMÓRIAS DISSIPAM POTÊNCIA bem como o ACESSO A ELAS também!

- TAMANHO DO CÓDIGO:

## Tamanho do código

- **Metas:**
  - Reduzir preço do hardware (memória)
  - Reduzir dissipação de potência nas memórias
- **Duas possibilidades:**
  - Dados
  - Instruções

- OBJETIVOS ao se REDUZIR O TAMANHO DO CÓDIGO (METAS):

- Redução no PREÇO (Quanto MAIS MEMÓRIA o SEMB utilizar MAIS CARO ele será!).

- DISSIPAR MENOS POTÊNCIA NO FUNCIONAMENTO da aplicação (MEMÓRIAS DISSIPAM POTÊNCIA e o ACESSO A ELAS também, portanto, reduzir suas dissipações de potência no sistema é um objetivo importante).

- Portanto, é preciso DIMINUIR O GASTO DE MEMÓRIA não só para que determinado SoC se torne MAIS BARATO como, também, para que ele precise de MENOS POTÊNCIA PARA FUNCIONAR (tem-se um BENEFÍCIO DUPLO!).

- O que tem-se para MEDIR e OTIMIZAR quando se fala de MEMÓRIAS:

- Dados (Estão armazenados em memória).

- Instruções (Também encontram-se armazenadas em memórias).

- Quais TECNOLOGIAS DE MEMÓRIA são utilizadas nos SEMB's (podem variar dependendo da plataforma):

## Tecnologias de memória

- **Microcontroladores (SoC):**
  - Flash/ROM (Código, constantes ...)
  - SRAM (variáveis globais, pilha ...)
- **Linux (Processadores):**
  - RAM

- Se estamos falando de **MICROCONTROLADORES** (podemos chamar de SoC também) as **memórias são organizadas** em:

- **FLASH ou ROM** (é onde estará o **código da aplicação** e as **constantes** da mesma, que são os valores que não precisam ser alterados em tempo de execução, ...).

- **SRAM** (é onde estão as **variáveis globais, pilhas**, ...).

- \*(**Obs.**) Nos **SoC's** tipicamente se **utiliza a memória "SRAM"** pois as **MEMÓRIAS ESTÁTICAS** são mais fáceis de trabalhar e não PRECISAM DE "**REFRESH**". **SoC's** se comparados aos **PROCESSADORES** são **PLATAFORMAS COM MENOS RECURSOS** computacionais.

- Se estamos falando de **PROCESSADORES** as memórias são organizadas em (por processadores nos referimos as **PLATAFORMAS COM MAIS RECURSOS** onde tipicamente são **utilizados SO's sobre elas**):

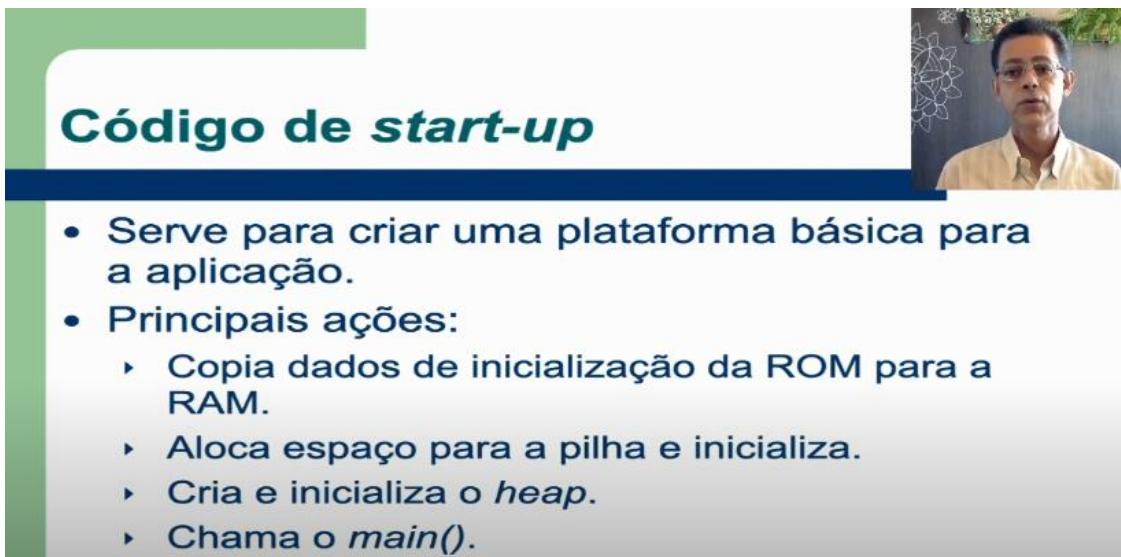
- **RAM** (vai ser basicamente **TODA A MEMÓRIA DO SISTEMA**):

- Tanto o **CÓDIGO DA APLICAÇÃO** vai estar nessa memória quanto os **DADOS** dele.

- Também, são **memórias do tipo "DINÂMICAS"**.

- \*(**Obs.**) Nos **PROCESSADORES** É muito comum que as **MEMÓRIAS** sejam **DO TIPO "DINÂMICA"**.

- **CÓDIGO DE "START-UP"**:



## Código de start-up

- Serve para criar uma plataforma básica para a aplicação.
- Principais ações:
  - Copia dados de inicialização da ROM para a RAM.
  - Aloca espaço para a pilha e inicializa.
  - Cria e inicializa o *heap*.
  - Chama o *main()*.

- Quando o **compilador GERA** o código pra ser executado na **PLATAFORMA ALVO** outro código é incluído chamado de "**START-UP**" para **PREPARAR O AMBIENTE DA MÁQUINA** para que **ela POSSA EXECUTAR A APLICAÇÃO**.

- **PRINCIPAIS AÇÕES** do "**START-UP CODE**":

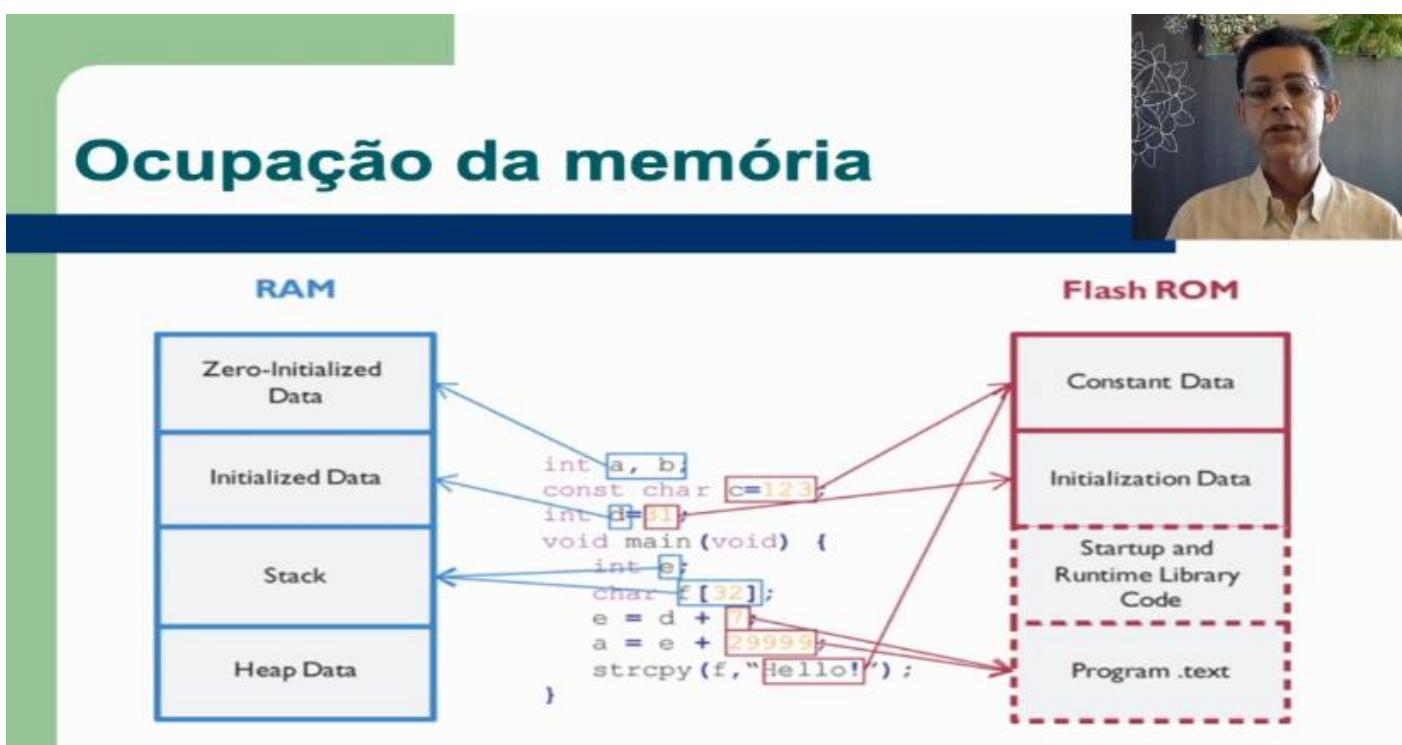
- **Copia dados de inicialização que vão ter que ir na ROM para a RAM.** \*\*\***Exemplo:** Se uma aplicação tem alguns valores que precisam ser inicializados as variáveis vão estar na RAM, quando a máquina for ligada não vai ter nada gravado lá e é preciso mover esses dados pré-definidos pelo programador no código que estarão na ROM nesse momento para que possam ir na RAM e a partir dali as variáveis sejam devidamente inicializadas.

- **Alocar ESPAÇO PARA A PILHA e INICIALIZÁ-LA.**

- Vai CRIAR e INICIALIZAR o espaço chamado de “HEAP” (que é onde estarão as **VARIÁVEIS GLOBAIS** do sistema).

- Chamar o método MAIN (é ele que vai assumir o controle do sistema depois de que todos os “eventos acima” tiverem sido devidamente executados pelo “START-UP CODE”).

- Ocupação da memória (ANÁLISE ILUSTRATIVA):



Fonte: Embedded Systems Fundamentals with Arm Cortex-M based Microcontrollers, 2017

Ativar o Wii

- A figura acima é um **RESUMO** de como as variáveis e suas diversas formas criadas no código são **CONSTRUÍDAS PELO COMPILADOR** dentro do **SISTEMA**.

- A figura acima toma por base uma **ARQUITETURA DE MICROCONTROLADORES** que é onde tem-se o **CÓDIGO** e as **CONSTANTES** em **MEMÓRIA FLASH** (que é uma **MEMÓRIA NÃO VOLÁTIL**) e as **VARIÁVEIS** (que precisam ser alteradas em **TEMPO DE EXECUÇÃO**) que vão estar construídas na **RAM**.

- As variáveis **DECLARADAS DENTRO DE UM MÉTODO** (que no caso da figura acima são as variáveis “e” e “f” declaradas dentro do método “main(void)”) são **CONSTRUÍDAS** dentro da **PILHA**, que normalmente **NÃO É MUITO GRANDE**.

- Não pode-se declarar **VARIÁVEIS MUITO GRANDES LOCALMENTE DENTRO DOS MÉTODOS**.

- Fora do método “main(void)” nota-se as variáveis “a” e “b”, que são variáveis **NÃO INICIALIZADAS**, e a variável “d” que é pré-inicializada com um certo valor, esse valor vai estar lá na **FLASH** e vai ser **UTILIZADA** pelo “START-UP CODE” pra “setar” esse valor antes de chamar o método “main(void)”.

- Mesmo em computadores **DESKTOP** se forem declaradas **VARIÁVEIS GIGANTES** dentro do método “main(void)” isso vai dar **PROBLEMA DE MEMÓRIA** (até o compilador pode reclamar do muito uso de memória pelo algoritmo).

- Num **MICROCONTROLADOR** a **PILHA** que recebe essas “**VARIÁVEIS DECLARADAS DENTRO DE UM MÉTODO**” é **AINDA MAIS LIMITADA**. Então, mais ainda do que num **DESKTOP**, em **MICROCONTROLADORES** deve-se tomar **MAIOR CUIDADO** para não se **DECLARAR VETORES MUITO GRANDES, ESTRUTURAS DE DADOS MUITO GRANDES** dentro dos métodos.

- Variáveis MAIORES DEVEM SER DECLARADAS nas ÁREAS “COMUNS”, nas ÁREAS DE VARIÁVEIS GLOBAIS isso porque nessa área se consegue EXTENDER PARA TODO O TAMANHO DA MEMÓRIA.

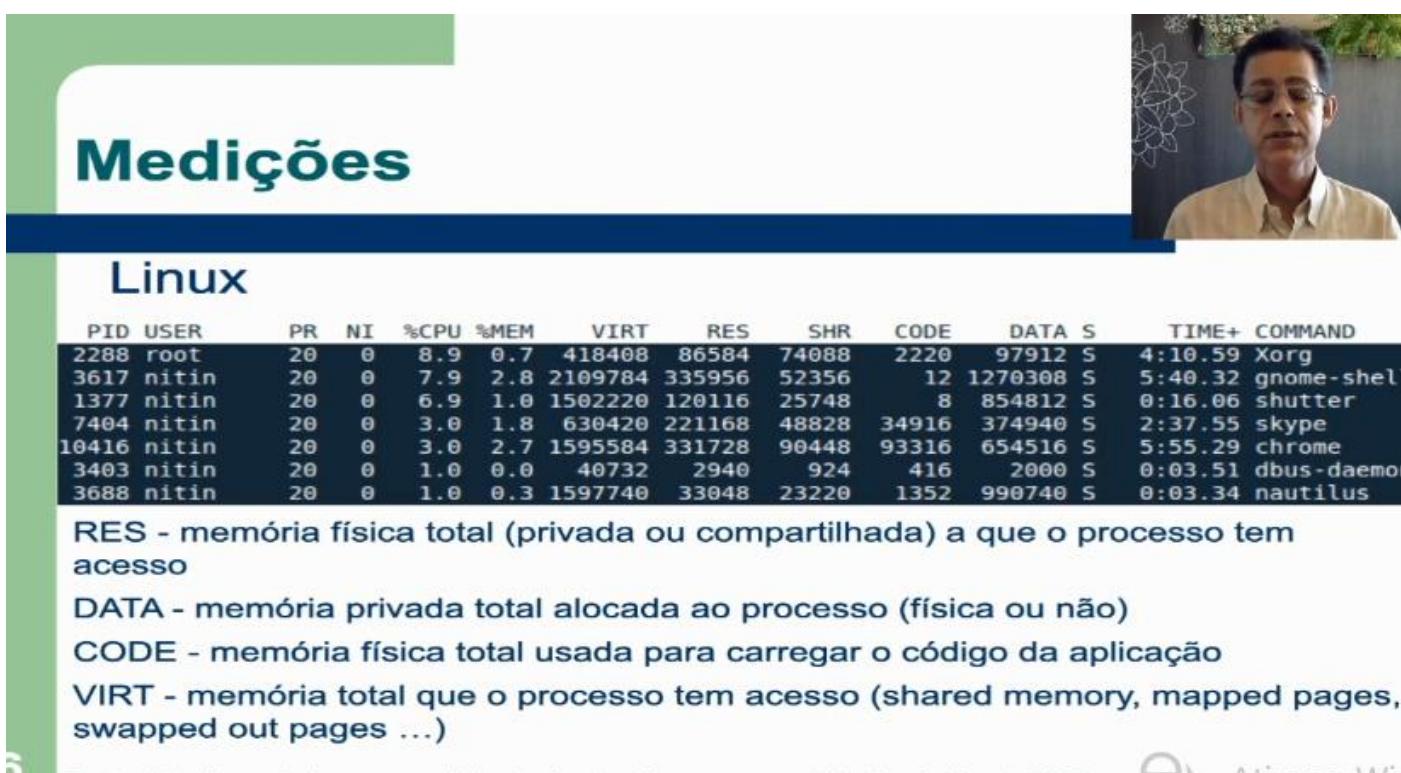
- Então, se um MICROCONTROLADOR tem 64 Kbites de MEMÓRIA RAM, não pode-se ocupar esses 64 Kbites com VARIÁVEIS DECLARADAS DENTRO DE SEUS MÉTODOS, tem-se que usar VARIÁVEIS GLOBAIS para assim poder usar TODA essa MEMÓRIA.

- MEDIÇÃO DE MEMÓRIA UTILIZANDO MPLABX – MICROCONTROLADOR PIC16(**EXEMPLO**):



- A imagem acima vemos o QUANTO DE MEMÓRIA essa aplicação de exemplo está utilizando.
- É mostrado as MEMÓRIAS de DADOS e PROGRAMAS.

- MEDIÇÃO DE MEMÓRIA – SISTEMA DE GRANDE PORTE LINUX(**EXEMPLO**):



Fonte: <http://www.tothenew.com/blog/understanding-memory-utilization-in-linux/>, 2016



Ativar o Wii

- O programa usado nas medições da imagem acima usa de uma SÉRIE DE MÉTRICAS que são relativas ao gasto de memória. O problema com essa estratégia de medição é que ela indica os programas que estão NESSE MOMENTO NA MEMÓRIA DAQUELE SISTEMA e QUANTO ESTÃO UTILIZANDO DELA, ou seja, é como uma “FOTOGRAFIA” desse “momento”. Algumas dessas medidas podem ser interessantes para a avaliação dessas aplicações que estão em execução:

- **RES**: A memória física total disponível para aquela aplicação (já que o SO aloca memória para as aplicações de acordo com os pedidos que as aplicações fazem).

- **DATA**: A memória privada de dados.

- **CODE**: A memória de código.

- **VIRT**: Memória virtual total disponível para a aplicação.

- Quando se trata de APLICAÇÕES EMBARCADAS é comum que mesmo havendo um SO por baixo aquela aplicação acaba por “DOMINAR O USO” da máquina em que está operando, ou seja, ao ligar o SEMB a aplicação começa a funcionar e NÃO PARA MAIS NUNCA (não é o que acontece com o computador desktop típico por exemplo, onde se carrega/descarrega aplicações elas entram e começam a usar memória mas logo depois saem e liberam memória para outras aplicações).

#### **- ESTRATÉGIAS PARA REDUZIR A QUANTIDADE DE MEMÓRIA QUE UMA APLICAÇÃO EMBARCADA PRECISA:**

##### **Reducindo o tamanho dos dados**

- Uso de dados depende do estilo de programação => **Muito espaço para otimização**
- Reuso de constantes, variáveis, buffers de dados em diferentes partes do código
  - Baixo nível
  - Requer verificação cuidadosa da correção (usar com cuidado)  
=> **Variáveis globais geram problema de concorrência!**

(2010) <https://www.embedded.com/five-top-causes-of-nasty-embedded-software-bugs/>

##### **Reducindo o tamanho dos dados**

- Geração de dados usando instruções
- Tamanho de buffer, empacotamento de dados...



- (**Problemática inicial** - **explicação**) O uso de memórias, principalmente MEMÓRIAS DE DADOS (ou só “DADOS” se achar melhor), depende muito do ESTILO DE PROGRAMAÇÃO (de COMO o programador está ACOSTUMADO a utilizar DETERMINADA MÁQUINA):

- Se um programador já tem experiência com APLICAÇÕES EMBARCADAS provavelmente ele não vai sair “criando” variáveis grandes quando ele já conhece a necessidade e sabe que a necessidade é “POUCA” no “MUNDO dos EMBARCADOS”, por exemplo:

- “Ao se criar uma variável “buffer” para receber uma “string” que o usuário vai digitar pelo teclado sabemos que tal “string” é uma palavra de 10-20 caracteres, mas por segurança colocamos como 200 caracteres no seu tamanho”. << Então, que irá acontecer?

- **Para um desktop**: Provavelmente será só uma variável gastando MAIS MEMÓRIA do que PRECISA.

- **Para uma aplicação funcionando numa plataforma mais limitada (Aplicação embarcada no nosso caso)**: Esse “TIPO” de jeito de programar acaba abrindo mais espaço para otimização já que se um programa feito desse jeito que foi

descrito como exemplo para analisa-lo se poderá ver MUITAS VARIÁVEIS MAL DIMENSIONADAS como essa, o que dará MUITO ESPAÇO para OTIMIZAÇÕES.

- (Resumindo...) Se quem fez certo programa já teve os cuidados de NÃO GASTAR MEMÓRIA DESNECESSARIAMENTE o ESPAÇO que vai sobrar para OTIMIZAÇÃO vai ser MENOR. Caso essa pessoa não tenha os cuidados certos e crie um programa para a aplicação cheio de MAUS DIMENSIONAMENTOS o ESPAÇO que vai sobrar para OTIMIZAÇÃO vai ser MAIOR.

- Diante da problemática apresentada, algumas ESTRATÉGIAS SÃO SUGERIDAS:

- \*REUSO de constantes, variáveis, buffers de dados em DIFERENTES PARTES DO CÓDIGO. É uma estratégia que contraria “boas práticas de programação”, isso significa que:

- Quando pensarmos em “reaproveitar” um buffer, por exemplo, em diferentes partes do código isso significa dizer que esse código ficará “MENOS LEGÍVEL” e com “MAIOR POSSIBILIDADE DE ERROS ESTRANHOS” (sem explicação para o usuário ou até mesmo o programador).
  - Afinal, uma variável como a usada como exemplo, um buffer no caso, que é declarada “GLOBALMENTE” (num determinado método ela é usada para receber uma string de um teclado, noutro método é usada como vetor intermediário de uma certa computação, e etc ...) por começar a ser “reaproveitada” por vários métodos que a chamam começa a ir criando espaço para alguns erros no código que podem se tornar difíceis de detectar (PROBLEMAS DE CONCORRÊNCIA).
  - O que acontece é que ALGUMAS VEZES essa estratégia é o “ÚNICO RECURSO” que o PROGRAMADOR TEM DISPONÍVEL! (Para “economizar um determinado número de bytes que estejam faltando para fazer com que o código caiba na memória do processador” por exemplo).
  - Essa é uma estratégia que utiliza de programação de “BAIXO NÍVEL” (onde por limitações do hardware o programador acaba tendo que fazer concessões quanto a qualidade do seu código).
  - A linguagem que os códigos dessa estratégia usa tipicamente REQUER VERIFICAÇÃO CUIDADOSA DA SUA “CORREÇÃO”.
  - Possui um TIPO DE PROGRAMAÇÃO que deve ser usada na maioria das vezes em ÚLTIMO CASO e COM BASTANTE CUIDADO.
  - Deve ter-se bastante cuidado no USO DE VARIÁVEIS GLOBAIS como estratégia de se deixar o código MAIS ENXUTO pelo fato dessas passarem a ser “DISPUTADAS POR MAIS DE UM MÉTODO” gerando PROBLEMAS DE CONCORRÊNCIA (difícies de se detectar) na APLICAÇÃO.
- \*\*Geração de DADOS usando INSTRUÇÕES. Pode ser que precisemos de informações em um código que podem SER CALCULADAS em TEMPO DE EXECUÇÃO o que fará com que haja um prejuízo no TEMPO DE COMPUTAÇÃO mas por outro lado haverá economia dos parâmetros necessários a uma determinada estrutura de dados que já estarão previamente calculados.

- \*\*\***EMPACOTAMENTO DE DADOS, Tamanho do buffer, etc.** Essa estratégia de **EMPACOTAMENTO DE DADOS** é muito interessante quando se tem **VARIÁVEIS BOOLEANAS**:

- Por exemplo, sabemos que a LINGUAGEM C utiliza um TIPO INTEIRO para representar um TIPO BOOLEANO, ou seja, um PROCESSADOR pode utilizar uma posição de memória de 16 bits onde o que interessa é se determinada variável é 0 ou é 1. Podemos usar EMPACOTAMENTO DE DADOS, que significaria utilizar cada bit de determinada palavra para uma VARIÁVEL BOOLEANA, então se determinada aplicação possuir MUITAS VARIÁVEIS BOOLEANAS pode ser interessante CRIAR UM MÉTODO para gerenciar aquela palavra e aí ao invés de ocupar-se CENTENAS DE POSIÇÕES DE MEMÓRIA para GRAVAR BOOLEANOS pode-se REDUZIR isso a POCOS BYTES significando a “SALVAÇÃO” para determinada aplicação em termos de RECURSOS DE MEMÓRIA.

- **ESTRATÉGIAS PARA REDUZIR O TAMANHO DO CÓDIGO:**

## Reduzindo tamanho do código

- Evitar *function inlining* (inserção da função em todas as suas ocorrências no código)
- Escolher processador com instruções compactas (ARM-Thumb, MIPS-16, CISC)
- Usar instruções especializadas, se possível (ASIP - DSP)

- **Evitar “function inlining”:** Especificamente para REDUÇÃO DO TAMANHO DO CÓDIGO os COMPILADORES EM GERAL assumem a ESTRATÉGIA DE BUSCAR o MENOR TEMPO DE COMPUTAÇÃO POSSÍVEL. Então, uma das coisas que eles fazem AUTOMATICAMENTE é INSERIR O MÉTODO no LOCAL onde ele É CHAMADO (chamado de “*function inlining*”), ou seja:

- Se determinada aplicação declara uma função que é chamada 20 vezes num código, se essa função não for muito grande, automaticamente o compilador sairá colocando em CADA CHAMADA o CÓDIGO DAQUELA FUNÇÃO o que fará o DESEMPENHO SER MELHOR (pois se ECONOMIZA o tempo de chamada e de retorno a função) mas o TAMANHO DO CÓDIGO CRESÇA. << (Solução para redução do tamanho do código):

>>> Pode-se avisar para o COMPILADOR que não queremos esse SERVIÇO de “*function inlining*” (como “*function inlining*” é uma prática que muitos programadores costumam fazer sabe-se que para ECONOMIZAR MEMÓRIA DE CÓDIGO devemos evitá-la!).

- Usar o recurso que alguns processadores oferecem de “INSTRUÇÕES COMPACTAS” (Escolher processador com instruções compactas). Essa instruções compactas são INSTRUÇÕES MENORES que vão permitir que o código de uma aplicação se torne 60-70% MENOR do que UTILIZANDO as INSTRUÇÕES NORMAIS DO PROCESSADOR.

- Usar instruções especializadas, se possível (ASIP - DSP):

- Os processadores especializados ASIP tem INSTRUÇÕES ESPECIALIZADAS MAIS COMPACTAS do que INSTRUÇÕES NORMAIS de PROCESSADORES COMUNS. Então, além de

**DIMINUIR O TEMPO DE COMPUTAÇÃO** eles vão AJUDAR A DIMINUIR O TAMANHO DO CÓDIGO.

