UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Sistema Dinâmico de Economia de Energia em RTOS

César Augusto Marcelino dos Santos

Itajubá, fevereiro de 2017

UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

César Augusto Marcelino dos Santos

Sistema Dinâmico de Economia de Energia em RTOS

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Sistemas de Computação - Hardware e Software Básico

Orientador: Prof. Dr. Carlos Henrique Valério de Moraes
Coorientador: Prof. Dr. Rodrigo Maximiano Antunes de Almeida

Fevereiro de 2017
Itajubá - MG

Agradecimentos

Primeiramente, gostaria de agradecer aos meus orientadores desta Dissertação de Mestrado: prof. Dr. Carlos Henrique Valério de Moraes, o qual me apresentou as melhores técnicas para se crescer academicamente conciliados com projetos de indústria, e o prof. Dr. Rodrigo Maximiano Antunes de Almeida, que me acompanha desde a minha primeira Iniciação Científica em 2011 e me apresentou o mundo dos Sistemas Embarcados.

Agradeço também à empresa Honeywell por ter me apoiado a realizar a Pós-Graduação e incentivar desenvolvimento continuado de seus colaboradores.

Sem dúvida, outra pessoa que me ajudou a chegar neste nível de maturidade na área de Sistemas Embarcados foi meu colega e grande amigo Lucas Carvalho de Sousa, o qual sempre apaixonado por Eletrônica, me auxiliou muito a aprender e navegar neste assunto quando éramos bolsistas de Iniciação Científica.

Além disso, agradeço à minha família, que até hoje me apoiou incondicionalmente para a conclusão do meu Mestrado. Em especial, minha esposa e filho que entenderam os momentos que me privei de estar com eles para seguir este projeto.

Não posso deixar de citar a UNIFEI, e todos os projetos e professores que passaram por minha vida acadêmica, pois foram estes que me direcionaram e deram toda a formação para que eu me consagrasse um Engenheiro da Computação, e em breve Mestre em Ciência e Tecnologia da Computação.

Resumo

Este documento apresenta a utilização de diferentes modos de economia de energia aplicados em sistemas operacionais de tempo real, quando o mesmo não possui tarefas a serem executadas. Espera-se que, ao entrar em modos de baixo consumo oferecidos pelo próprio microcontrolador, haja uma diminuição no consumo de energia, proporcional ao número de módulos internos que são desativados. Entretanto, a economia pode não se mostrar válida de acordo com a carga de processamento que o sistema possui, pois a corrente instantânea consumida após a transição do estado de baixo consumo para o modo ativo é superior ao consumo do sistema em operação constante. Desta forma, será apresentada uma política de economia de energia e como aplicá-la, simultaneamente minimizando a carga energética e balanceando a exigência de processamento.

Palavras-chaves: Política energética, Sistemas de tempo real, Sistemas embarcados, Baixo consumo

Abstract

This document presents different power saving modes applied to real time operating systems, whenever no tasks are being executed. It is expected that low power modes supported by the microcontroller diminishes energy consumption, proportional to the number of internal modules deactivated. On the other hand, this economy can become invalid based on system processing load, because instant consumed current after changing from a low power state to active mode is higher than a same system under continuous operation. This way, a power saving policy will be presented as well as how to apply such, simultaneously minimizing energy load and balancing it with load processing.
Key-words: Energy policy, Real-time systems, Embedded systems, Low consumption

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de Siglas

ABB - *Adaptive Body Bias*

ACK - *Asynchronous Clock*

ACPI - *Advanced Configuration and Power Interface*

API - *Application Programming Interface*

ARM - *Advanced RISC Machine*

CMOS - *Complementary Metal Oxide Semiconductor*

CPU - *Central Processing Unit*

CSB - *Context Switch Bitmap*

DMA - *Direct Memory Access*

DPD - *Dynamic Power Down*

DPM - *Dynamic Power Management*

DVFS - *Dynamic Voltage and Frequency Scaling*

DVS - *Dynamic Voltage Scaling*

EDF - *Earliest Deadline First*

FPGA - *Field-Programmable Gate Array*

GPIB - *General Purpose Interface Bus*

GPL - *General Public License*

I/O - *Input/Output*

IDE - *Integrated Development Environment*

KDS - *Kinetis Design Studio*

LPM - *Low-Power Mode*

MCU - *Microcontroller Unit*

NPLC - *Number of Power Line Cycles*

PLL - *Phase-Locked Loop*

POR - *Power-On Reset*

QoS - *Quality of Service*

RAM - *Random Access Memory*

RISC - *Reduced Instruction Set Computer*

RR - *Round Robin*

RT - *Real Time*

RTOS - *Real Time Operating System*

SCPI - *Standard Commands for Programmable Instruments*

SDA - *Serial and Debug Adapter*

SEU - *Single Event Upset*

SMU - *Source/Measure Unit*

SO - Sistema Operacional

SP - *Stack Pointer*

SRAM - *Static RAM*

TCB - *Task Control Block*

TLB - *Translation Lookaside Buffer*

VLPW - *Very Low Power Wait*

WCET - *Worst-Case Execution Time*

Sumário

1 Introdução

Sistemas operacionais de tempo real (RTOS - *Real Time Operating Systems*) são uma forma de abstração utilizadas principalmente na programação de sistemas embarcados para simplificar e facilitar a implementação de atividades com requisitos de tempo real, os quais possuem *kernels* capazes de atender demandas temporais. Em geral, um RTOS consome recursos de processamento para gerenciar uma lista de tarefas, realizando comparações entre o intervalo de reexecução definidos por cada processo e um relógio interno (*tick*) do próprio sistema.

Uma das características importantes em sistemas embarcados é o consumo de energia, visto que diversas aplicações são desenvolvidos para operar com baterias.

Diversos trabalhos já propuseram soluções para determinar e resolver este impasse, seja ele por sistemas reconfiguráveis, que podem combinar a flexibilidade de um processador de propósito geral e a eficiência de um hardware dedicado através de tecnologia FPGA [7] ou através de técnicas que combinam software e hardware, tais como ajuste dinâmico de tensão e frequência de operação (DVFS - *Dynamic Voltage-Frequency Scaling*) ou gerenciamento dinâmico de energia (*DPM - Dynamic Power Management*) [8].

Entretanto, ao se trabalhar com técnicas de economia de energia baseadas em modos de baixo consumo do microcontrolador, pode ocorrer um aumento repentino de corrente ao retornar para seu modo de execução, podendo anular a economia prevista. Além disso, outro critério essencial para um RTOS realizar operação de forma adequada é o seu determinismo, ou seja, a capacidade de rapidamente detectar novos eventos a serem processados, a qual não necessariamente significa rapidez de processamento de eventos (responsividade).

As análises de determinismo e consumo de corrente foram implementadas com base no sistema operacional de código aberto FreeRTOS [9].

1.1 Objetivos

Criar e aplicar uma política de redução de consumo de corrente de um sistema, baseada em seu histórico da carga de processamento, de forma a otimizar a energia necessária para operar um sistema do tipo *soft real-time*, sem perder o determinismo necessário para a aplicação-alvo.

1.2 Estrutura da Dissertação

Este documento se divide em quatro partes: o capítulo 2 traz uma revisão bibliográfica do tema e a análise de outros trabalhos e projetos desenvolvidos para RTOS, assim como técnicas semelhantes à utilizada. O capítulo 3 apresenta o desenvolvimento da proposta, bem como o processo de coleta de dados, exigindo configuração de hardware medidor apropriado, modificações no *kernel* do sistema e automação em software para coleta de dados. O capítulo 4 mostra os resultados obtidos, demonstrando o aumento repentino de corrente em determinadas ocasiões, assim como a validade da economia de energia baseada na carga dinâmica de processamento. Por fim, o capítulo 5 conclui com os prós e contras da nova técnica proposta, analisando como aplicá-la em outros sistemas de forma efetiva.

2 Referência Bibliográfica

2.1 Sistemas Operacionais

O sistema operacional, SO, é um conjunto de códigos que funciona como uma camada de abstração do *hardware*, provendo funcionalidades para as aplicações de alto nível [10]. Este isolamento permite que a aplicação não sofra alteração quando há mudança no *hardware*. Esta é uma característica muito desejada em sistemas embarcados, nos quais existe uma pluralidade nos tipos de periféricos, dificultando a reutilização de código.

De acordo com [11], o cerne (*kernel*) de um Sistema Operacional é composto de cinco módulos: gerenciamento de tempo (um *clock* interrompe o *kernel* periodicamente, para que possa mover processos entre filas de espera e de execução), gerência de processos (coloca os processos em fila para execução, assim como permite comunicação entre tarefas), gerência de interrupções (interrupções podem ser mapeadas como objetos do sistema para maior controle), gerência de memória (controle sobre a alocação de memória das tarefas) e gerenciamento de exceções (permite que o *kernel* manipule exceções ocorridas durante a execução).

De forma simplificada, os sistemas operacionais possuem três principais responsabilidades [12]:

- manusear a memória disponível e coordenar o acesso dos processos a ela;
- gerenciar e coordenar a execução dos processos através de algum critério;
- intermediar a comunicação entre os periféricos de *hardware* e os processos.

Estas responsabilidades se relacionam com os três recursos fundamentais de um sistema computacional: o processador, a memória e os dispositivos de entrada e saída. Na figura 1 estes recursos são ilustrados.

A ausência de um sistema operacional implica que toda a responsabilidade de organizar o andamento dos processos, os acessos ao *hardware* e o gerenciamento da memória é do programador. Este aumento de responsabilidade, a baixa capacidade de reutilização de código, e a consequente necessidade de recriar os códigos e rotinas, podem ser causadores de erros nos programas.

A capacidade de se reutilizar os programas é benéfica por dois pontos principais: diminui o tempo para entrega do projeto e permite que o programador utilize melhor o tempo, eliminando os erros ao invés de recriar os códigos. A consequência é a diminuição do custo de produção do projeto, existindo o risco de propagar vulnerabilidade não documentadas de códigos legados.

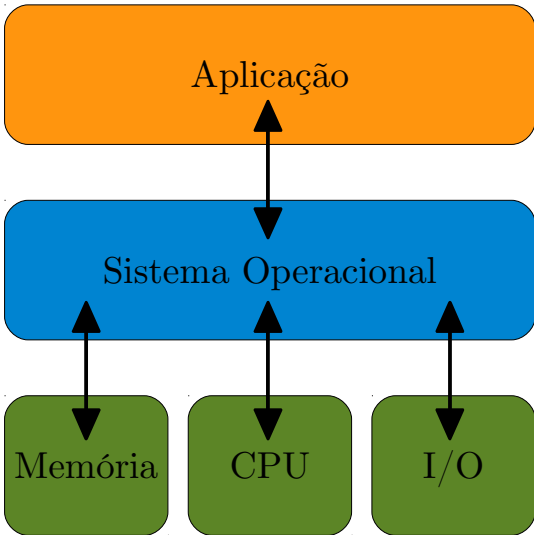


Figura 1 – Interfaceamento realizado pelo sistema operacional

A figura 2 apresenta com mais detalhes os componentes de um sistema operacional. Nota-se que o núcleo de um SO é o *kernel* e, do mesmo modo que o sistema operacional realiza a interface entre a aplicação e o *hardware*, o *kernel* faz a interface entre os códigos de acesso ao *hardware*, conhecidos como *drivers*, e as ferramentas disponibilizadas para que o programador crie as aplicações.

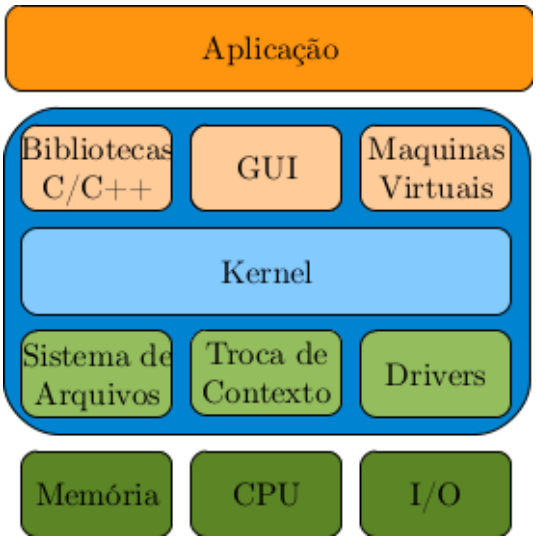


Figura 2 – Relação entre troca de contexto e o *kernel*

As aplicações, na presença de um sistema operacional, são implementadas como processos que passam a ser gerenciados pelo *kernel*. A sequência com que os processos são executados fica a cargo de algoritmos conhecidos como escalonadores, que por sua vez dependem de um procedimento de troca de contexto para efetuar a mudança de qual processo será executado.

A troca de contexto é o procedimento pelo qual um processo A, que está em execução no processador, é pausado, e um processo B toma seu lugar para iniciar, ou

continuar, sua execução. Este procedimento acontece dentro de uma das rotinas de interrupção do sistema que indica ao *kernel* o momento de realizar essa troca. Esta interrupção pode ser oriunda de um relógio interno ou de algum evento externo. No segundo caso o sistema é denominado *tickless* [13].

A troca de contexto esta intimamente ligada com as operações do *kernel*. Toda a gestão dos processos culmina na troca de contexto ditada pelo algoritmo utilizado no escalonador. Por ser necessário operar com os registros internos do processador, esta é uma das poucas rotinas de um sistema operacional que não pode ser escrita totalmente em linguagem de alto nível, além de ser extremamente dependente da arquitetura e até mesmo do modelo de processador utilizado.

O contexto de um processo é formado pelo conjunto dos dados internos ao processador: seus registradores, ponteiro de programa, ponteiro de pilha, códigos de condição de execução, entre outros, além das variáveis e estado do próprio processo. É necessário salvar esse contexto de modo que o processador possa recuperá-lo mais tarde e continuar a execução exatamente do ponto onde parou. É comum que todas estas informações, tanto do processador quanto do processo, se encontrem na pilha no momento da interrupção da troca de contexto. Isto acontece automaticamente por causa da estrutura utilizada para o tratamento de interrupção conforme ilustrado na figura 3.

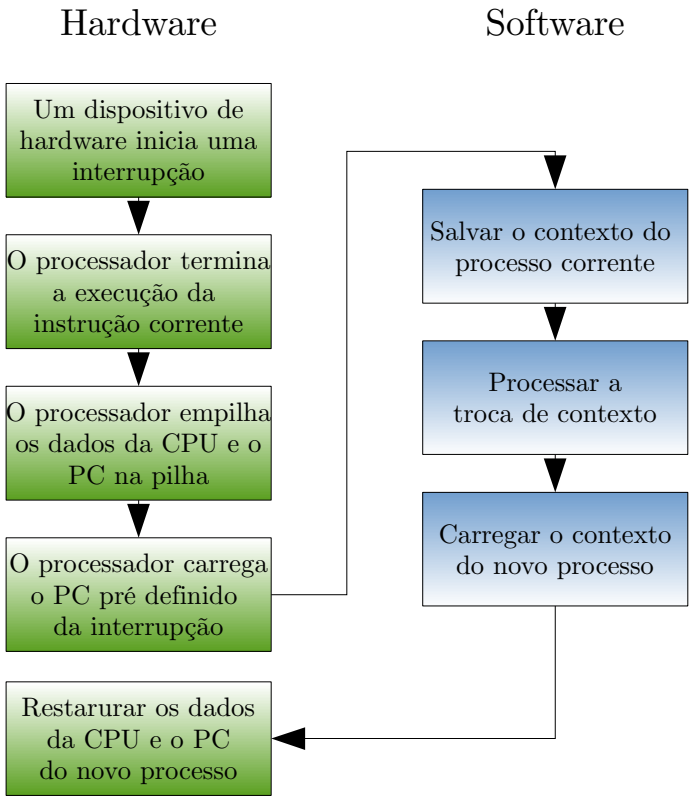


Figura 3 – Processo de interrupção e salvamento de contexto [1]

Se o processador utilizado é compatível com a estrutura apresentada na figura 3, basta alterar a parte do *software* para que, ao invés de retornar para o processo que estava em execução antes da interrupção, o fluxo volte para um segundo processo arbitrário. O modo mais simples de fazê-lo é implementar uma pilha de memória para cada processo, bastando então apenas mudar o ponteiro de pilha para a pilha do segundo processo antes de o *hardware* restaurar os dados.

Na figura 4 são apresentas as quatro etapas básicas da mudança de contexto em um sistema: interrupção do processo em andamento, salvamento do contexto atual, troca dos dados, mudança no ponteiro de pilha - *stack pointer* ou SP e, por fim, restauração do novo contexto. Tal esquema pode apresentar variações dependendo da arquitetura, podendo haver uma pilha separada somente para interrupções [14].

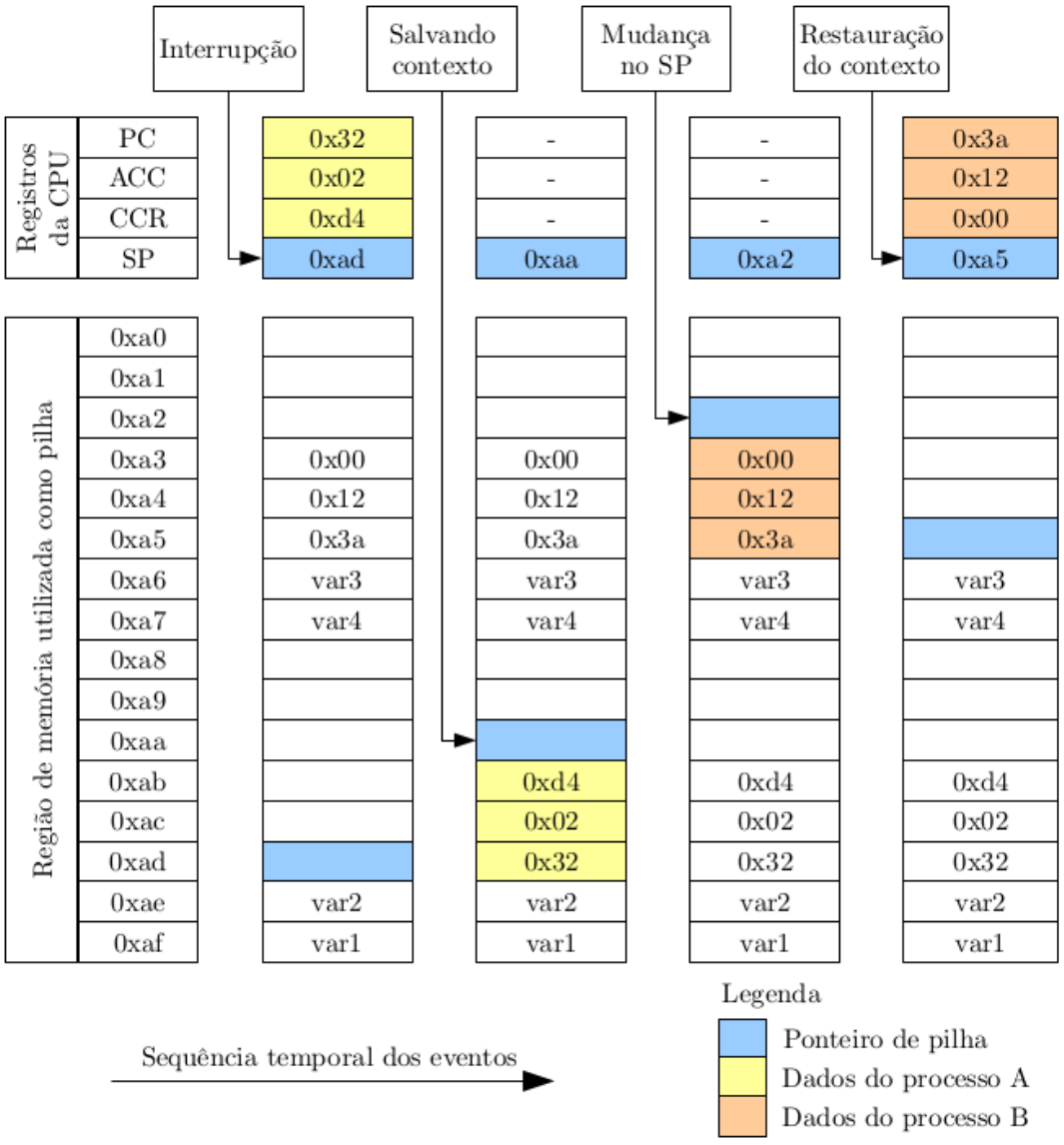


Figura 4 – Troca de contexto e manipulação dos dados

As regiões demarcadas em azul representam a posição indicada pelo ponteiro de pilha (SP). Na primeira etapa, este valor é incrementado automaticamente devido ao *hardware* de interrupção. Este incremento visa a criar espaço enquanto os valores do contexto atual (em amarelo) são salvos na pilha.

A mudança no SP, da segunda para a terceira coluna da figura 4, permite que o sistema carregue as informações referentes ao segundo processo (em laranja). Após a restauração do contexto, o sistema começa, ou continua a executar, dependendo do estado que o segundo processo se encontrava.

2.1.1 Sistemas Operacionais de Tempo Real

Os chamados Sistemas Operacionais de Tempo Real (RTOS) são caracterizados pela habilidade de atender a eventos com pouco ou nenhum atraso e, em algumas vezes, de processar eventos rapidamente ou em grande quantidade de dados, dentro da expectativa de tempo do usuário. Estas características são determinadas e exemplificadas por [15] como determinismo, responsividade (*responsiveness*) e taxa de transferência de dados (*throughput*), respectivamente. Determinismo e responsividade são objetivos críticos para sistemas *hard real-time*. Se um sistema é rápido o suficiente para atender a todas as demandas temporais sempre, então aumentar sua velocidade é irrelevante. Porém deve-se determinar o quanto é suficiente sua velocidade de execução, pois perder uma demanda significa falha. Para tal, deve-se conhecer o conjunto de tarefas, prazos de execução, os períodos de reexecução destas tarefas, o tempo alocado para a execução de um processo, etc.

Para [16], os RTOS se dividem em duas categorias: quando o atendimento a um serviço deve ser garantido, o ambiente é caracterizado como *hard real-time*, mas se uma distribuição estatística de tempos de resposta é aceitável, então o sistema é dito *soft real-time*.

Por outro lado, [17] classifica em três diferentes grupos:

- *Soft Real Time*: erros no determinismo do sistema são toleráveis e sua frequência determina a qualidade do serviço (*QoS - Quality of Service*). Comum para aplicações multimídia, em que a alta precisão de transmissão de cada bit não é necessária para compreensão da informação, ou seja, busca-se um equilíbrio entre *throughput* e responsividade;

- *Firm Real Time*: erros no determinismo do sistema são toleráveis, mas sua frequência pode torná-lo inutilizável. Um exemplo são a transmissão de dados em radiofrequência, como conexões 3G e *Wireless*. Nesta situação, o importante não é responsividade, mas *throughput*.

- *Hard Real Time*: erros no determinismo do sistema causam falhas consideráveis ou graves no sistema. Exemplo são aplicações médicas, como marca-passo ou então controle do sistema de direção de um veículo. Neste caso, responsividade é o foco.

Uma das formas para se determinar o *throughput* do sistema é através da taxa de utilização da CPU, ou carga máxima de processamento que, segundo [16], pode ser determinada de acordo com a quantidade m de tarefas do sistema:

$$U = m(2^{1/m} - 1) \tag{1}$$

Para valores muito grandes, a carga de processamento máxima teórica converge para 69.3%.

Em termos de determinismo, uma métrica comum é o cálculo do pior tempo de execução de uma tarefa do sistema (WCET – *Worst-Case Execution Time*), a qual pode fornecer o pior tempo de resposta que uma tarefa pode ter, tal como determinado em [11], explícito na equação 2.

$$R_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{\left\lceil \frac{R_j}{T_{tic}} \right\rceil T_{tic}}{T_j} \right\rceil C_j + \sum_{\forall j \in tasks} \left\lceil \frac{R_j}{T_j} \right\rceil C_{ql} + \left\lceil \frac{R_i}{T_{tic}} \right\rceil C_{int} \tag{2}$$

sendo R_i o pior tempo de resposta de uma tarefa, C_i o WCET, B_i o maior tempo de espera que a tarefa τ_i pode ter por atrasos causados por tarefas de menor prioridade (em situações de inversão de prioridade), T_j o tempo mínimo que uma tarefa τ_j pode ter para sair da fila de execução e efetivamente executar (equivalente ao período de *tick* T_{tic} para tarefas periódicas), C_{ql} o tempo máximo que uma tarefa pode demorar para ser removida da fila de espera, C_{int} o tempo para manipular a interrupção do *timer* do sistema, *tasks* o conjunto de todas as tarefas do sistema e *hp(i)* o conjunto de todas as tarefas de maior prioridade que a tarefa τ_i.

Como estas informações são difíceis de se adquirir em tempo de execução de um sistema, [18] propõe realizar este cálculo através de conjuntos de números nebulosos (*fuzzy*) para lidar com estas imprecisões, baseado no algoritmo EDF (*Earliest Deadline First*).

2.1.2 FreeRTOS

FreeRTOS [9] é um Sistema Operacional de Tempo Real com código aberto, cujo kernel tem permissão GPL, portável para mais de 30 arquiteturas. A missão do grupo que o mantém é "disponibilizar um produto gratuito que supere a qualidade e os serviços exigidos por usuários de alternativas comerciais".

O fato do kernel ser GPL permite que seu código possa ser reutilizado integralmente, ao mesmo tempo que protege a propriedade intelectual do sistema. Entretanto, qualquer modificação feita nos arquivos originais do *kernel* não podem ser de código fechado, e devem fazer, ao menos, referência ao site de distribuição do FreeRTOS.

Dentre as características que possui:

- escalonador preemptivo, cooperativo e possibilidades híbridas;
- implementações voltadas para baixo consumo, como o modo *tickless* (não há um *clock* interno que pede a troca de contexto);
- desenvolvido para ser pequeno, simples e de fácil uso, de modo que o arquivo binário do *kernel* possa ter entre 4K e 9KBytes;
- código-fonte altamente portável, predominantemente escrito em C;
- opção de detecção de estouro de pilha;
- sem restrições quanto ao número de tarefas de tempo real ou número de prioridades e é livre de *royalties*.

Os processos do FreeRTOS podem ser *tasks* (tarefas) ou *co-routines*. O primeiro permite maior segurança para o contexto (aqui chamado de *Task Control Block* – TCB) de cada tarefa, e também limita seu escopo, tornando uma tarefa FreeRTOS simples, sem restrições, com opções de preempção e prioridade. Já as *co-routines* foram desenvolvidas para os casos em que a memória RAM é limitada e, como consequência, a região de memória entre todos os processos é compartilhada.

Os diferentes estados que uma tarefa pode ter está representado na figura 5. Os estados são descritos como Disponível (*Ready*), Execução (*Running*), Bloqueado (*Blocked*) e Suspenso (*Suspend*). Assim que uma tarefa é criada ou quando está pronta para ser executada, vai para o estado *Ready*; se estiver em execução pelo sistema estará em modo *Running*; caso esteja indisponível indeterminadamente, a não ser que seja explicitamente chamada pelo sistema para o modo *Ready* é definido como estado *Suspend*; se uma tarefa está indisponível aguardando algum evento (normalmente relacionado a temporização, como por exemplo esperar 2 segundos para ser re-executada) está em modo *Blocked*.

Por fim, quando não há mais tarefa alguma para ser executada, uma pseudo-tarefa chamada IDLE passa a ser executada, inicialmente sem implementação alguma. Com isso, é possível disparar uma tarefa que seja específica quando a fila de processos estiver vazia.

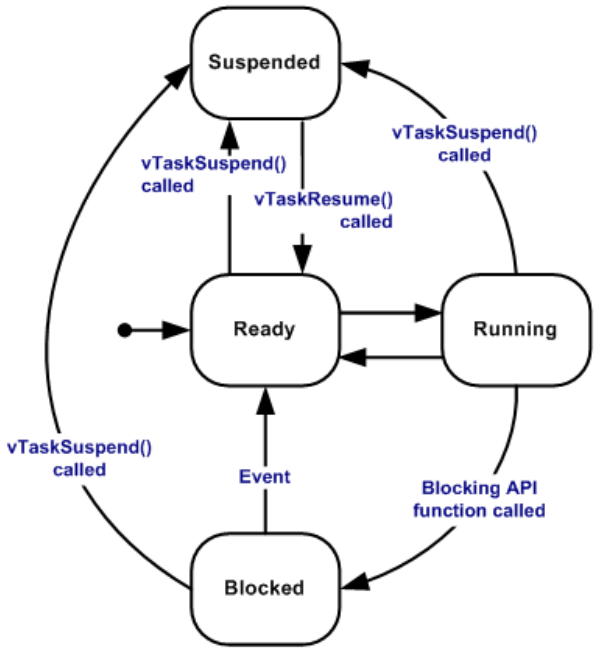


Figura 5 – Estados possíveis de uma tarefa do sistema FreeRTOS [2].

Apesar das facilidades que o sistema apresenta, seu padrão e estilo de codificação não são convencionais, de forma que as variáveis e funções são nomeadas baseadas em seu tipo e escopo, exigindo um prefixo especificado em documentação [19].

2.2 Sistemas Embarcados

Os sistemas embarcados são sistemas microprocessados projetados para um propósito ou aplicação específica [20,21], possuindo, em geral, poucos recursos de memória e processamento limitado. Na maioria dos casos, são sistemas projetados para aplicações que não necessitem de intervenção humana [22].

Outra característica marcante é a especificidade de suas aplicações, sendo geralmente projetados para realizar apenas uma função, não sendo possível alterações pelo usuário final. O usuário pode alterar ou configurar a maneira como o sistema se comporta, porém não pode alterar a função que este realiza [23–25].

Há algumas décadas se iniciou o movimento para migrar circuitos lógicos específicos a uma aplicação para um código que execute a aplicação específica em um processador. Tal influência se deve ao custo de configuração e manutenção de uma linha de fabricação. Desta forma, sistemas embarcados podem tomar duas frentes: utilização de chips comerciais ou tecnologias customizáveis por FPGA [26].

Um dos motivos desta crescente tecnologia é a necessidade de portabilidade, impondo restrições em tamanho, peso e energia. Energia é particularmente importante por conta de limitações de carga de baterias. Desta forma, os sistemas exigem cada

vez mais baixo consumo e alta taxa de transferência de dados. Mesmo em aplicações que não necessitam de baterias, o consumo é relevante por conta do aquecimento gerado por dissipação de energia, com a densidade de transistores cada vez maior nos circuitos integrados.

Entretanto, de acordo com um estudo publicado em 2013 [27], questiona-se a forma como os dados fornecidos pelo fabricante de um microcontrolador são levantados: se através da execução de uma instrução "NOP", por um *loop* infinito ou por um algoritmo específico, os quais não são discriminados em suas folhas de dados (*datasheet*). Além disso, não mencionam quais periféricos estão ligados e quais estão desligados. Deste modo, o que é fornecido em um *datasheet* representa uma referência, mas não uma realidade absoluta para todo e qualquer projeto. Para preencher esta lacuna, o consórcio EEMBC (*Embedded Microprocessor Benchmark Consortium*) [28] realiza a avaliação de diversos *benchmarks* voltados à computação heterogênea, sistemas móveis, computação ubíqua e também eficiência energética. Com relação a este último, o *ULPBench* executa uma série pré-definida de instruções e alterna os modos de economia de energia fornecidos pelo sistema, calculando-se uma métrica própria [29].

Para ilustrar tal cenário, as subseções a seguir descrevem os microcontroladores MSP430F5172 da Texas Instruments e o MKL25Z128VLK4 da Freescale, de acordo com as informações fornecidas pelo fabricante.

2.2.1 Microcontrolador MSP430F5172

Este microcontrolador possui baixos níveis de consumo de corrente, sendo uma linha da Texas Instruments definida como muito baixo consumo (*ultra-low power*) [5].

Este microcontrolador possui 6 diferentes modos de operação: ativo (*Active*), LPM0 (*Low Power Mode* 0), LPM1, LPM2, LPM3 e LPM4. A diferença entre eles define o número de clocks internos (periféricos ativos) trabalhando, assim como se a CPU está ativa ou não. Além de variar a quantidade de corrente consumida em cada um destes modos, também muda o tempo que é necessário para se sair de um modo LPM ao modo ativo (figura 6).

Em relação ao consumo de corrente de cada modo, a tabela 1 mostra as informações oferecidas pelo fabricante:

Para cada um dos estados energéticos que o microcontrolador se encontra, há uma latência diferente para retornar ao estado de execução (*Active*), chamado de *wake-up time*. O fabricante fornece estes tempos, os quais são vistos na tabela 2.

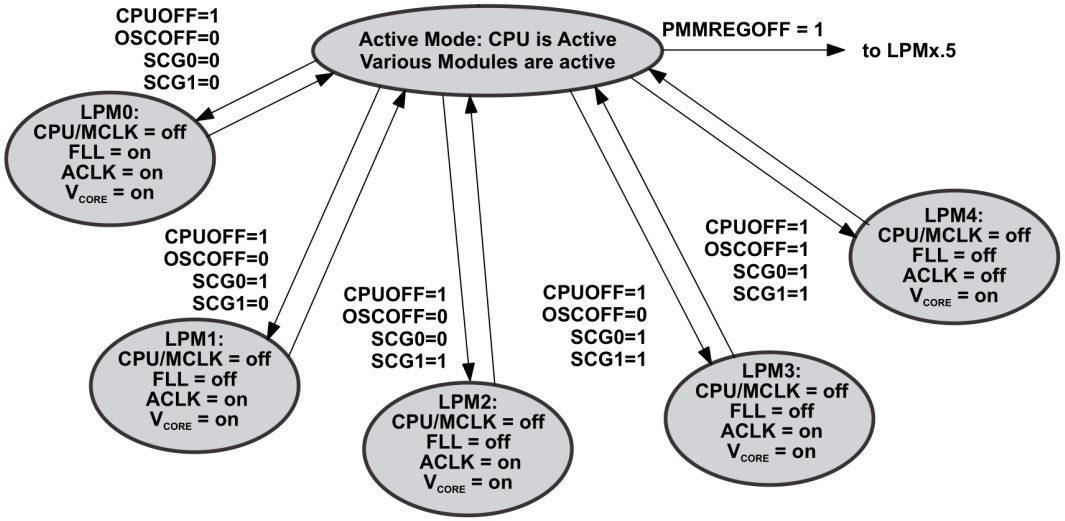


Figura 6 – Os modos de economia de energia do microcontrolador MSP430F5172 [3].

Modo de energia	Consumo médio de corrente
Active (25 MHz)	6,15 mA
LPM0	100 µA
LPM1	(não fornecido)
LPM2	13 µA
LPM3	1,5 µA
LPM4	1,3 µA

Tabela 1 – Consumo médio máximo de corrente do chip MSP430F5172 fornecidos pelo fabricante [5].

Modo de economia de energia	Tempo de *wake-up*
LPM0	≈ 0
LPM1	≈ 0
LPM2	6,5 µs
LPM3	6,5 µs
LPM4	6,5 µs

Tabela 2 – Tempos de *wake-up* para os diferentes modos de baixo consumo do chip MSP430F5172, de acordo com o fabricante [5].

2.2.2 Placa de prototipação KL25Z

A plataforma de desenvolvimento *Freedom* se trata de um conjunto de ferramentas em software e hardware para avaliação e desenvolvimento. O modelo KL25Z (codinome FRDM-KL25Z) contém uma série de periféricos, tendo como unidade central de processamento um microcontrolador ARM Cortex-M0+, modelo MKL25Z128VLK4 [3].

Este microcontrolador implementa diversos modos de operação: regulares (*Normal Run, Normal Wait, Normal Stop*), baixo consumo (*Very Low Power Run, Very Low Power Wait, Very Low Power Stop*) e modos de retenção (*Low Leakage Stop, Very Low Leakage Stop*3, *Very Low Leakage Stop*1, *Very Low Leakage Stop*0).

Os modos regulares interferem somente no núcleo de processamento, já os modos de baixo consumo alteram as velocidades de transmissão e sincronização de informações nos diferentes barramentos, diminuindo as taxas de clock. Por último, os modos de retenção desligam diversos clocks e algumas regiões de memória são parcial ou totalmente inacessíveis [30].

De forma análoga, nas tabelas 3 e 4 são vistos alguns dos consumos de corrente médio para os diferentes estados energéticos, assim como os tempos de *wake-up*.

Modo de energia	Consumo médio de corrente
Run (48 MHz)	5,9 mA
Wait	3,8 mA
Very Low Power Wait	366 µA
Stop	343 µA
Very Low Power Stop	8,46 µA

Tabela 3 – Consumo médio máximo de corrente do chip MKL25Z128VLK4 fornecidos pelo fabricante [6].

Modo de economia de energia	Tempo de *wake-up*
Wait	≈ 0
Very Low Power Wait	≈ 0
Stop	4,4 µs
Very Low Power Stop	4,4 µs

Tabela 4 – Tempos de *wake-up*para os diferentes modos de baixo consumo do chip MKL25Z128VLK4, de acordo com o fabricante [6].

Conforme visto, o fabricante não fornece o cenário de testes e a instrumentação para medição destes valores. A seguir será feita uma análise de quais fatores influenciam o consumo de energia dos núcleos de processamento, para se buscar uma técnica efetiva de sua medição, e avaliar se as informações aqui demonstradas são coerentes com as medições capturadas.

2.3 Consumo de energia de RTOS embarcados

De acordo com [31], técnicas primárias de economia de energia em sistemas embarcados se referem a desligar um periférico (tal como um disco ou a luz de fundo de um display) quando permanece um certo tempo ocioso. Por outro lado, é possível modificar a CPU para conservar energia, quando a mesma contém circuitos que podem modificar seu *clock* ou permitir diferentes tensões de alimentação.

Desta forma, o consumo de energia pode ser otimizado em diversas camadas: compilador, hardware, Sistema Operacional, e combinar as análises nestes diferentes domínios pode maximizar a redução de consumo. Por outro lado, combiná-las individualmente sem a análise desta integração pode anular a otimização de uma das camadas [32].

2.3.1 Definições

O consumo dinâmico de energia de um sistema CMOS pode ser visto na Equação 3, sendo C a capacitância média de chaveamento, V_t a tensão de limiar do dispositivo, $V_0 = (V_{max} - V_t)^2 / V_{max}$, T_s o período de amostragem do consumo de energia, f_r a frequência máxima de *clock* (ou seja, quando $V_{DD} = V_{max}$), e r a taxa de processamento normalizada, ou o *clock* de processamento normalizado em f_r [33]:

$$E(r) = C \cdot V_0^2 \cdot T_S \cdot f_r \cdot r \cdot \left[\frac{V_t}{V_0} + \frac{r}{2} + \sqrt{r \cdot \frac{V_t}{V_0} + \left(\frac{r}{2}\right)^2} \right]^2 \tag{3}$$

De uma forma mais simplificada e complementar, [8] define a dissipação dinâmica de potência aproximada pela fórmula da Equação 4:

$$P_d = C_l \cdot N_{sw} \cdot V_{dd}^2 \cdot f \tag{4}$$

tal qual C_l é a capacitância de carga, de forma que uma porcentagem desta está relacionada às ligações elétricas, N_{sw} o número de chaveamentos por ciclo de clock, V_{dd} a tensão de alimentação e f a frequência de clock. O encolhimento dos transistores permitiu a diminuição da tensão de alimentação necessária, mas aumenta o tempo de resposta dos semicondutores à impulsos elétricos, reduzindo sua frequência de *clock* efetiva. Tal cálculo pode ser aproximado para a Equação 5:

$$f = k \cdot \frac{(V_{dd} - V_{th})^2}{V_{dd}} \tag{5}$$

sendo k uma constante e V_{th} a tensão de limiar. Logo, P_d está cubicamente interligado à frequência, conforme visto na Equação 6:

$$P_d \approx C_l \cdot N_{sw} \cdot \frac{f^3}{k^2} \tag{6}$$

Como resultado, a redução na tensão de alimentação e na frequência de *clock* reduzem cubicamente a potência dinâmica (e quadraticamente a energia dinâmica), ao custo de um aumento linear no tempo de execução.

Já a dissipação estática de potência refere-se à corrente de fuga entre a fonte e o terra de um transistor. Corrente de fuga de *subthreshold* representa o consumo no qual o gate de um transistor está supostamente desligado, representando de 20% a 40% da dissipação total de energia de um sistema.

Por fim, potência de curto-circuito por *glitch* é a dissipação quando ambos os transistores n e p de um gate CMOS conduzem simultaneamente. Como as potências estática e dinâmica são superiores à potência de *glitch*, este componente não têm chamado tanta atenção da comunidade acadêmica e da indústria de sistemas de tempo real [8].

A partir dos parâmetros anteriormente definidos nas equações, as técnicas de otimização de consumo de energia se baseiam em adequar o software executado pelo sistema, de forma a minimizar o acesso a hardware ou diminuir seu tempo de execução, ou então utilizar técnicas e módulos de economia de energia que o próprio hardware já possui, conforme descrito nas subseções abaixo.

2.3.2 Otimizações em software

O método utilizado por [20] é a medição do consumo de corrente pelo processador ao executar blocos de instrução e avaliar seu consumo de energia. Entretanto, a sequência de execução proposta pode ser reordenada (por ser um sistema dinâmico, ou por otimização de compilação), impedindo o determinismo da medida de corrente.

Para resolver este problema, [34] mede usando um laço (*loop*) infinito com uma única instrução dentro, havendo uma instrução de salto (*jump*) embutida na execução. Para minimizar sua influência, a mesma instrução é inserida várias vezes no mesmo *loop*. A energia consumida por ela será a potência consumida multiplicada pelo seu tempo de execução, sendo o valor-base de energia consumido pela instrução.

Por outro lado, [35] desenvolveu o tradutor *UnStacked C*, o qual recebe como entrada um programa em C que utiliza *threads* preemptivas (utilizadas no sistema *TinyOS*) e o transforma para uma versão em máquina de estados orientada a eventos, reduzindo o consumo de memória e também seu consumo energético.

2.3.3 Otimização por acesso a hardware

Um dos problemas existentes na constante diminuição dos transistores é a subutilização da área de física dos chips (*dark silicon*) [36]. Por conta de limitação em ventilação e aquecimento em sistemas móveis, somente uma parcela do chip pode permanecer constantemente ativa. Desta forma, propõe-se uma "*arrancada computacional*" (*com-*

putational sprinting), utilizando todos os recursos possíveis para aumentar responsividade e, posteriormente, desativar o máximo de periféricos do chip. Desta forma, desenvolvedores de circuitos integrados devem buscar otimizar o consumo de energia durante sua ociosidade.

De forma oposta, diminuir a velocidade do processador para que a tarefa em execução faça completo uso da CPU é assumir que a tarefa terá seu pior caso de tempo de execução (WCET – *Worst-case Execution Time*), gerando momentos de ociosidade e diminuindo a velocidade de processamento [37]. Se o escalonador souber qual será o real tempo de execução, poderá reduzir a velocidade de processamento, e consequentemente economizar mais energia. Utilizando predição, a redução do consumo de energia pode ser ainda maior do que somente aplicar modos de baixo consumo. A predição pode ser mais eficiente e precisa aplicando conceitos de Inteligência Artifical e Redes Neurais, mas se torna uma implementação mais custosa.

Mas a tendência é que os chips sejam projetados com baixíssimo consumo de energia (*ultra-low-power*) [38], sendo que técnicas agressivas de escalonamento de tensão de alimentação tem o potencial de reduzir o consumo sem ferir os demais requisitos do sistema. Então, uma técnica de organização da memória SRAM em diferentes domínios de tensão de alimentação não somente pode causar uma diminuição da potência dissipada, mas também diferenciar quais informações estarão em quais bancos. Ao oferecer o suporte necessário para o compilador, pode ser observada uma redução de até 47% do consumo de energia [38].

2.3.4 Técnicas de modificação dinâmica do comportamento da CPU

2.3.4.1 DPD/DPM

De forma alternativa, uma das técnicas utilizadas nas implementações de sistemas de tempo real é chamada de DPM (*Dynamic Power Management*) ou DPD (*Dynamic Power Down*), a qual sugere trocar os componentes ociosos de seu estado corrente para um modo de baixo consumo de energia, ou "dormir". Porém, mudar de um estado ativo para econômico e depois retornar para o modo de execução gera uma sobrecarga chamada de *DPD overhead*. Desta forma, colocar o processador para dormir com frequência pode ser improdutivo. Se esta sobrecarga for menor ou igual ao consumo ocioso do sistema, então é válido utilizar DPD [39].

Colocar o processador para "dormir" (*sleep mode*) significa executar uma instrução que informe a mudança de estado em um registrador interno de controle, capaz de parar o *clock* do *core* de processamento. Isso efetivamente paralisa todas as atividades aplicando *Clock Gating* (desativação dos clocks quando não estão em uso, e reativação dos mesmos somente quando necessários). A interface de barramento permanece em operação com uma máquina de estados mais simples, de forma a garantir DMA

(*Direct Memory Access*) de periféricos enquanto o processador dorme. Para atingir o *sleep mode*, pode ser necessário diminuir a tensão de alimentação do processador e/ou do PLL (*Phase Locked Loop*) [40].

Fabricantes de software e hardware criaram o padrão ACPI (*Advanced Configuration and Power Interface)*, que introduzem diversos modos de operação, permitindo o desligamento de algumas partes do sistema como os núcleos de processamento, discos rígidos, módulos de Ethernet, etc [8]. Dispositivos mais complexos (como microprocessadores de notebooks ou tablets) já atendem a esta especificação. A intenção do DPM é que se utilize os modos mais intensos de economia de energia quanto mais ocioso o sistema estiver. No entanto, a desvantagem é que, quanto mais agressiva é a economia de energia, maior é a latência para retornar para um estado operacional.

A escolha de qual modo de baixo consumo deve se utilizar está mais intimamente ligado à fila de processamento, analisando se o processo seguinte é mais dependente de leitura/escrita de dados na memória (*memory-bound*) ou se necessita de maior processamento (*CPU-bound*).

Por último, [41] descreve que sistemas que aplicam DPM se baseiam na suposição de poder prever as variações de carga de processamento. Tais predições não podem consumir muita energia. Geralmente, uma tarefa gerenciadora de energia implementa os controles baseado nas observações. Tal controle é comumente chamado de *política*.

Na sequência, descreve que o desligamento da CPU é baseado em software e o acordar (*wake-up*) é orientado a interrupção. Desta forma, um *sleep mode* não pode reduzir seu consumo a zero pois partes do sistema não são gerenciáveis.

Para se entrar em um modo de baixo consumo é requerido o desligamento da fonte de alimentação. Então retornar para um estado ativo de execução requer uma latência para: ligar e estabilizar a fonte de alimentação e o clock; reinicializar o sistema; restaurar o contexto. Desta forma, equaciona-se a potência economizada de energia por realizar a transição para um estado S através da equação:

$$P_{saved,S} = (P_{On} - P_S) \cdot \frac{(T^{avg}_{idle>T_{BE,S}} - T_{BE,S})}{T^{avg}_{idle}} \cdot (1 - F(T_{BE})) \tag{7}$$

sendo $P_{saved,S}$ a economia por utilizar o estado S como política de DPM, P_{On} a potência consumida no modo ativo, $T^{avg}{}_{idle>TBE,S}$ a média de períodos ociosos maiores que $T_{BE,S}$ (chamado de *break-even time* para sair de um estado S para o ativo), $T^{avg}{}_{idle}$ a média dos períodos de tempo ociosos, $F(T_{BE})$ a distribuição de probabilidade de T_{idle} (ou seja, a razão de tempo para a qual o processador permanece ocioso).

Determinar a economia de energia de um sistema baseado em predições estáticas de processamento pode se mostrar ineficiente por incertezas na real carga de execução. Desta forma, não somente deve-se prever a carga de processamento, mas também adaptá-lo à mesma, ou seja, deve-se continuamente observar o histórico de tarefas.

2.3.4.2 DVS/DVFS

A proposta da técnica de ajuste dinâmico de tensão e frequência de operação (DVFS - *Dynamic Voltage-Frequency Scaling*, ou simplificadamente DVS – *Dynamic Voltage Scaling*), é diminuir a dissipação de potência dinâmica de um sistema, através do ajuste da frequência do núcleo de processamento e da fonte de alimentação, de forma que o processador se mantenha rápido o suficiente para concluir todas as tarefas em tempo [42].

Para implementar o DVS são necessários um sistema operacional capaz de modificar a velocidade do processador, um circuito regulador capaz de gerar a tensão mínima para a frequência desejada e um microprocessador com capacidade para operar em diferentes tensões [40].

Apesar de aplicações de DVS reduzirem o tempo de ociosidade alocado para as tarefas, aumenta as chances de um transiente ou SEU (*Single Event Upsets* – inversão do valor de um bit na memória pelo impacto de partículas subatômicas nos flip-flops) [40]. Desta forma, DVS e SEU são uma grande preocupação em aplicações espaciais [43]. Outra técnica chamada de ABB (*Adaptive Body Bias*) pode ser aplicada em conjunto com DVFS para evitar este problema. A mesma atua nas características dos transistores ao variar as tensões aplicadas internamente em seu corpo, necessitando de circuitos específicos para tal [40].

2.3.4.3 Sistemas mais complexos

Comparando diversos RTOS, quanto mais simples eles são, menos energia o escalonador consome, mas a adição de cada tarefa para escalonar aumenta o consumo consideravelmente [34]. Por outro lado, sistemas mais complexos consomem uma parcela significativa em seu escalonador, mas com o aumento de tarefas para execução, não se modifica muito o consumo para escalonar.

Desta forma, o consumo de energia durante uma troca de contexto é definido através da Equação 8 [44] define:

$$E_{CS} = \sum_{1 \leq i \leq n} E_r(i) + \sum_{1 \leq j \leq m} E_W(j) + E_{Schedule} + \sum_{1 \leq k \leq s} E_{Stage}(k) + E_{TLB} \tag{8}$$

sendo E_{CS} consumo de energia na troca de contexto, E_r consumo de energia em leitura de um único dado em um registrador, E_W consumo de energia na escrita de um único dado em um registrador, $E_{Schedule}$ consumo de energia na execução do escalonador do SO, E_{Stage} consumo de energia no esvaziamento do pipeline e E_{TLB} consumo de energia na perturbação do *Translation Lookaside Buffer* (TLB). Com isso, o consumo de energia médio de cada tarefa, assim como o consumo do sistema todo

são determinados:

$$E_{system} = \sum_{i=1}^{n} E_r(i) + P_{base} \times T_e \tag{9}$$

$$E_\tau(i) = \left(T_e(i) \times P_\tau(i) \times \frac{C_i}{T_i} \right) + NCS_\tau(i) \times E_{CS} \tag{10}$$

sendo E_{system} consumo de energia do sistema incluindo execução das tarefas e consumo imediato (sem a execução das tarefas), $E_{\tau(i)}$ consumo de energia da tarefa τ_i, P_{base} potência média do sistema sem a execução de uma dada tarefa, T_e tempo estimado de tarefas, $P_{\tau(i)}$ potência média de execução da tarefa τ_i, $NCS_{\tau(i)}$ número de trocas de contexto, E_{CS} consumo de energia de uma única troca de contexto.

Analisando outros aspectos de processadores mais complexos, o comportamento de algumas atividades do sistema depende das informações de entrada de sensores, dados em cache, *pre-fetching*, *pipelining*, DMA e interrupções, que aumentam a performance média, mas diminui o determinismo [45]. Desta forma, aplicar os conceitos tradicionais em ambientes altamente dinâmicos com hardware mais complexo gera desperdícios de recurso e aumentam seu custo. [46] e [47] aplicam o conceito de DPM para sistemas *multicore*, avaliando seu determinismo e responsividade, baseado nos números de núcleos de processamento ativos e de tarefas em fila para execução.

Para casos em que a carga de processamento é não-determinística, [48] propõe o planejamento de utilização de recursos e alocação de tarefas não-determinísticas através de predição baseado no histórico de consumo de uma estação base. Mesmo quando a predição gerava erros, o sistema de tempo real foi capaz de diminuir seu consumo de energia.

Já explicitamente para sistemas *hard real-time*, [49] propõe uma técnica para dinamicamente configurar o tempo de execução de cada tarefa, de forma a manter o sistema menos suscetível a falhas internas e ainda diminuir seu consumo de energia, baseada no número máximo de ciclos permitidos para se executar uma tarefa, no número máximo de ciclos para realizar diagnóstico e se criar um *checkpoint* de recuperação e também nos piores casos estimados de tempo de execução das tarefas.

Por outro lado, pode-se aplicar a redundância de processador (uma unidade primária e outra sobressalente). Deste modo, os dois processadores são capazes de atender a uma mesma fila de processamento, e caso um falhe, o outro o substitui para continuar a execução. Para otimizar o consumo de energia, utilizou-se uma técnica baseada em DPM e DVS, de modo que uma tarefa é responsável por analisar o consumo energético das demais tarefas, conforme Equação 11 [50]:

$$E_{PR}(T_i) = ET_i + (I_{Sub} \cdot V_{MAX} + C_{eff} \cdot V_{MAX}^2 \cdot f_{MAX} \cdot \rho_i^2) \cdot (AT_i + \tau_{EM}) \tag{11}$$

sendo ET_i a energia consumida pela mudança entre a tensão anterior e a tensão atual para executar esta tarefa (aplicação do DVS), I_{Sub} a corrente de fuga, V_{MAX} a tensão máxima de alimentação, C_{eff} a capacitância efetiva da carga, f_{MAX} a frequência máxima de processamento quando em V_{MAX}, $\rho_i = V_i / V_{MAX} = f_i / f_{MAX}$, AT_i o tempo de ativação para que o sistema "acorde" e inicie a execução da tarefa corrente, τ_{EM} o tempo gasto para executar a tarefa de gerenciamento de energia.

2.3.5 Avaliação das técnicas estudadas

Trabalhar com técnicas de DPM e DVFS requerem um reconhecimento do funcionamento do hardware e do dinamismo de execução do sistema, muitas vezes negligenciado durante a etapa de desenvolvimento de um projeto. Por outro lado, levantar diversos destes parâmetros atrasa a implementação de um sistema, pois muitas vezes é necessário avaliar a capacitância do sistema, ou até mesmo o consumo de energia de módulos individuais do microprocessador, como do escalonador. Considerando sistemas que passem por revisões em software e hardware, novamente tais parâmetros devem ser levantados.

A técnica sugerida se assemelha à equação 7, sendo necessário fazer uma primeira avaliação do real comportamento da CPU em seu estado ativo e seus modos de baixo consumo, conhecendo suas questões temporais (os tempos de *wake-up*), e a carga de processamento será determinada pelo próprio sistema de forma dinâmica e contínua, com o mínimo de sobrecarga de processamento.

3 Desenvolvimento

Com a meta de desenvolver um sistema de tempo real com minimização do consumo de energia, as técnicas e equações apresentadas anteriormente não são suficientemente documentadas sobre como adaptá-las à aplicação desejada, ou pouco praticáveis pela exigência de determinação de parâmetros dificilmente medidos.

As informações oferecidas pelos fabricantes também não indicam qual a opção ideal de modo de economia de energia quando se busca equilibrar determinismo e baixo consumo energético, ou como realizar combinações dos mesmos, deixando a cargo do desenvolvedor esta análise e responsabilidade. Apesar da forma de avaliação realizada pela EEMBC ser pública, não foram encontradas medições para o microprocessador MSP430F5172 disponíveis em seu website.

As seções abaixo mostram quais foram os hardwares selecionados como meio de teste, a instrumentação necessária para obter as medições e demonstrar a validade da técnica proposta, assim como a modelagem simplificada de consumo de energia do sistema.

3.1 Placas de teste

Foram utilizadas duas placas de teste voltadas a aplicações de baixo consumo de energia: um circuito mínimo com o microcontrolador MSP430F5172, da Texas Instruments [5], e a placa de prototipação e testes da Qualcomm (antiga Freescale), Freedom KL25Z [3]. Ilustração dos componentes e partes das duas placas podem ser vistos nas figuras 7 e 8.

De forma a restringir o consumo energético aos núcleos de processamento, instrumentação se fez necessária para não capturar ruidos advindos dos periféricos das placas: para a placa contendo MSP430, apenas interligaram-se dois fios de alimentação entre os terminais V_{SYS} e GND, tal qual visto na figura 7, enquanto na FRDM-KL25Z, foi necessário remover os resistores R73 e R81, e inserir a alimentação do sistema em J4, de acordo com o esquemático da figura 9.

Desta forma, a medição de potência se dará indiretamente através da corrente consumida pelos microcontroladores durante a execução do sistema, sem a interferência de periféricos.

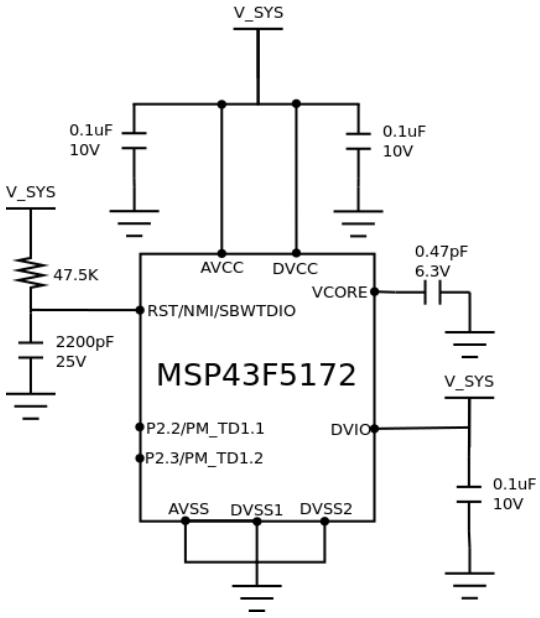


Figura 7 – Esquemático do circuito básico para operação do MSP430F5172.

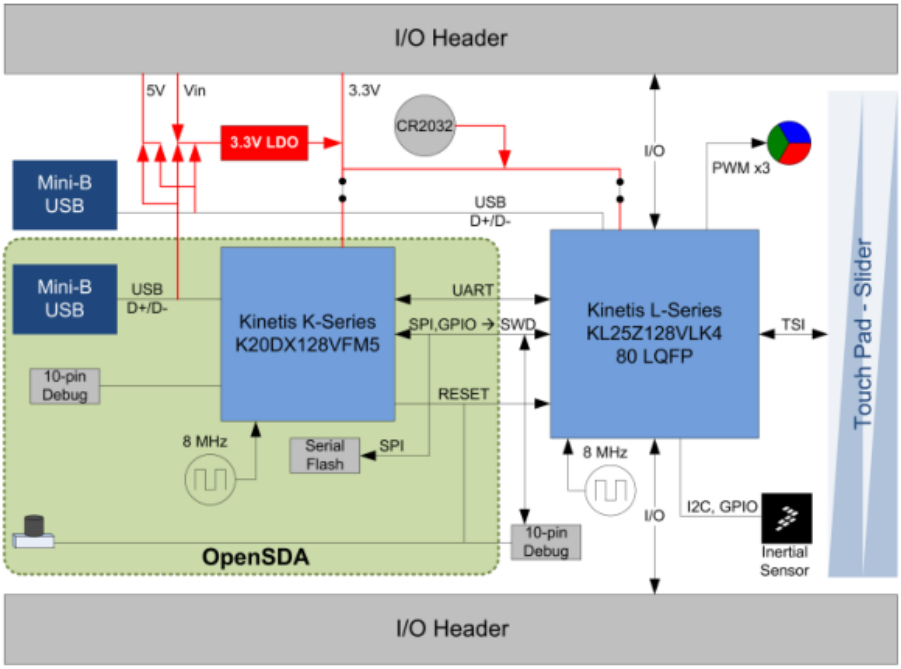


Figura 8 – Diagrama de blocos do sistema utilizado para operação do microcontrolador MKL25Z128VLK4 [3].

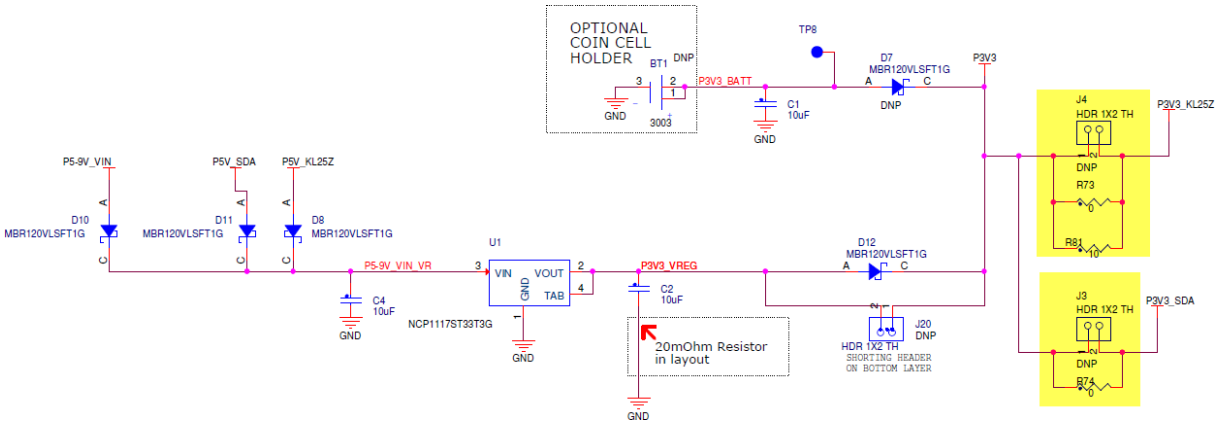


Figura 9 – Instrumentação realizada nos pontos R73, R81 e J4 na placa FRDM-KL25Z [4].

3.2 Instrumentos de medição

Inicialmente, planejou-se a medição automática através do controle por software da fonte de alimentação Keysight E3646A [51] e multímetro digital de bancada Tektronix DMM4040 [52, 53]. O motivo deste uso conjugado é que a fonte de alimentação não é capaz de capturar o consumo de corrente na faixa dos microampères.

A primeira limitação se deu na interface de comunicação com a fonte, pois esta só possui disponíveis as portas Serial e GPIB. A linguagem de programação utilizada para fazer este acesso foi C# através da IDE Microsoft Visual Studio 2015 [54], e foram necessárias implementações assíncronas para gerar estabilidade na comunicação Serial pela máquina virtual .NET da Microsoft [55–57]. Quanto ao multímetro, o mesmo possui comunicação Ethernet, interface de acesso mais comum em linguagens de alto nível, possuindo APIs prontas que realizam esta comunicação, além da maior velocidade de comunicação e facilidade na transmissão de dados em massa. Foi utilizada a API de comandos SCPI (*Standard Commands for Programmable Instruments*) para controlar os instrumentos [58].

Para apurar as leituras com maiores precisão e quantidade de pontos ao longo do tempo, utilizou-se a SourceMeter Tektronix 2602A [59, 60]. Apesar de o processo de coleta dos valores de corrente ser manual, com este instrumento foi possível capturar pontos a cada 110μs devido à configuração do NPLC *Number of Power Line Cycles*.

Mais detalhes da configuração do instrumento pode ser vista na Seção 6.3.

3.3 Determinismo do sistema e economia de energia

Cada vez que o sistema entra em modo de economia de energia, o mesmo só voltará ao seu estado ativo quando houver uma interrupção que o “acorde”. De acordo com

as configurações do microcontrolador e do estado no qual se encontra, diferentes periféricos internos estão desativados e as condições de interrupção se tornam restritas. No entanto, o projetista deve se atentar aos modos escolhidos para utilizar em sua aplicação, pois os mesmos podem bloquear periféricos-chave para sua execução, tal como o *timer* de *tick*. Este é o caso do LPM4 para o MSP430, que desativa o *clock* assíncrono (ACLK), o qual é responsável pelo ciclo de trocas de contexto.

Para restringir e diminuir o número de opções de estados de economia de energia, outro critério utilizado foi o *wake-up time*, sendo este impactante de acordo com as necessidades de determinismo da aplicação. Como visto na tabela 2, os modos LPM0 e LPM1 possuem tempos semelhantes, assim como LPM2 e LPM3. Para este estudo, os modos LPM0 e LPM2 foram descartados por haver opções análogas que consomem menos corrente e possuem mesmo tempo de *wake-up*.

Com o mesmo raciocínio, para o microcontrolador MKL25Z128VLK4, foram escolhidos os modos *Wait* e VLPW (*Very Low Power Wait*), pois os modos de retenção e *Stop* não permitem serem acordados por interrupção do *tick*.

Como plataforma de teste, o sistema FreeRTOS foi utilizado com um *tick* de 1ms de troca de contexto, sendo esta a resolução de determinismo exigida. Uma alternativa seria utilizar o modo *tickless*, desativando a periodicidade de interrupção do escalonador temporariamente, mas o mesmo não oferece uma garantia e manutenção do determinismo quando o sistema oscilar de uma baixa carga de processamento para uma alta.

3.4 Política de economia de energia

De forma semelhante ao determinado em [41], podemos medir a energia utilizada por um sistema baseado somente em seu consumo de corrente, visto que a fonte de alimentação permanece constante. Para formular o cenário em teste, é imprescindível a determinação dos intervalos de tempo de execução de uma tarefa (t_{ON}) e em economia de energia (t_{OFF}). Além disso, serão necessárias as medições de consumo de corrente do sistema em modo ativo (i_{ON}), em economia de energia (i_{OFF}) e em estado ativo após retorno do modo de economia de energia (i_{ON2OFF}). Conforme descrito em [39], as variáveis anteriormente definidas podem ser aproximadas pela figura 10.

O consumo médio de corrente do sistema (i_{LPM}), para uma aplicação que utilize modos de baixo consumo de energia, pode ser equacionado conforme apresentado na equação 12.

$$i_{LPM} = \frac{t_{ON} \cdot i_{ON2OFF} + t_{OFF} \cdot i_{OFF}}{t_{ON} + t_{OFF}} \tag{12}$$

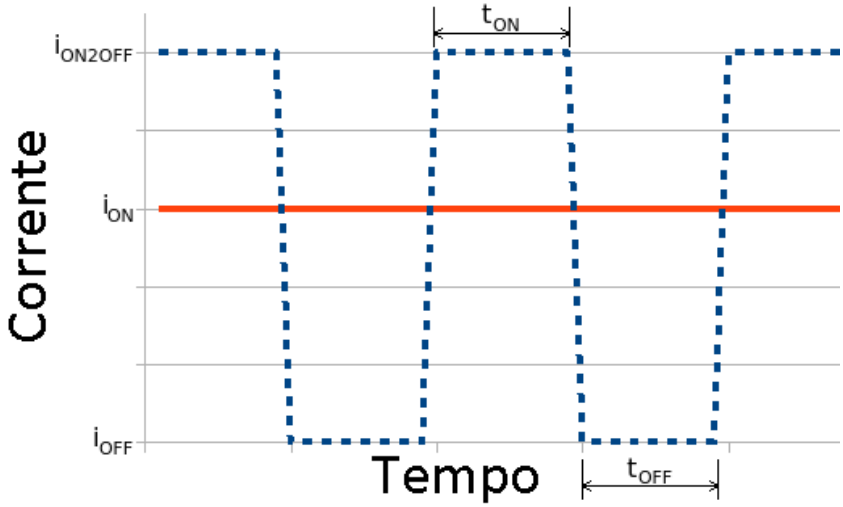


Figura 10 – Definição das variáveis t_{ON}, t_{OFF}, i_{ON}, i_{OFF} e i_{ON2OFF}.

Na equação 12 é considerado um sistema periódico com apenas uma tarefa em execução e uma tarefa idle, seguindo o conceito apresentado na figura 10. Para generalizar o conceito, podemos adotar a ideia de carga de processamento ($C_{\%}$). Este pode ser definido como o somatório dos tempos consumidos (t_{ON_i}) pelas N tarefas do sistema, dividido pelo tempo total (t), conforme equação 13. Este somatório representa o tempo total ativo em que o sistema está consumindo a corrente i_{ON2OFF}.

$$C_{\%} = \frac{\sum_{i=0}^{N} t_{ON_i}}{t} \tag{13}$$

De modo análogo, o tempo ocioso do sistema é definido pelo somatório de todos os intervalos de tempo em economia de energia dividido pelo tempo total. O tempo ocioso também pode ser definido como o tempo total menos o somatório dos tempos em execução, conforme equação 14.

$$C_{\%_{IDLE}} = \frac{\sum_{i=0}^{N} t_{OFF_i}}{t} = \frac{t - \sum_{i=0}^{N} t_{ON_i}}{t} \tag{14}$$

Substituindo a equação 13 na equação 14 obtemos a equação 15, relacionando o consumo de processamento $C_{\%}$ com o tempo ocioso $C_{\%_{IDLE}}$.

$$C_{\%_{IDLE}} = 1 - C_{\%} \tag{15}$$

Expandindo a equação 12 para considerar todos os tempos das tarefas em execução, bem como o somatório dos tempos ociosos, podemos reescrevê-la no formato da equação 16.

$$i_{LPM} = \frac{\left(\sum_{i=0}^{N} t_{ON_i}\right) \cdot i_{ON2OFF} + \left(\sum_{i=0}^{N} t_{OFF_i}\right) \cdot i_{OFF}}{t} \tag{16}$$

Utilizando as equações 13 e 14 em 16, pode-se calcular a corrente a ser consumida por um dado sistema utilizando-se as informações de consumo de corrente dos modos de execução e a carga de processamento esperada, conforme apresentado na equação 17.

$$i_{LPM} = C_{\%} \cdot i_{ON2OFF} + (1 - C_{\%}) \cdot i_{OFF} \tag{17}$$

Pela equação 17, a economia de energia em um sistema ocorre de acordo com os fatores $C_{\%}$, i_{OFF} e i_{ON2OFF}. Como a carga de processamento é dinâmica, deve-se buscar maximizar a influência da corrente em modo de baixo consumo. Para tal, quanto menor a carga de processamento, menor será a influência da corrente de retorno ao modo de execução.

Porém, a decisão de implementação do sistema dinâmico foi buscar balancear economia de energia com manutenção de determinismo para sistemas *soft real-time*, sem modificar a tensão de alimentação da CPU, sendo esta uma variação da técnica DPM [39]. Assim, quanto maior a carga, menor deve ser o impacto por tempos de *wake-up*, então menos agressivo deve ser o modo de baixo consumo selecionado.

A política proposta se baseia na redução temporária da potência dinâmica para o mínimo disponível, ou seja, enquanto o núcleo de processamento está "dormindo". A politica deve ser capaz de determinar o cálculo dinâmico da carga de processamento. Uma opção é monitorar a razão entre a quantidade de execuções da tarefa IDLE e a quantidade de trocas de contexto do sistema, ou seja o fator ($1 - C_{\%}$).

Por questões de notação, adotou-se o modo de economia de energia com menor tempo de *wake-up* (e menos econômico) como *Modo*1 (LPM1 para MSP430 e *Wait* para MKL25Z128VLK4), e o modo de economia de energia com maior tempo de *wake-up* (e mais econômico) como *Modo*2 (LPM3 para MSP430 e VLPW para MKL25Z128VLK4).

Deste modo, uma possível política P de economia de energia pode ser definida através de quais níveis de carga de processamento levam o sistema a adotar cada um dos modos de economia. Um exemplo de política é apresentado na equação 18.

$$P(C_{\%}) = \begin{cases} \textit{Execução} \Leftrightarrow C_{\%} = 100\% \\ \textit{Modo1} \Leftrightarrow 75\% < C_{\%} < 100\% \\ \textit{Modo2} \Leftrightarrow C_{\%} \leq 75\% \end{cases} \tag{18}$$

Ilustrando o algoritmo da política a ser implementada em um diagrama, tem-se a figura 11.

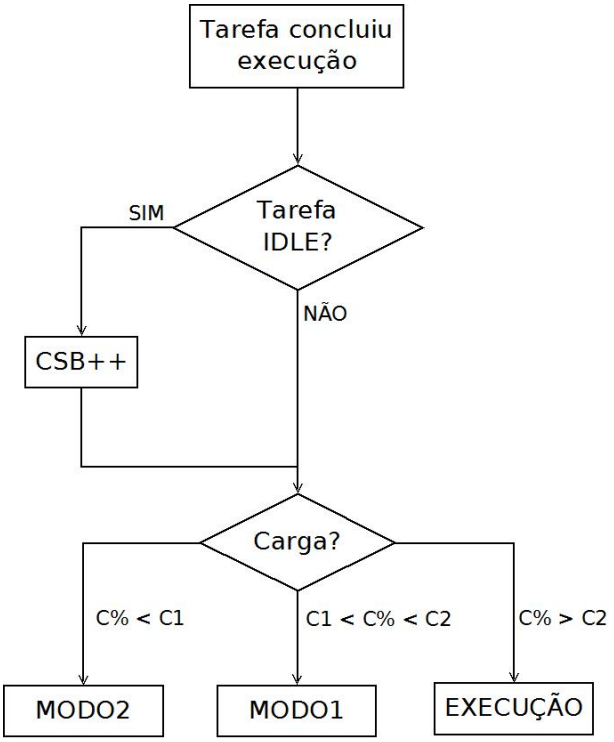


Figura 11 – Algoritmo proposto para a implementação da política dinâmica de economia de energia.

3.5 Implementação do sistema

Conforme mostrado na Seção 2.1.2, o sistema utilizado foi o FreeRTOS, versão 9.0 (a mais atual no momento da escrita deste documento). O mesmo já possuia ports tanto para o sistema do MSP430 quanto para a placa FRDM-KL25Z, facilitando sua integração inicial, a qual foi feita sem qualquer ativação de otimização no sistema.

Os ambientes de desenvolvimento utilizados para implementação do *firmware* foram as IDEs *IAR Embedded Workbench for MSP*430 versão 5.50.1 [61] e *Kinetis Design Studio* (KDS), versão 3.2.0 [62]. O Sistema Operacional da máquina *host* utilizado foi Windows 7 Home Premium Service Pack 1, 64-bits. O motivo de não se utilizar uma plataforma padrão tal para ambos os dispositivos, tal como o GCC, é a baixa capacidade de suporte e manutenção do projeto em comparação à soluções proprietárias durante o período de desenvolvimento deste trabalho (durante o ano de 2016 a Texas Instruments começou a dar maior atenção a este projeto [63]).

As modificações no sistema foram realizadas para permitir a observação de três eventos: sinalização das trocas de contexto no momento em que ocorrem, sinalização de mudança de estado ativo para estado de baixo consumo de energia, criação de histórico de execução de tarefas no sistema.

3.5.1 Sinalizações de troca de contexto e mudança de estado energético

O FreeRTOS possui algumas otimizações para reduzir a quantidade de trocas de contexto. Por esse motivo foi necessário forçar a rotina de troca de contexto a ser executada a cada *tick*, garantindo que o mesmo ocorrerá mesmo que não hajam mais tarefas na fila. A modificação realizada para o MSP430 é exemplificada no código 1.

Código 1 – Rotina de Tratamento de Interrupção modificada do tick do sistema feito em assembly para o MSP430

```
1  vPortTickISR:
2  /* The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
3  to save it manually before it gets modified (interrupts get disabled).
4  Entering through this interrupt means the SR is already on the stack, but
5  this keeps the stack frames identical. */
6  push.w sr
7  portSAVE_CONTEXT
8  calla #xTaskIncrementTick

10 /*-----------------------------------------------------------------------
11 ---------------- MODIFIED FOR POWER SAVING PURPOSES ----------------
12 -----------------------------------------------------------------------*/
13 //  cmp.w   #0x0, R12
14 //      jeq     SkipContextSwitch

16 calla #vTaskSwitchContext
17 SkipContextSwitch:
18 portRESTORE_CONTEXT
```

Para a visualização dos eventos (tarefa IDLE, troca de contexto, etc), foi utilizada a ativação e desativação de um terminal de I/O, externalizando essa informação para ser lida por um osciloscópio. O método *vPowerUpLed()*(código 2) é o responsável por controlar esse terminal.

O terminal (mapeado como pino de GPIO 3 nas placas de desenvolvimento) será ativado a cada troca de contexto dentro da função *vTaskSwitchContext()*(código 3), sendo ligado quando a função for chamada e desligado ao final da função, permitindo observar o tempo total da troca de contexto.

Código 2 – Função para instrumentação e monitoramento do processamento

```
1 void vPowerUpLed( portBASE_TYPE arg, portBASE_TYPE bit ){
2   if( arg == pdTRUE ){
3     #ifdef __MSP430F5172__
4       P2OUT |= 1 << bit;
5     #endif
6     #ifdef CPU_MKL25Z128VLK4
7       GPIO_WritePinOutput(GPIOB, bit, pdTRUE);
8     #endif
9   } else { // arg == pdFALSE
10    #ifdef __MSP430F5172__
11      P2OUT &= ~( 1 << bit );
12    #endif
13    #ifdef CPU_MKL25Z128VLK4
14      GPIO_WritePinOutput(GPIOB, bit, pdFALSE);
15    #endif
16  }
17 }
```

Código 3 – Modificação da função vTaskSwitchContext()

```
1 void vTaskSwitchContext(void) {
2   /* ************************************************************************
3   ********** MODIFIED FOR POWER SAVING PURPOSES **********
4   ************************************************************************ */
5   // debug here
6   vPowerUpLed(pdFALSE, 2);
7   vPowerUpLed(pdTRUE, 3);
8
9   ...
10
11  /* ************************************************************************
12  ********** MODIFIED FOR POWER SAVING PURPOSES **********
13  ************************************************************************ */
14  if (flagNotTick) {
15    flagNotTick = pdFALSE;
16  } else {
17    usContextSwitchBitmap <<= 1;
18    if (pxCurrentTCB->uxPriority == tskIDLE_PRIORITY) {
19      usContextSwitchBitmap++;
20    }
21  }
22  vPowerUpLed(pdFALSE, 3);
23 }
```

A outra observação é feita ativando um segundo pino (mapeado como pino de GPIO 2 nas placas de desenvolvimento) ao final da execução da tarefa IDLE (*vApplicationIdleHook()*), antes de determinar o novo estado de execução do sistema (se ativo ou baixo consumo). Sua desativação ocorre ao início da execução da função *vTaskSwitchContext()*.

Estes mapeamentos podem ser observados nas figuras 12 e 13.

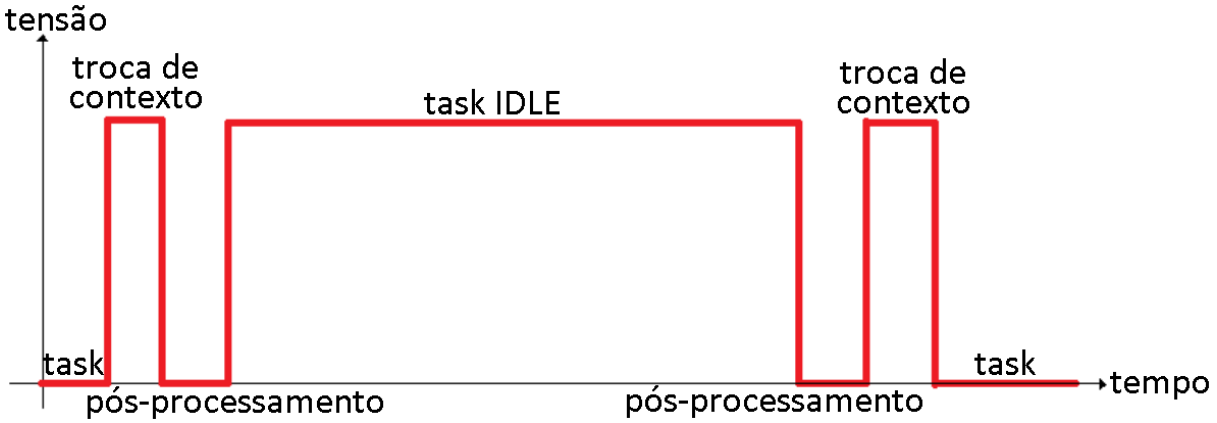


Figura 12 – Aproximação da forma de onda vista em osciloscópio para mapeamento da troca de contexto.

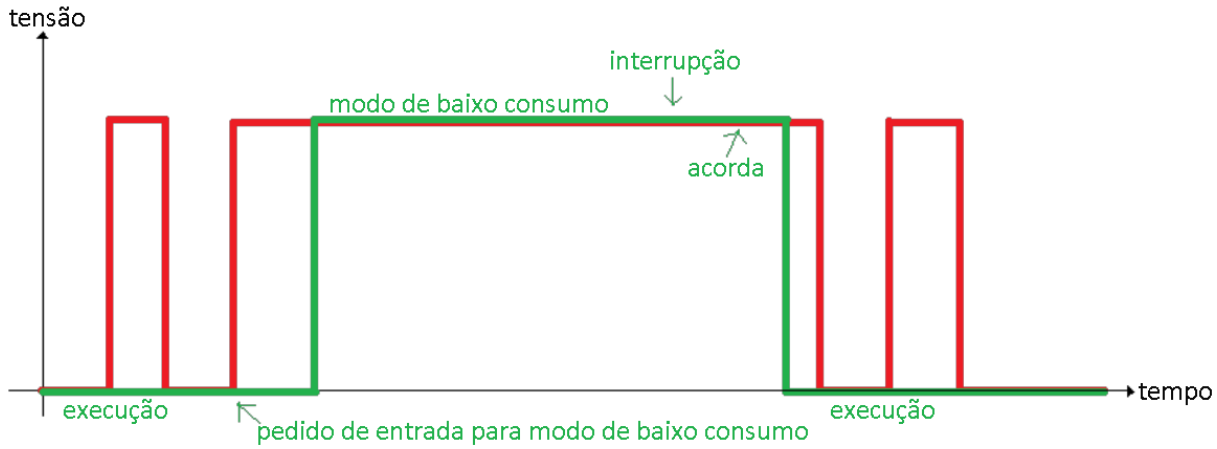


Figura 13 – Aproximação das formas de onda sobrepostas vista em osciloscópio para mapeamento da troca de contexto e da transição do microcontrolador para outro estado energético.

3.5.2 Captura e geração de histórico de execução das tarefas do sistema

De forma a gerar um histórico de processamento do sistema em questão, modelou-se um vetor o qual conteria as últimas execuções em um intervalo de tempo T, passados alguns *ticks*. Supondo tal vetor com 5 posições, a figura 14 demonstra a execução de 1, 2 e de 5 tarefas nos últimos 5 *ticks*, respectivamente.

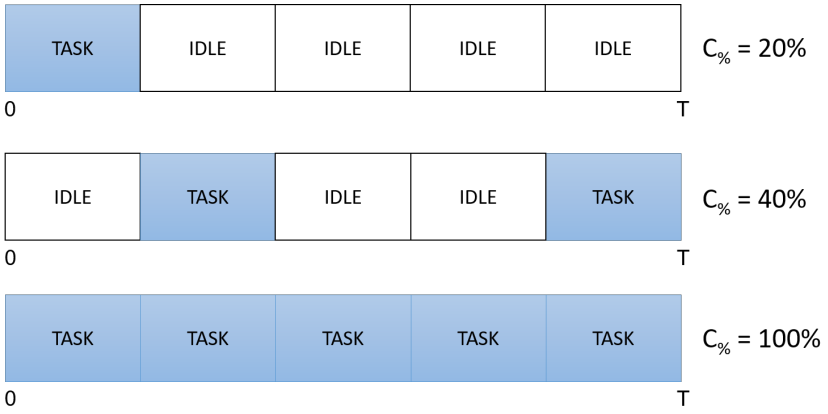


Figura 14 – Ilustração de histórico de execução de tarefas de acordo com o vetor proposto, contendo cargas de processamento de 20%, 40% e 100%.

Para gerar o mínimo de sobrecarga no sistema, deseja-se que este vetor seja um mapa binário (*bitmap*) que contenha o histórico das últimas vezes na qual o sistema estava ou executando uma tarefa específica (bit 0), ou a tarefa IDLE (bit 1). Logo, este mapa representa as trocas de contexto efetivas de uma tarefa para outra, o qual chamaremos de *Context Switch Bitmap* (CSB).

Supondo um CSB de 8 bits esteja preenchido com o conteúdo "00000011", e posteriormente com o conteúdo "01100011". Isso significa que, inicialmente, a carga de processamento do sistema era de 75% e passou para 50%.

A quantidade de tarefas contidas no CSB dependerá da frequência de *tick* do sistema e de seu tamanho. Na implementação testada, o vetor binário é representado pela variável de 16 bits *usContextSwitchBitmap*.

De forma a garantir que houve execução da troca de contexto, a variável binária *flagNotTick* foi determinada, modificando os métodos *vTaskDelayUntil()* (código 4) e *vTaskSwitchContext()*(código 3). Sem esta variável, o mapa binário poderia conter repetidas instâncias de uma tarefa dentro de um mesmo *tick*.

Código 4 – Modificação da função vTaskDelayUntil()

```
1 void vTaskDelayUntil(portTickType * const pxPreviousWakeTime, ↩
    portTickType xTimeIncrement) {
2   // Force a reschedule if xTaskResumeAll has not already done so, we may
3   //have put ourselves to sleep.
4   if (xAlreadyYielded == pdFALSE) {
5     /* ************************************************
6     ******** MODIFIED FOR POWER SAVING PURPOSES ********
7     ************************************************ */
8     flagNotTick = pdTRUE;

10    portYIELD_WITHIN_API();
11  }
12 }
```

Com o histórico de tarefas executadas, baseado na quantidade de 1's presentes, uma heurística determinará o novo estado energético do sistema quando não houverem tarefas na fila de execução. Como o sistema não possui primitivas para tratar bits diretamente, foi necessário utilizar um algoritmo eficiente de contagem de bits do CSB, sem causar impacto no processamento: o algoritmo de contagem paralela de bits foi selecionado por cumprir estes requisitos [64].

3.5.3 Sobrecarga da tarefa IDLE e implementação da política de economia de energia

Para realizar a contagem de bits, o sistema deve estar ocioso e avaliar o valor presente no CSB, sendo uma medida indireta da carga do sistema. Em outras palavras, se o sistema entrou em IDLE poucas vezes significa que passa a maior parte do tempo em processamento, então atrasos para economizar energia podem prejudicar a resposta esperada no tempo predeterminado.

Caso tenha uma grande quantidade de acessos à IDLE, significa que a maior parte do tempo está em "descanso", então atrasos para que se possa economizar energia não devem prejudicar o andamento do processamento. Esta afirmação é válida para aplicações do tipo *soft real-time*.

Com essas diretrizes, implementa-se a *vApplicationIdleHook()*. Note que o código 5 é um exemplo de aplicação, mas as decisões baseadas na quantidade de entradas em IDLE e qual o melhor modo de economia de energia é uma decisão que deve ser tomada pelo projetista.

Código 5 – Implementação da tarefa IDLE

```
1 void vApplicationIdleHook( void ){
2   uint16_t v = usContextSwitchBitmap; // count bits set in this (16-bit ↩
      value)
3   uint16_t c; // store the total here

5   c = v - ( ( v >> 1 ) & 0x5555 );
6   c = ( ( c >> 2 ) & 0x3333 ) + ( c & 0x3333 );
7   c = ( ( c >> 4 ) + c ) & 0x0F0F;
8   c = ( ( c >> 8 ) + c ) & 0x00FF;

10  // Turns On LPM LED debug
11  vPowerUpLed( pdTRUE, 2 );

13  switch( c ){
14    // For a full-load system, not a good idea to go to any LPM mode
15    case 0:
16      break;

18    // For a high-load system, wake-up time can be a problem for RT
19    case 1: /* Fall through case. */
20    case 2: /* Fall through case. */
21    case 3:
22      #ifdef __MSP430F5172__
23      __bis_SR_register( LPM1_bits + GIE );
24      #endif

26      #ifdef __CPU_MKL25Z128VLK4__
27      SMC_SetPowerModeWait(SMC);
28      #endif
29      break;

31    // Otherwise, it is worth saving energy
32    case 4: /* Fall through case. */
33    case 5: /* Fall through case. */
34    case 6: /* Fall through case. */
35    case 7: /* Fall through case. */
36    case 8: /* Fall through case. */
37    case 9: /* Fall through case. */
38    case 10:/* Fall through case. */
39    case 11:/* Fall through case. */
40    case 12:/* Fall through case. */
41    case 13:/* Fall through case. */
42    case 14:/* Fall through case. */
43    case 15:/* Fall through case. */
44    case 16:
45      #ifdef __MSP430F5172__
```

```
46        __bis_SR_register( LPM3_bits + GIE );
47        #endif

49        #ifdef __CPU_MKL25Z128VLK4__
50        SMC_SetPowerModeVlpr(SMC);
51        SMC_SetPowerModeVlpw(SMC);
52        SMC_SetPowerModeRun(SMC);
53        #endif
54        break;
55      default:
56        /* Unexpected value. Do nothing! */
57        break;
58    }
59  }
```

3.5.4 Implementação de tarefa-padrão para o sistema

Para todos os testes demonstrados durante este documento, foi implementada uma tarefa simples, que realiza a contagem de 0 a 3000. Ao final deste processo, a tarefa é posta em modo *Blocked* durante um período de *ticks* passado por parâmetro (variável *pvParameters*).

Na execução do sistema, três tipos de carga foram definidas:

- 1 tarefa, com período de *Blocked* igual a 1 *tick*
- 2 tarefas, todas com período de *Blocked* igual a 1 *tick*
- 3 tarefas, todas com período de *Blocked* igual a 3 *ticks*

Uma única tarefa foi implementada, conforme o código 6. A mudança no período bloqueado é realizada por parâmetro na criação da tarefa.

Código 6 – Implementação da tarefa de carga

```
1 void vTaskCode( void * pvParameters ){
2   TickType_t xLastWakeTime;
3   const TickType_t xFrequency = ( TickType_t ) pvParameters;

5   // Initialise the xLastWakeTime variable with the current time.
6   xLastWakeTime = xTaskGetTickCount();
7   volatile int i = 0;

9   for( ;; ){
10    /* The function __delay_cycles() messes up with the system determinism.
11    * Instead, use dummy operations, like incrementing. */
12    for( i = 0; i < 3000; i++);
13    if( xFrequency > 0 ){
14      vTaskDelayUntil( &xLastWakeTime, xFrequency );
15    }
16  }
17 }
```

4 Resultados

Nesta seção, serão comparados os tempos de *wake-up* e os valores de corrente em modos de baixo consumo fornecidos pelo fabricante e os obtidos pelas metodologias mostradas anteriormente. Com isso, segue uma análise sobre a carga máxima de processamento permitida para viabilizar economia de energia, assim como a eficiência da política adotada.

Primeiramente, a automação em software da corrente consumida pelas placas de teste foi um sucesso, pois permitiu a leitura contínua dos dados e seu log em planilhas. No entanto, a taxa de amostragem máxima do multímetro é de 995 pontos/segundo, não sendo efetiva para a observação criteriosa da execução entre *ticks* de 1ms, então optou-se por analisar somente os dados coletados pela SourceMeter.

4.1 Validação dos parâmetros dos microcontroladores

Conforme visto na referência bibliográfica as informações energéticas fornecidas pelo fabricante são uma referência para os projetistas, mas podem não representar a realidade. Deste modo o primeiro passo foi utilizar o sistema de aquisição para confirmar esses dados, bem como testar o próprio sistema.

Apesar de não haver menção sobre variabilidade nos tempos de *wake-up* nas referências estudadas, também optou-se por realizar esta análise para o microcontrolador MSP430F5172.

4.1.1 Tempo de *wake-up*

O primeiro teste realizado se baseou na instrumentação fornecida por dois terminais de I/O. Um monitorava a duração da troca de contexto e o segundo a execução da tarefa IDLE. A figura 15 foi obtida através de um osciloscópio e os números 1 à 4 adicionados posteriormente.

Os tempos marcados com o número "2" representam as várias execuções da tarefa IDLE. Os intervalos marcados com "1" e "3" apresentam a troca de contexto entre as diferentes tarefas do sistema. O tempo indicado por "4" indica a execução de uma tarefa padrão implementada com o método *vTaskCode()*. Com isso, é possível visualizar que, para cada troca de contexto, ou o sistema volta a executar a tarefa (marcação "4") ou entra em IDLE (marcação "2").

Para os testes, um mesmo sistema operando com uma mesma carga de processamento (ou seja, que possui um mesmo tempo de execução em IDLE) foi executado por

10 vezes e obtiveram-se valores coerentes de *wake-up*, tornando confiável a informação fornecida pelo fabricante.

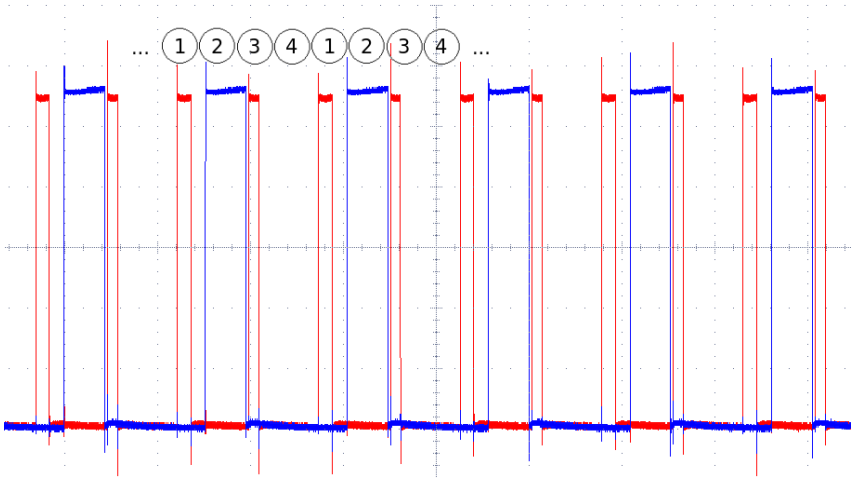


Figura 15 – Visualização das trocas de contexto e da tarefa IDLE no microcontrolador MSP430F5172.

Sendo assim, a diferença de tempo entre o final de "2" e o início de "3" fornece uma estimativa do *wake-up time*, sintetizado na tabela 5.

Modo de Low-Power	**Tempo em modo Idle**	**Tempo de Wake-up medido**	**Tempo de Wake-up fornecido pelo fabricante**
Active	$17,76 \pm 0,09\ \mu s$	-	-
LPM0	$17,79 \pm 0,12\ \mu s$	≈ 0	≈ 0
LPM1	$17,61 \pm 0,03\ \mu s$	≈ 0	≈ 0
LPM2	$24,64 \pm 1,93\ \mu s$	$6,88 \pm 1,93\ \mu s$	$6,5\ \mu s$
LPM3	$24,44 \pm 0,00\ \mu s$	$6,68 \pm 0,00\ \mu s$	$6,5\ \mu s$

Tabela 5 – Tempos de *wake-up* do MSP430F5172 estimados através das medições no osciloscópio

Com isso, é possível visualizar que os valores determinados por medição em osciloscópio estão coerentes com as informações fornecidas pelo *datasheet* do microcontrolador, com uma precisão de μs, mesmo considerando os atrasos entre o acionamento dos pinos de GPIO e o real comportamento da CPU em execução.

4.1.2 Consumo de energia

Checando os valores fornecidos pelo fabricante, de acordo com [27], não podemos afirmar que os dados de consumo de potência, energia ou corrente são coerentes com a aplicação-alvo executada pelo hardware. Desta forma, é importante que as medições estejam na mesma ordem de grandeza das informações descritas no *datasheet*.

Foram coletadas 10 amostras de cada modo energético, para ambas as placas de teste. Os gráficos a seguir são resultado da média aritmética de cada uma destas

medidas. O tempo total de aquisição foi 110ms para cada amostra, contendo 1000 pontos igualmente espaçados entre si.

A figura 16 apresenta as curvas de consumo de energia do microcontrolador MSP430 após a aplicação de energia. Nelas, tão logo o sistema seja inicializado, o microcontrolador é mantido nos modos Execução, Modo1 ou Modo2.

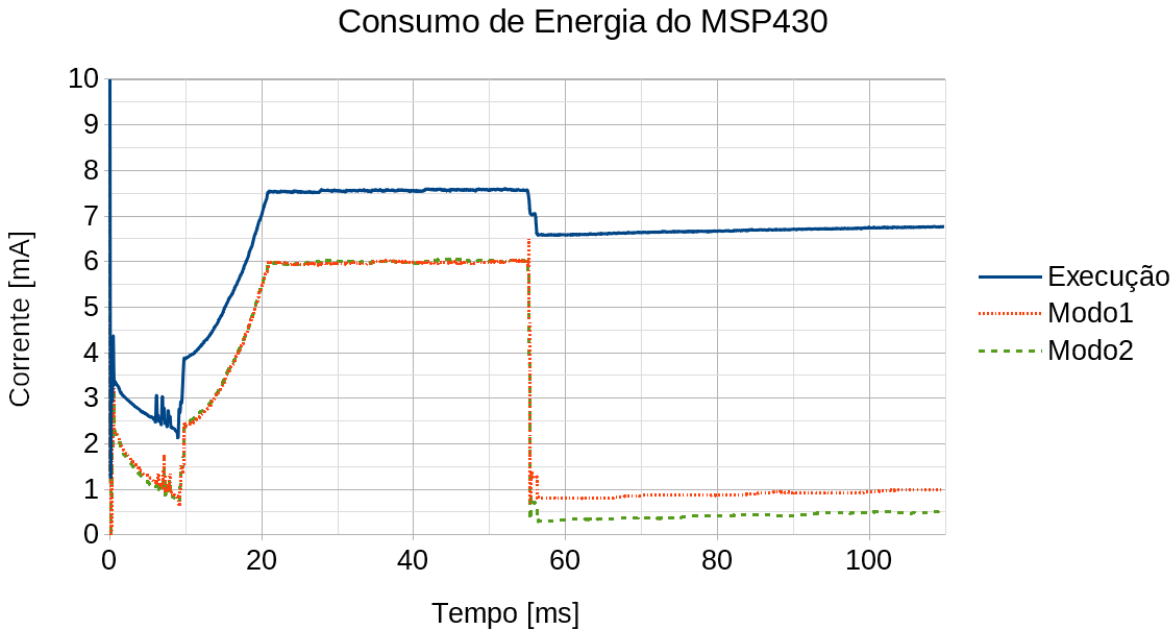


Figura 16 – Modos de Execução, Modo1 e Modo2 do microcontrolador MSP430F5172.

A figura 17 apresenta as curvas de consumo de energia do microcontrolador MKL25Z após a aplicação de energia. De modo análogo aos resultados obtidos com o microcontrolador MSP430, tão logo o sistema seja inicializado, o microcontrolador é mantido nos modos de Execução, Modo1 ou Modo2.

A tabela 6 apresenta o consumo médio de cada um dos três modos analisados de ambos os microcontroladores, excluindo-se o período de tempo necessário na inicialização, cerca de 55ms para o MSP430 e 10ms para o MLK25Z.

Microcontrolador	Modo de energia	Consumo médio (fabricante)	Consumo médio (obtido)
MSP430F5172	Execução (25 MHz)	$6,15$ mA	$6,68 \pm 0,06$ mA
	Modo1	não fornecido	$895,58 \pm 0,06$ µA
	Modo2	$1,5$ µA	$422,04 \pm 0,06$ µA
MKL25Z128VLK4	Execução (48 MHz)	$5,9$ mA	$2,04 \pm 0,00$ mA
	Modo1	$3,8$ mA	$1,56 \pm 0,00$ mA
	Modo2	366 µA	$1,56 \pm 0,00$ mA

Tabela 6 – Comparação entre os dados fornecidos pelo fabricante e os coletados.

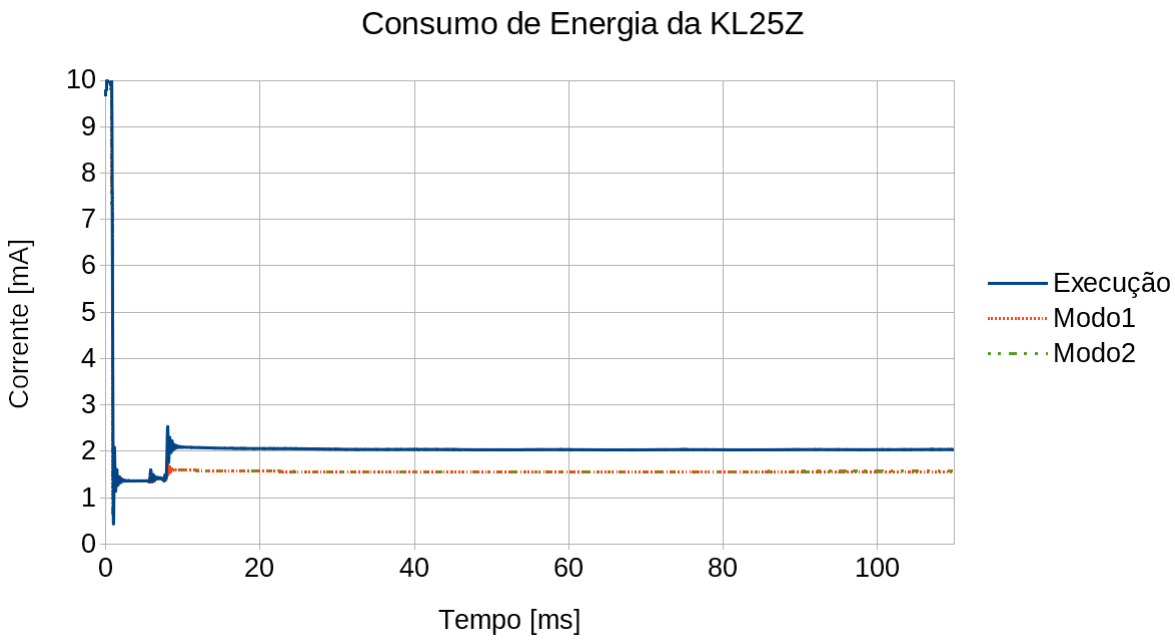


Figura 17 – Modos de Execução, Modo1 e Modo2 do microcontrolador MKL25Z128VLK4.

Conforme explicado anteriormente, o fabricante não especifica quais foram as condições de teste, e as leituras obtidas comprovam [27], pois as diferenças são significativas. Com isso, é importante notar que a tabela 6 se torna uma referência para o estabelecimento da Política P na equação 19 e obter o máximo de economia de energia possível no sistema-alvo, se recuperando do consumo acima do especificado.

Pelas curvas medidas em modos de economia de energia da placa FRDM-KL25Z, nota-se que o consumo de corrente é praticamente o mesmo. Isto se deve pelo consumo de outros periféricos presentes na placa que não puderam ser removidos no momento da medição, tal como a comunicação *OpenSDA* (utilizada na gravação de *firmware*).

Vale ressaltar que a velocidade de processamento das duas placas são diferentes, mas ajustados para seus máximos valores, e tal fato pode ser determinado com os diferentes tempos de estabilidade: a placa MSP430 demora aproximadamente 55ms para iniciar a execução das tarefas no FreeRTOS, enquanto que a FRDM-KL25Z necessita de cerca de 10ms. Processos anteriores ao escalonamento das tarefas são referentes à própria configuração e inicialização do hardware, processo conhecido como *Power On Reset* (POR). Sua análise não faz parte do escopo deste trabalho.

4.2 *Context Switch Bitmap* como estrutura de histórico de processamento

De acordo com o CSB, pode ser determinado qual o melhor modo de economia de energia. Entretanto, os valores adquiridos podem se tornar confusos caso não se atente à janela de tempo de execução, ou fatia de tempo (*time window*, *timeslice*).

Supondo que uma determinada tarefa A demore 1 *tick* e meio para ser executada, enquanto uma tarefa B demora aproximadamente 1 *tick*, porém ambas possuem a mesma implementação. A janela de tempo máxima reservada que uma tarefa possui para ser executada é de 1 *tick*. Isso significa que, se for necessário mais tempo, deve ser executada quando uma outra poderia ser processada completamente, ou seja, a janela de tempo da tarefa A deve ser maior.

Isso significa que não há uma periodicidade simples na entrada e saída do modo IDLE, mas sim composta. Um exemplo de situação pode ser visualizado na figura 18. Para este caso, nota-se que duas tarefas consomem 3 ticks.

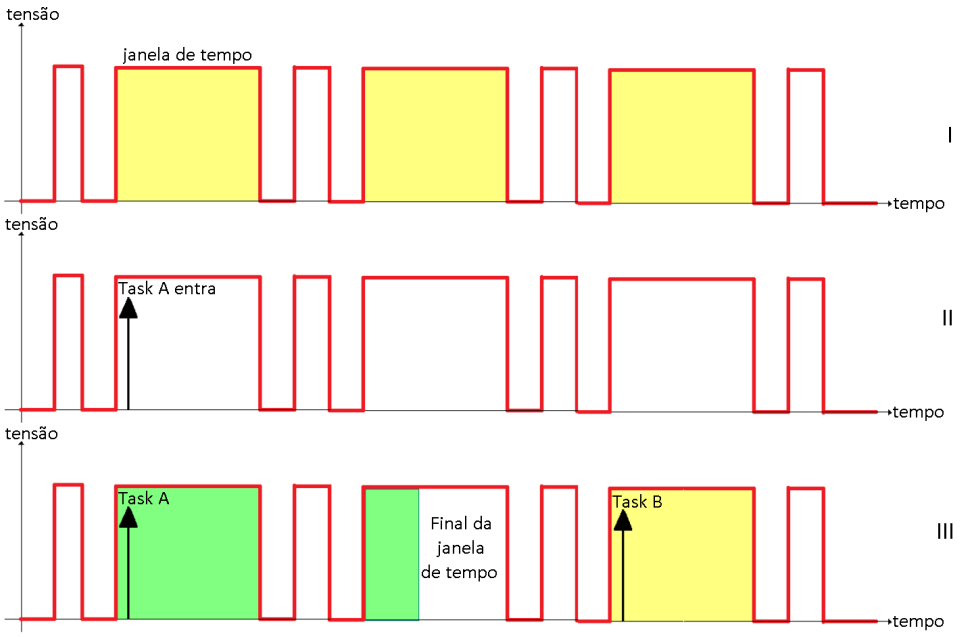


Figura 18 – Diferença entre slot de tempo e quantidade de tarefas executadas

Foram realizados vinte e cinco testes de execução de três tarefas com implementação idêntica e os valores do CSB coletados. Arbitrariamente 5 foram selecionados para serem exibidos na tabela 7, assim como o maior e menor valor de carga de processamento encontrados nesta bateria de teste.

Com relação aos resultados numéricos, conforme a equação 1, o limite máximo teórico da carga de processamento para um sistema com três tarefas deve ser de 77,98%, sendo esta uma condição necessária para que a técnica de cálculo de carga de processamento via CSB seja válida.

Número do Teste	CSB	Razão IDLE/Tarefas	Carga de processamento
1	0111111100000011	9/16	43,8%
2	1111000011111100	10/16	37,5%
3	0011111100001111	10/16	37,5%
4	0000111111000001	7/16	56,3%
5	1111000000111111	10/16	37,5%
Maior carga	0000001111110000	6/16	62,5%
Menor carga	1111111000011111	12/16	25%

Tabela 7 – Leituras em instantes arbitrários de tempo do mapa binário CSB, determinando a razão entre os acessos à tarefa IDLE com o total de execuções, assim como o cálculo da carga de processamento.

Conforme explicado, é esperado um determinismo e um padrão no processamento das tarefas, pois são as únicas a serem executadas no sistema, não há comunicação com periféricos externos e todas possuem a mesma implementação. De acordo com a Seção anterior, a carga de processamento foi de 3 tarefas, todas com período de *Blocked* igual a 3 *ticks*.

No entanto, o próprio CSB demonstra que as tarefas utilizam diferentes tamanhos de janela de tempo. Como todas executam as mesmas instruções, era esperado que precisassem de uma mesma quantidade de tempo para finalizar seu processamento, logo a sequência de bits 0 (representando um *tick* voltado para as tarefas) entre duas sequências de bits 1 seria um múltiplo de 3.

Desta forma, como o tempo necessário para o sistema executar uma tarefa não é fixo, percebe-se que o RTOS pode oferecer determinismo em software, mas que o mesmo é limitado pelo microcontrolador.

4.3 Politica de economia de energia

Conforme sinalizado por [39], a corrente consumida ao se retornar de um modo de baixo consumo para a execução das tarefas do sistema é superior ao valor de execução contínua. De acordo com a equação 17, a energia drenada pelo sistema está diretamente relacionada à carga de processamento.

Desta forma, foram avaliadas diferentes cargas, assim como diferentes políticas, e seus resultados comparados. A fonte de alimentação fornece 3,3V, e a frequência de *tick* do SO vale 1kHz.

Primeiramente, na figura 19 não é aplicada política alguma de DPM no MSP430, de forma que aguarda em IDLE a entrada de novas tarefas na fila de execução do sistema. Em outras palavras, nenhuma ação é executada quando há a oportunidade

de se economizar energia.

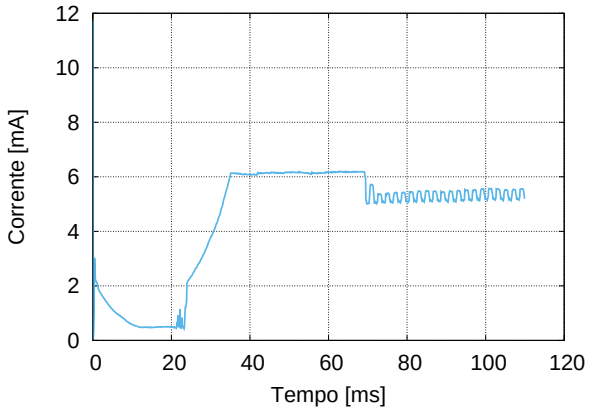


Figura 19 – 1 tarefa em processamento na placa MSP430, sem uma política de DPM.

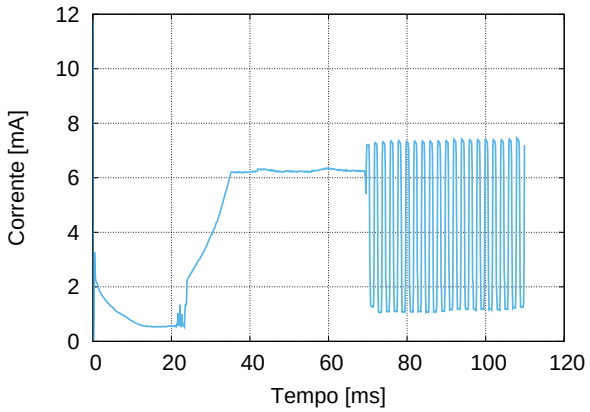


Figura 20 – 1 tarefa em processamento na placa MSP430, sob uma política de Modo1.

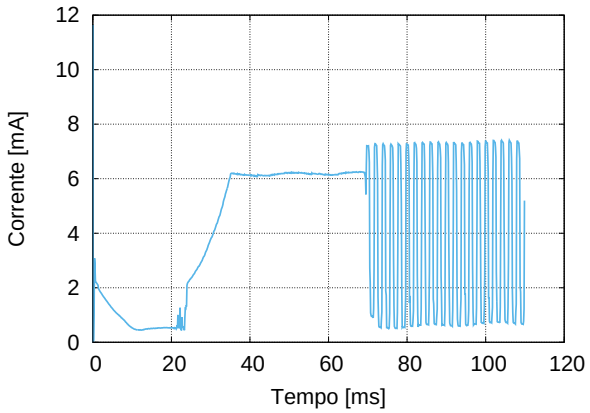


Figura 21 – 1 tarefa em processamento na placa MSP430, sob uma política de Modo2.

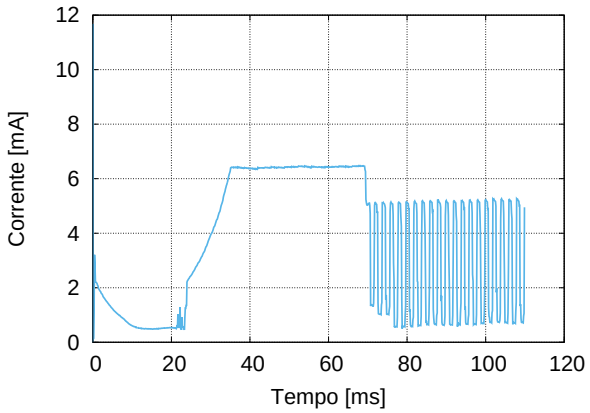


Figura 22 – 1 tarefa em processamento na placa MSP430, sob uma política de DPM de acordo com os valores de CSB.

Já nas figuras 20 e 21, as políticas utilizadas são estáticas, ou seja, quando há a oportunidade de se economizar energia, utiliza-se o Modo1 ou Modo2, respectivamente. Além disso, é comprovado o cenário de consumos de corrente descritos na equação 17 como i_{ON2OFF} e i_{OFF}.

Quando se aplica a política de DPM, a economia se torna mais efetiva, pois buscará o modo mais econômico de acordo com o histórico de execução armazenado no CSB, e tal fato pode ser visualizado pela escolha dos modos de baixo consumo se adaptarem conforme o tempo passa (entre 70 e 80 ms). Apesar de ocorrer em outros casos testados, a elevação súbita da corrente para valores acima do estabelecido pelo fabricante (*DPD overhead*) não ocorreu na média das 10 medições realizadas para o gráfico da figura 22.

A tabela 8 apresenta o consumo médio em cada um dos cenários para o MSP430. Os valores entre parênteses representam a economia no consumo de corrente ao se adotar uma política de avaliação dinâmica em comparação com o cenário proposto em cada coluna.

Política → Carga de processamento ↓	Mantém o modo de Execução	Modo1 para baixo consumo	Modo2 para baixo consumo	Política P avalia o melhor modo energético
1 tarefa + IDLE	$5,525 \pm 0,602$ mA	$4,532 \pm 1,231$ mA (17,97%)	$4,298 \pm 1,317$ mA (22,21%)	$3,736 \pm 0,425$ mA (32,39%)
2 tarefas + IDLE	$5,499 \pm 0,400$ mA	$4,098 \pm 2,655$ mA (25,47%)	$3,867 \pm 2,843$ mA (29,67%)	$3,421 \pm 2,366$ mA (37,79%)
3 tarefas + IDLE	$5,550 \pm 0,445$ mA	$6,286 \pm 2,308$ mA ($-13,24$%)	$6,238 \pm 2,489$ mA ($-12,37$%)	$5,490 \pm 2,396$ mA (1,09%)

Tabela 8 – Consumo médio de corrente da placa MSP430, de acordo com a carga de processamento e a política praticada

De forma análoga, as figuras a seguir apresentam as curvas de consumo de energia para as diferentes políticas na placa FRDM-KL25Z, para uma carga de processamento

fixa (2 tarefas de implementação idênticas), de forma que a figura 23 mostra a execução de um sistema o qual não aplica uma política de DPM, a figura 24 representa o processamento do sistema aplicando uma política estática de Modo1, e por último, a figura 25 exibe a aplicação da política de DPM de acordo com os valores de CSB.

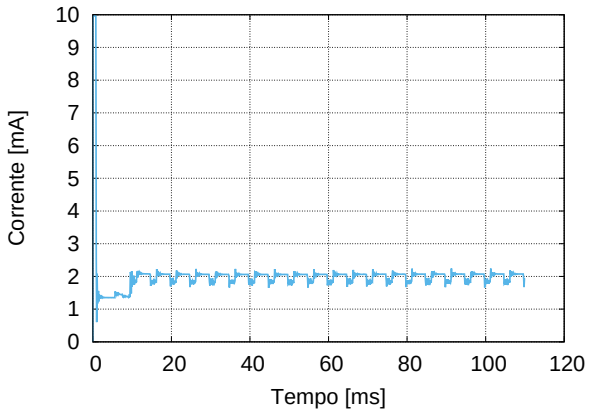


Figura 23 – 2 tarefas idênticas em processamento na placa FRDM-KL25Z, sem uma política de DPM.

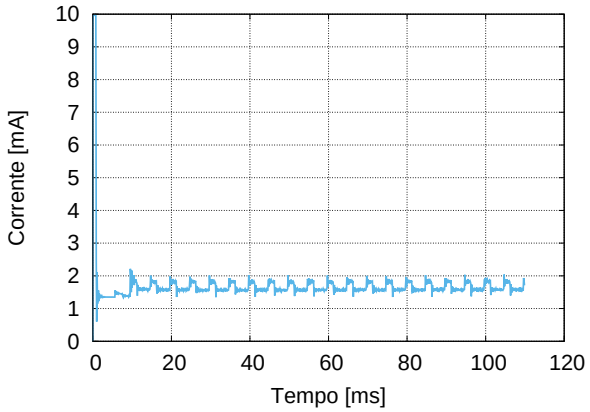


Figura 24 – 2 tarefas idênticas em processamento na placa FRDM-KL25Z, sob uma política de Modo1.

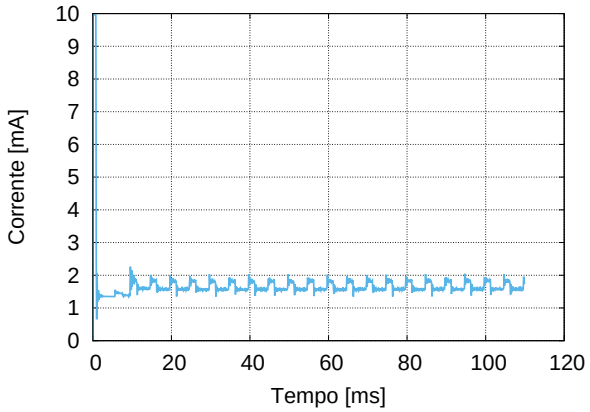


Figura 25 – 2 tarefas idênticas em processamento na placa FRDM-KL25Z, sob uma política de DPM de acordo com os valores de CSB.

Para a placa FRDM-KL25Z nota-se que, durante uma parcela significativa do tempo, o núcleo se mostra em processamento. A quantidade de oscilações de estados energéticos se torna mais intensa dependendo de quantas tarefas estão em fila para execução e qual sua periodicidade. Ainda assim, é possível notar, através da tabela 9, que a adoção da política dinâmica P, no pior caso, gera uma economia idêntica à utilização de um modo fixo de baixo consumo.

Política → Carga de processamento ↓	Mantém o modo de Execução	Modo1 para baixo consumo	Modo2 para baixo consumo	Política P avalia o melhor modo energético
1 tarefa + IDLE	$1,975 \pm 0,128$ mA	$1,679 \pm 0,134$ mA (14,99%)	$1,679 \pm 0,134$ mA (14,99%)	$1,676 \pm 0,134$ mA (15,14%)
2 tarefas + IDLE	$1,996 \pm 0,144$ mA	$1,651 \pm 0,158$ mA (17,28%)	$1,653 \pm 0,163$ mA (17,18%)	$1,652 \pm 0,162$ mA (17,23%)
3 tarefas + IDLE	$1,880 \pm 0,014$ mA	$1,781 \pm 0,031$ mA (5,27%)	$1,781 \pm 0,033$ mA (5,27%)	$1,782 \pm 0,009$ mA (5,21%)

Tabela 9 – Consumo médio de corrente da placa FRDM-KL25Z, de acordo com a carga de processamento e a política praticada.

De modo geral, utilizar uma política dinâmica de DPM baseada no histórico de processamento, oferece uma economia igual ou superior à uma política estática. Não havendo tarefas classificadas como *hard real-time*, a própria política também pode aumentar o determinismo por buscar modos energéticos menos agressivos quando momentaneamente o sistema atingir uma alta carga de processamento. Esses modos, em geral, possuem baixos tempos de *wake-up*.

Retomando as situações na qual ocorre *DPD overhead*, de acordo com a equação 17, o consumo médio do sistema é dependente do modo utilizado e da carga de processamento do sistema. É possível inferir um ponto teórico de operação C_{MAX}, para o qual o sistema, com uso do Modo1 ou Modo2, consuma a mesma quantidade de energia do que o sistema permanentemente ligado. Isto pode ser feito igualando i_{LPM} à i_{ON}, obtendo-se a equação 19.

$$C_{MAX} = \frac{i_{ON} - i_{OFF}}{i_{ON2OFF} - i_{OFF}} \tag{19}$$

Por outro lado, o ponto de operação mínimo seria uma carga de processamento nula, que geraria um consumo de corrente mínimo (potência de dissipação estática e *glitch*), equivalente a i_{OFF}.

Com as cargas de processamento mínima e máxima e os respectivos consumos de corrente, é possível traçar uma curva contendo a relação linear entre consumo de corrente e carga de processamento. Utilizando-se os parâmetros levantados para

o microcontrolador MSP430F5172 podemos obter essa relação conforme apresentado na figura 26.

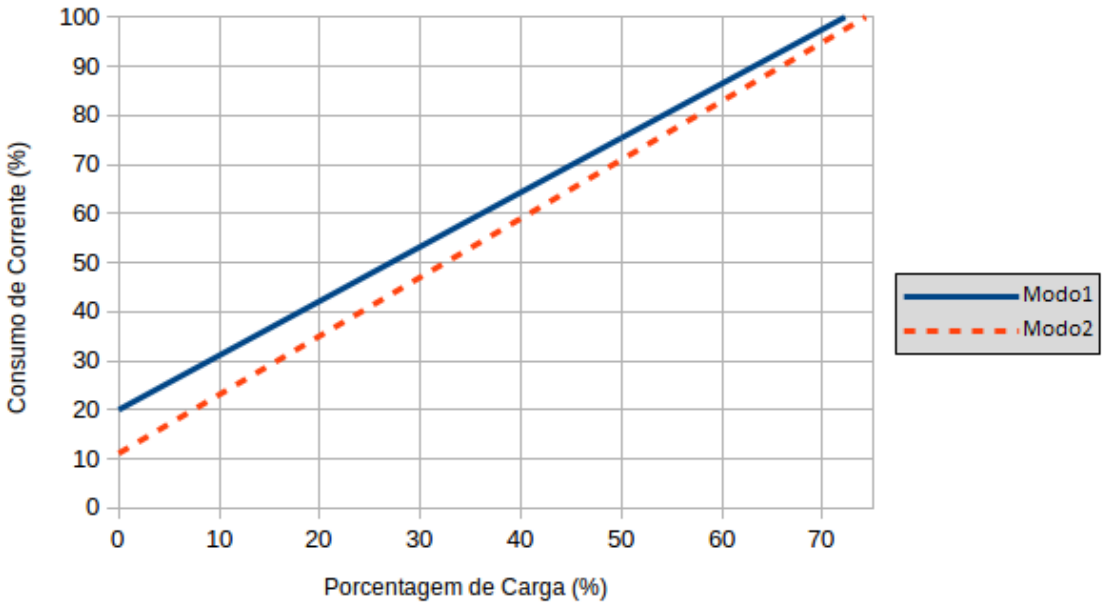


Figura 26 – Relação entre carga de processamento e a variação no consumo de corrente, comparado com o sistema utilizando o Modo1 e Modo2.

Pode-se notar que um sistema utilizando Modo1 como método de economia de energia, dentro tarefa IDLE, é capaz de reduzir o consumo em até 80%, desde que não realize nenhuma tarefa útil ($C_{\%} = 0$). Para Modo2, a diminuição de corrente chega a 89%. Para valores superiores a C_{MAX}, não é efetiva a aplicação da política proposta P.

Observando o aspecto de carga máxima de processamento, em Modo1, o valor de C_{MAX} é $72,3\%$, ou seja, apesar de consumir a mesma quantidade de energia que um sistema permanentemente ativo, ele está limitado a $72,3\%$ da capacidade de processamento. Se o sistema necessitar de maior capacidade de processamento, os ganhos com a economia de energia gerados pelo Modo1 são anulados. Para o Modo2, o valor crítico C_{MAX} é dado por $74,3\%$ de capacidade de processamento. Estes valores se mostram coerentes, pois estão abaixo do limite definido pela equação 1. Para a placa FRDM-KL25Z, como não foi observada a ocorrência de *DPD overhead*, espera-se que a carga máxima de processamento seja 100%, visto que $i_{ON2OFF} = i_{ON}$.

5 Conclusões

Tendo em vista uma ascensão de tecnologias que necessitam minimizar seu consumo de energia sem interferir na capacidade de processamento, uma alternativa prática é delinear uma estratégia de economia no próprio Sistema Operacional para atingir ambos objetivos. Estes estudos focaram em analisar técnicas de redução de consumo de energia em RTOS, de forma a apresentar uma variação da técnica DPM.

Até a presente data, não foram encontrados outros trabalhos que realizassem políticas similares de economia de energia, pois nenhuma outra metodologia referenciada buscou balancear determinismo com economia de corrente. A aplicação desta técnica gerou diminuição no consumo de energia de até 37% em alguns casos.

Desta forma, o projetista de uma aplicação embarcada deve buscar analisar os valores reais de consumo da CPU, referenciados como i_{ON}, i_{OFF} e i_{ON2OFF}, sem deixar de avaliar os tempos de *wake-up*.

Assim como demonstrado, é importante buscar isolar a influência de periféricos da unidade sob teste, assim como módulos internos do MCU em estudo.

Em termos de software, para a implantação da política proposta, o sistema-alvo necessita de poucas adaptações, sendo a principal delas a criação de uma gerência de consumo de energia, representada neste trabalho pela tarefa IDLE no FreeRTOS. Outras modificações a nível de *kernel* se mostram necessárias para coletar dados da carga de processamento, essencial para que a política de economia de corrente tome a decisão baseada no histórico de tarefas.

Em relação ao CSB, deve-se buscar um tamanho ideal para que se tenha um histórico significativo das tarefas executadas, mas que não seja tão grande a ponto de gerar sobrecarga na somatória de bits do mapa binário. No entanto, para ambas as arquiteturas testadas, o tamanho de 16 bits se mostrou eficiente.

Conclui-se que não se deve analisar ou desenvolver uma política de gestão energética se baseando somente nos modos de economia de energia do processador, sendo esta a maior contribuição deste estudo, sintetizado pelas equações 17 e 19. É necessário realizar uma análise conjunta da questão energética com a capacidade de processamento. Esta análise deve ser pautada em medições reais de consumo, visto que as informações fornecidas pelos fabricantes podem não contemplar todas as características do sistema analisado, inclusive se o mesmo é suscetível a *DPD overhead* (as flutuações nos valores de corrente podem ser inerentes da própria tecnologia do chip, ou influenciada por fatores como manuseio ou qualidade de fabricação das placas de teste [65]).

Além das questões abordadas, o modo de economia de energia gera um atraso temporal cada vez que é ativado ou desativado. Esta limitação temporal, aliado à

carga máxima de processamento (C_{MAX}), pode reduzir ainda mais a quantidade de processamento disponível para execução de tarefas.

Estes estudos ainda produziram um artigo técnico apresentado no Congresso Brasileiro de Automática de 2016 (CBA2016) e um pedido de patente no Instituto Nacional da Propriedade Industrial (INPI) em 2015. Ambos apresentados no apêndice deste documento.

Para trabalhos futuros, é interessante analisar em quais cenários ou para quais tecnologias ocorrem *DPD overhead*, analisando diferentes modelos de microcontroladores, inclusive em cenários de microchips *multicore*. Outra abordagem seria desenvolver uma política que utilizasse como critério de decisão dos modos de economia de energia as cargas máximas de processamento.

Também pode-se estudar a otimização do tamanho do CSB para diferentes casos de uso, determinando regras de recomendação de seu tamanho para aplicações *CPU-bound* e *memory-bound*.

Os autores propõe-se também a avaliar a influência de outros valores de *tick* do sistema, que podem significativamente impactar na performance: se a janela de tempo dada a cada tarefa for maior, o sistema pode permanecer mais tempo em IDLE. Mesmo com um *tick* ideal, o sistema ainda está pouco otimizado para tarefas de alto processamento. Por exemplo, suponha haver uma câmera em constante movimento conectada a um microprocessador/microcontrolador, sendo abstraída sob a forma de uma tarefa. Se a câmera necessita captar movimentos sob demanda, o atraso gerado pela rotina de tratamento de interrupção do *timer* de *tick* pode prejudicar e não permitir uma devida leitura dos dados enviados. O sistema poderia ser otimizado para lidar com tais situações, pois tarefas deste tipo requereriam alta disponibilidade do núcleo de processamento, porém, quando desativadas, espera-se alta economia de energia. Sendo assim, determinar o valor ideal de *tick* para maximizar a economia de energia, assim como o determinismo em tempo de projeto, mostra-se essencial pois modificar dinamicamente seu valor em tempo de execução pode causar um impacto considerável no determinismo modelado pelo projetista.

Um tópico não avaliado foi a responsividade do sistema. Este é um importante critério para projetos práticos que abordem entrada e saída de dados. Os dispositivos ou usuários em comunicação com o sistema embarcado possuem uma expectativa do tempo de resposta entre o envio de um comando até a recepção dos dados de resposta (seja pressionar uma tela *touch screen* e modificar a imagem na tela, ou transmitir um comando de leitura de dados via conexão Serial e receber as informações desejadas em um intervalo de tempo aceitável – *timeout*).

Uma outra abordagem a ser tomada seria combinar diversas técnicas estudadas com a variação de DPM proposta: utilizar DVFS modifica consideravelmente o determinismo e os ganhos gerados pela política proposta nesta dissertação, mas sua união

possui o potencial de reduzir o consumo significativamente. Tal análise deve ser norteada pelas recomendações de [66], que enfatiza que a implementação de sistemas embarcados mais econômica em consumo de energia é dependente simultaneamente de processador e RTOS. Sendo assim, sugere: reescrever porções com maior consumo de energia de uma aplicação para evitar uso desnecessário do escalonador; quando sincronização entre tarefas se mostra implícita, não utilizar recursos do RTOS para gerar sincronia redundante (apesar da redundância aumentar a robustez do código); usar primitivas do RTOS, aumentando sua portabilidade e baixa sobrecarga; caso análises indiquem que a gerência de memória do RTOS consome muita energia, considerar fazer seu próprio gerenciador; usar modos especiais disponíveis no processador (tais como modos de baixo consumo).

Referências

[1] William Stallings. *Arquitetura e organização de computadores*. Pearson Education do Brasil, 2010.

[2] Real Time Engineers. Tasks. `http://www.freertos.org/tskstate.gif`, 2013.

[3] Freescale Semiconductor Incorporated. *FRDM-KL25Z Users Manual*, September 2012. Rev. 1.0.

[4] Erich Styger. Tutorial: Using the frdm-kl25z as low power board. `https://mcuoneclipse.com/2013/10/20/tutorial-using-the-frdm-kl25z-as-low-power-board/`, 2013.

[5] Texas Instruments Incorporated. *MSP430F51x1, MSP430F51x2 Digital Signal Microcontroller Datasheet*, August 2010. Revised May 2016.

[6] Freescale Semiconductor Incorporated. *Data Sheet: Technical Data. Kinetis KL25 Sub-Family*, August 2014. Rev 5.

[7] Sangeet Saha, Arnab Sarkar, and Amlan Chakrabarti. Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms. *IEEE Embedded Systems Letters*, 7(1):23–26, 2015.

[8] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of energy-cognizant scheduling techniques. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1447–1464, 2013.

[9] Real Time Engineers. Freertos. `http://www.freertos.org`, agosto 2016.

[10] William Wulf, Ellis Cohen, William Corwin, Anita Jones, Roy Levin, Charles Pierson, and Fred Pollack. Hydra: The kernel of a multiprocessor operating system. *Communications of the ACM*, 17(6):337–345, 1974.

[11] Alan Burns, Ken Tindell, and Andy Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480, 1995.

[12] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*. J. Wiley & Sons, 2009.

[13] Suresh Siddha and Venkatesh Pallipadi. Getting maximum mileage out of tickless. In *Linux Symposium*, volume 2, pages 201–207, 2007.

[14] Joseph Yiu and Andrew Frame. Arm cortex-m3 processor software development for arm7tdmi processor programmers. In *White Paper ARM*. ARM Limited, 2009.

[15] Bill Gallmeister. *POSIX. 4 Programmers Guide: Programming for the real world*. "O'Reilly Media, Inc.", 1995.

[16] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[17] Michael Christofferson. 4 ways to improve linux performance. Technical Report 2, Enea Software, Jan Stenbecks Torg 17 P.O. Box 1033 Kista, SE-164 21 Sweden, July 2013. IEEE/ENEA Webinar.

[18] Marin Litoiu and Roberto Tadei. Real-time task scheduling with fuzzy deadlines and processing times. *Fuzzy Sets and Systems*, 117(1):35–45, 2001.

[19] Real Time Engineers. Coding standard and style guide. `http://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html`, 2013.

[20] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, 1994.

[21] Thatyana Seraphim Rodrigo Almeida, Carlos Moraes. *ProgramaÃ§Ã£o de Sistemas Embarcados: Desenvolvendo software para microcontroladores em linguagem C*. Elsevier Editora Ltda, 2016.

[22] Christopher Hallinan. *Embedded Linux primer: a practical, real-world approach*. Pearson Education India, 2007.

[23] P Marwedel. *Embedded System Design*. Springer, 2006.

[24] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–12. IEEE, 2013.

[25] Mehran Mozaffari Kermani, Meng Zhang, Anand Raghunathan, and Niraj K. Jha. Emerging frontiers in embedded security. In *Proceedings of the 2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, VLSID '13, pages 203–208, Washington, DC, USA, 2013. IEEE Computer Society.

[26] Anantha P Chandrakasan, Samuel Sheng, and Robert W Brodersen. Low-power cmos digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[27] Hueliquis Fernandes. The true low power concept - implementing powerful embedded controls with minimum energy requirements. Technical Report 1, Renesas Electronics America Inc., TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan, August 2013. ESC Brazil 2013 Technical Insights.

[28] Embedded Microprocessor Benchmark Consortium. About eembc. `http://www.eembc.org/about/index.php`, 2016.

[29] Embedded Microprocessor Benchmark Consortium. Ulpbench (tm) scores. `http://www.eembc.org/ulpbench/`, 2016.

[30] Freescale Semiconductor Incorporated. *KL25 Sub-Family Reference Manual*, September 2012. Rev. 3.

[31] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382. IEEE, 1995.

[32] Hideki Takase, Gang Zeng, Lovic Gauthier, Hirotaka Kawashima, Noritoshi Atsumi, Tomohiro Tatematsu, Yoshitake Kobayashi, Shunitsu Kohara, Takenori Koshiro, Tohru Ishihara, et al. An integrated optimization framework for reducing the energy consumption of embedded real-time applications. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 271–276. IEEE, 2011.

[33] Vadim Gutnik and Anantha P Chandrakasan. Embedded power supply for low-power dsp. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4):425–435, 1997.

[34] Kathleen Baynes, Chris Collins, Eric Fiterman, Brinda Ganesh, Paul Kohout, Christine Smit, Tiebing Zhang, and Bruce Jacob. The performance and energy consumption of embedded real-time operating systems. *IEEE Transactions on Computers*, 52(11):1454–1469, 2003.

[35] William P McCartney and Nigamanth Sridhar. Stackless multi-threading for embedded systems. *IEEE Transactions on Computers*, 64(10):2940–2952, 2015.

[36] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P Pipe, Thomas F Wenisch, and Milo MK Martin. Utilizing dark silicon to save energy with computational sprinting. *IEEE Micro*, 33(5):20–28, 2013.

[37] Pavan Kumar and Mani Srivastava. Predictive strategies for low-power rtos scheduling. In *Computer Design, 2000. Proceedings. 2000 International Conference on*, pages 343–348. IEEE, 2000.

[38] Giuseppe Tagliavini, Davide Rossi, Andrea Marongiu, and Luca Benini. Synergistic hw/sw approximation techniques for ultra-low-power parallel computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[39] Yashwant Singh, Mayank Popli, and Shiv Shankar Prasad Shukla. Energy reduction in weakly hard real time systems. In *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*, pages 909–915. IEEE, 2012.

[40] Thomas D Burd, Trevor A Pering, Anthony J Stratakos, and Robert W Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of solid-state circuits*, 35(11):1571–1580, 2000.

[41] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

[42] Marcus Völp, Marcus Hähnel, and Adam Lackorzynski. Has energy surpassed timeliness? scheduling energy-constrained mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 275–284. IEEE, 2014.

[43] Alireza Ejlali, Bashir M Al-Hashimi, Marcus T Schmitz, Paul Rosinger, and Seyed Ghassem Miremadi. Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(4):323–335, 2006.

[44] Yongqi Ge, Yunwei Dong, and Hongbing Zhao. Energy-efficient task scheduling and task energy consumption analysis for real-time embedded systems. In *Theoretical Aspects of Software Engineering Conference (TASE), 2014*, pages 135–138. IEEE, 2014.

[45] Ms Shraddha S Nakate, Bandu B Meshram, and Mrs Jayamala P Chavan. New trends in real time operating systems. *system*, 2(4):883–892, 2012.

[46] Wan Y Lee. Stochastically power-minimum scheduling of real-time multicore systems with leakage power awareness. *Electronics Letters*, 49(13):791–793, 2013.

[47] Ajoy K Datta and Rajesh Patel. Cpu scheduling for power/energy management on multicore processors using cache miss and context switch data. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1190–1199, 2014.

[48] Chuting Yao, Chenyang Yang, and Zixiang Xiong. Energy-saving predictive resource planning and allocation. *IEEE Transactions on Communications*, 64(12):5078–5095, 2016.

[49] Rami Melhem, Daniel Mossé, and Elmootazbellah Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Transactions on Computers*, 53(2):217–231, 2004.

[50] Alireza Ejlali, Bashir M Al-Hashimi, and Petru Eles. Low-energy standby-sparing for hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):329–342, 2012.

[51] Keysight Technologies Incorporated. *Keysight E364xA Dual Output DC Power Supplies. Users and Service Guide*, November 2014. Edition 11.

[52] Tektronix Incorporated. *DMM4040 and DMM4050 Digital Multimeter Users Manual*, August 2014.

[53] Tektronix Incorporated. *DMM4040 and DMM4050 Digital Multimeter Programmer Manual*, October 2009.

[54] Microsoft Corporation. Visual studio ide. `https://www.visualstudio.com/vs/`, 2016.

[55] Microsoft Corporation. Serialport.datareceived event. `https://msdn.microsoft.com/en-us/library/system.io.ports.serialport.datareceived.aspx`, 2016.

[56] Sparx Engineering. If you *must* use .net system.io.ports.serialport. `http://www.sparxeng.com/blog/software/must-use-net-system-io-ports-serialport`, 2014.

[57] Adam Kewley. Extension methods to make system.io.ports.serialport easier to use with .net 4.5 async workflows (does not support timeouts). `https://gist.github.com/AdamK117/c607e872dbd260d15c63`, 2016.

[58] IVI Foundation. Scpi consortium. `http://www.ivifoundation.org/scpi/default.aspx`, 2013.

[59] Keithley Instruments. *Series 2600A System SourceMeter Users Manual*, May 2006. Rev. A.

[60] Keithley Instruments. *Series 2600A System SourceMeter Reference Manual*, January 2010. Rev. D.

[61] IAR Systems. Iar embedded workbench for msp430. `https://www.iar.com/iar-embedded-workbench/#!?architecture=MSP430&currentTab=features`, 2013.

[62] NXP Semiconductors V. L. Kds ide: Kinetis design studio integrated development environment (ide). http://www.nxp.com/products/software-and-tools/software-development-tools/kinetis-design-studio-integrated-development-environment-ide:KDS_IDE, 2016.

[63] Texas Instruments Incorporated. Gcc - open source compiler for msp microcontrollers. http://www.ti.com/tool/msp430-gcc-opensource, 2016.

[64] Sean Eron Anderson. Bit twiddling hacks - counting bits set, in parallel. http://graphics.stanford.edu/~seander/bithacks.html, 2005.

[65] Panasonic Corporation. Quality and reliability information. http://www.semicon.panasonic.co.jp/en/aboutus/reliability.html, 2012.

[66] Robert P Dick, Ganesh Lakshminarayana, Anand Raghunathan, and Niraj K Jha. Analysis of power dissipation in embedded systems using real-time operating systems. *IEEE transactions on computer-aided design of integrated circuits and systems*, 22(5):615–627, 2003.

6 Apêndices

6.1 Artigo apresentado no CBA2016

IMPLEMENTAÇÃO DE UMA POLÍTICA DE GESTÃO ENERGÉTICA NO FREERTOS

CÉSAR A. M. DOS SANTOS, RODRIGO M. A. DE ALMEIDA, CARLOS H.V. MORAES

Grupo de Engenharia Biomédica, IESTI, Universidade Federal de Itajubá
Rua Coronel Rennó, número 7, Bairro Centro, Itajubá – MG, CEP 37500-050
E-mails: cesaraugusto@unifei.edu.br, rodrigomax@unifei.edu.br, valerio@unifei.edu.br

Abstract— This article demonstrates different power saving modes applied to real time operating systems, whenever no tasks are being executed. It is expected that low power modes supported by the microcontroller diminishes energy consumption, proportional to the number of internal modules deactivated. On the other hand, this economy can become invalid based on system processing load, because instant consumed current after changing from a low power state to active mode is higher than a same system under continuous operation. This way, a power saving policy will be presented as well as how to apply such, simultaneously minimizing energy consumption and balancing it with load processing.

Keywords— Real time systems, Embedded systems, Microcontrollers, Energy saving

Resumo— Este artigo demonstra a utilização de diferentes modos de economia de energia aplicados em sistemas operacionais de tempo real, quando o mesmo não possui tarefas a serem executadas. Espera-se que, ao entrar em modos de baixo consumo oferecidos pelo próprio microcontrolador, haja uma diminuição no consumo de energia, proporcional ao número de módulos internos que são desativados. Entretanto, a economia pode não se mostrar válida de acordo com a carga de processamento que o sistema possui, pois a corrente instantânea consumida após a transição do estado de baixo consumo para o modo ativo é superior ao consumo do sistema em operação constante. Desta forma, será apresentada uma política de economia de energia e como aplicá-la, simultaneamente minimizando o consumo energética e balanceando a exigência de processamento.

Palavras-chave— Sistemas de tempo real, Sistemas embarcados, Microcontroladores, Economia de energia

1 Introdução

Sistemas operacionais de tempo real (RTOS - *Real Time Operating Systems*) são uma forma de abstração utilizadas na programação de sistemas embarcados, para simplificar e facilitar a implementação de atividades com requisitos de tempo real, os quais possuem *kernels* capazes de atender demandas temporais. Em geral, um RTOS consome recursos de processamento para gerenciar uma lista de tarefas, realizando comparações entre o intervalo de reexecução definidos por cada tarefa e um relógio interno (*tick*) do próprio sistema.

A programação destes sistemas ainda exige uma série de cuidados distintos não existentes em programas voltados para *desktop*, geralmente por restrições de memória, processamento, dimensão, peso, rigidez, entre outros (Juang et al., 2002). Uma das características importantes em sistemas embarcados é o consumo de energia, visto que diversos sistemas são desenvolvidos para operar sob baterias.

Diversos trabalhos já propuseram soluções para determinar e resolver este impasse, seja ele por sistemas reconfiguráveis, que podem combinar a flexibilidade de um processador de propósito geral e a eficiência de um hardware dedicado através de tecnologia FPGA (Saha, Sarkar, Chakrabarti, 2015) ou através de técnicas que combinam software e hardware, tais como ajuste dinâmico de tensão e frequência de operação (DVFS - *Dynamic Voltage-Frequency Scaling*) ou gerenciamento dinâmico de energia (DPM - *Dynamic Power Management*) (Zhuralev et al., 2013).

Entretanto, ao se trabalhar com técnicas de economia de energia baseadas em modos de baixo consumo do microcontrolador, ocorre um aumento repentino de corrente ao retornar ao seu modo ativo, podendo anular a economia prevista.

Esta análise foi implementada no sistema operacional de código aberto FreeRTOS (FreeRTOS, 2016a). O artigo apresentará uma política de gestão energética, que define limites de carga de processamento para o consumo de corrente desejado. Foi utilizada uma placa com o microcontrolador MSP430F5172 como plataforma de testes e obtenção de resultados práticos.

2 Desenvolvimento

A Texas Instruments (T.I., 2015) define cinco modos de baixo consumo de energia (*low power mode* – LPM), além do modo ativo (*active*). Cada LPM desabilita uma certa quantidade de periféricos no sistema.

Para este trabalho, o modo LPM4 foi descartado por não possuir nenhum *timer* em funcionamento, requisito exigido pelo FreeRTOS. Os modos LPM0 e LPM2 também não foram analisados, visto que possuem tempos de *wake-up* similares, aos modos LPM1 e LPM3, respectivamente. Este tempo é importante, pois pode reduzir a capacidade de processamento do sistema. Os modos LPM1 e LPM3 foram escolhidos, em detrimento aos outros dois, por serem mais econômicos.

O cenário mencionado de aumento de consumo de corrente é demonstrado na Figura 1: um sistema,

após seu processo de inicialização, também chamado de POR (*Power On Reset*), executa suas tarefas, e quando a fila de execução está vazia, diz-se que o sistema está ocioso. Então, o mesmo pode aguardar para executar a tarefa assim que a mesma estiver disponível (Figura 1a), ou pode colocar seu microcontrolador para entrar em modo de economia de energia, retornando ao modo *Active* quando houverem tarefas na fila de execução (Figura 1b).

No segundo caso, nota-se que o consumo instantâneo de corrente é superior se comparado ao consumo do modo ativo em execução contínua.

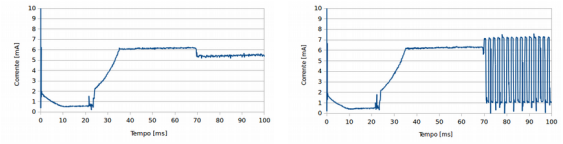


Figura 1. Ao retornar de um modo de economia de energia, o consumo de corrente instantâneo de um microcontrolador se mostra superior ao seu consumo em execução contínua.

O fenômeno mostrado não é esperado pelo programador do sistema, assim como não é previsto no *datasheet* do fabricante (T.I., 2015)(T.I., 2016).

Para que o problema seja formulado, é imprescindível a determinação dos intervalos de tempo de execução de uma tarefa (t_{ON}) e em economia de energia (t_{OFF}). Além disso, serão necessárias as medições de consumo de corrente do sistema em modo ativo (i_{ON}), em economia de energia (i_{OFF}) e em estado ativo após retorno do modo de economia de energia (i_{ON2OFF}). Conforme visto o fenômeno na Figura 1, as variáveis anteriormente definidas podem ser aproximadas pela Figura 2.

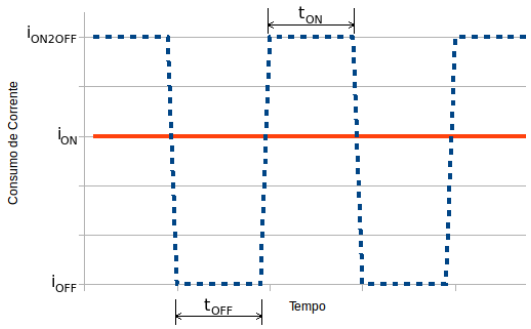


Figura 2. Definição das variáveis t_{ON}, t_{OFF}, i_{ON}, i_{OFF} e i_{ON2OFF}.

O consumo médio de corrente do sistema (i_{LPM}), para uma aplicação que utilize modos de baixo consumo de energia, pode ser equacionado de acordo com (1).

$$i_{LPM} = \frac{t_{ON} \cdot i_{ON\,2OFF} + t_{OFF} \cdot i_{OFF}}{t_{ON} + t_{OFF}} \quad (1)$$

Em (1) é considerado um sistema periódico com apenas uma tarefa em execução. Para generalizar o conceito, podemos adotar a ideia de carga de processamento ($C_{\%}$). Este pode ser definido como o somatório dos tempos consumidos (t_{ON_i}) pelas N tarefas do sistema, dividido pelo tempo total (t), conforme (2). Este somatório representa o tempo total ativo, consumindo a corrente i_{ON2OFF} .

$$C_{\%} = \frac{\sum_{i=0}^{N} t_{ON_i}}{t} \quad (2)$$

De modo análogo, o tempo ocioso do sistema é definido pelo somatório de todos os intervalos de tempo em economia de energia dividido pelo tempo total. O tempo ocioso também pode ser definido como o tempo total menos o somatório dos tempos em execução, conforme (3). Substituindo (2) em (3) tem-se (4).

$$C_{\%_{IDLE}} = \frac{\sum_{i=0}^{N} t_{OFF_i}}{t} = \frac{t - \sum_{i=0}^{N} t_{ON_i}}{t} \quad (3)$$

$$C_{\%_{IDLE}} = 1 - C_{\%} \quad (4)$$

Expandindo (1) para considerar todos os tempos das tarefas em execução, bem como o somatório dos tempos ociosos, determina-se (5).

$$i_{LPM} = \frac{\sum_{i=0}^{N} t_{ON_i} \cdot i_{ON\,2OFF} + \sum_{i=0}^{N} t_{OFF_i} \cdot i_{OFF}}{t} \quad (5)$$

Utilizando (2) e (4) em (5), obtém-se (6).

$$i_{LPM} = C_{\%} \cdot i_{ON2OFF} + (1 - C_{\%}) \cdot i_{OFF} \quad (6)$$

2.1 Política de economia de energia

O foco deste estudo é determinar em quais situações é válido adotar modos de economia de energia durante o período ocioso do sistema. Para tal, a partir de uma carga de processamento $C_{\%}$ estabelecida para o RTOS, define-se a política de gestão energética.

Através do equacionamento matemático dado por (6), para que o consumo de corrente médio do sistema seja válido, deve-se determinar se o sistema entrará em LPM enquanto estiver ocioso, ou se permanece em modo ativo, tal qual definido em (7).

$$i_{LPM}(C_{\%}) \begin{cases} >= & i_{ON} \Rightarrow Modo\ Ativo \\ < & i_{ON} \Rightarrow LPM \end{cases} \quad (7)$$

2.2 Hardware utilizado

O esquemático do hardware utilizado é mostrado na Figura 3. O mesmo possui somente resistores e capacitores necessários para sua operação, sem haver periféricos externos aderidos ao sistema.

O monitoramento dos testes foi viabilizado por duas modificações no hardware. A primeira foi utilizar um terminal do microcontrolador para supervisionar quando a tarefa *Idle* está em execução. A segunda alteração permite a análise das trocas de contexto através de outro terminal do microcontrolador.

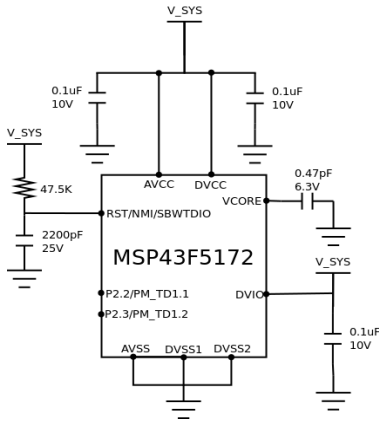


Figura 3. Esquemático do circuito básico para operação do MSP430F5172.

2.3 Implementação prática com o FreeRTOS

O FreeRTOS prevê a implementação de uma tarefa denominada *Idle*, que é executada quando não há tarefas ativas. A documentação sugere que o sistema seja colocado em modo de economia de energia quando a tarefa *Idle* estiver em execução (FreeRTOS, 2016b). A implementação desta tarefa está descrita no Código 1. A função foi desenvolvida para contemplar todos os modos de economia de energia desejados.

Código 1. Implementação de *Idle* do sistema FreeRTOS, voltado à economia de energia

```
// Escolhe o modo de consumo:
#define LPM_MODE 1

void vApplicationIdleHook (void) {
    /* Acende o LED de depuração */
    P2OUT |= 1 << 2;

    /* Não utilizar LPM */
    #if (LPM_MODE == -1)
        return;
    #endif

    /* Utiliza LPM1 */
    #if (LPM_MODE == 1)
        __bis_SR_register(LPM1_bits + GIE);
    #endif

    /* Utiliza LPM3 */
    #if (LPM_MODE == 3)
        __bis_SR_register(LPM3_bits + GIE);
    #endif
}
```

A aplicação-teste foi baseada em uma única tarefa a ser executada no sistema, a qual realiza algumas atividades, e ao seu término, é suspensa durante dois ciclos de *tick* do sistema. Logo, a carga de processamento do sistema permanece em 50%. A implementação desta tarefa é mostrada no Código 2.

2.4 Medições

Foram utilizados um osciloscópio Tektronix MDO4054 e uma *Source Meter* Keithley 2400. O primeiro foi ligado aos terminais mapeados para exibir informações da troca de contexto e da tarefa *Idle*, tal qual visto na Figura 4.

Código 2. Implementação de uma tarefa no sistema FreeRTOS

```
void vTaskCode() {
    portTickType xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();
    int i = 0;
    for (;;) {
        for (i = 0; i < 100; i++);
        vTaskDelayUntil(&xLastWakeTime, 2);
    }
}
```

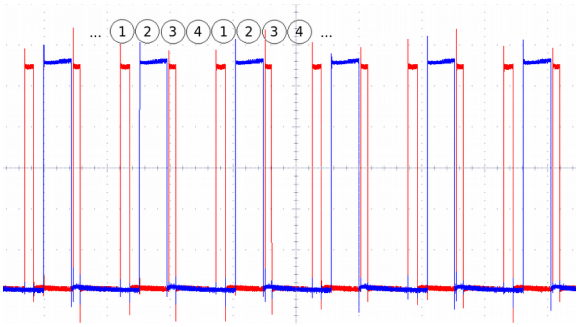


Figura 4. Visualização das trocas de contexto e da tarefa *Idle*.

Os tempos marcados com o número "2", na Figura 4, representam as várias execuções da tarefa *Idle*. Os intervalos marcados com "1" e "3" apresentam a troca de contexto entre as diferentes tarefas do sistema. O tempo indicado por "4" indica a execução da tarefa implementada no Código 2.

Com isso, é possível visualizar que, para cada troca de contexto, ou o sistema volta a executar a tarefa (marcação "4") ou entra em *Idle* (marcação "2"), o que configura uma carga de processamento $C_{\%}=50\%$, conforme esperado.

Quanto à corrente, foram obtidas as Figuras 5, 6 e 7. Foi destacada somente a área que contém execução do FreeRTOS, descartando-se o tempo de POR.

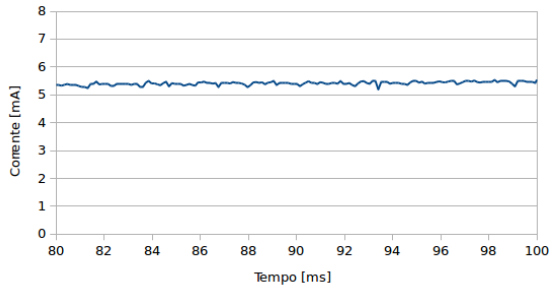


Figura 5. Consumo de corrente do sistema em *Active*.

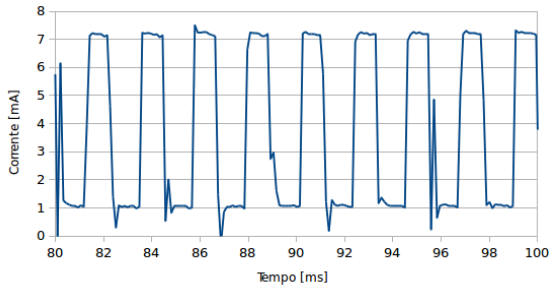


Figura 6. Consumo de corrente do sistema em LPM1, quando em *Idle*.

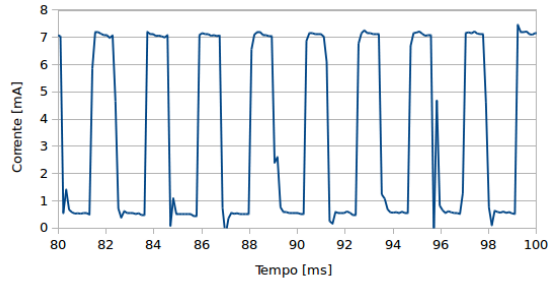


Figura 7. Consumo de corrente do sistema em LPM3, quando em *Idle*.

Os ruídos observados, que levam o sinal para níveis mais baixos que o patamar de 1mA (ou além de 7mA), são efeitos de *crosstalk* causado pelos terminais que são acionados justamente para monitorar o tempo de execução das tarefas, interferindo no sinal.

3 Resultados

A leitura obtida no modo LPM3 é bastante similar ao do LPM1, sendo a única diferença o consumo no estado de economia, em média 500µA em LPM3, em comparação aos 1mA do LPM1.

Um sistema operando permanentemente em modo *Active* consome cerca de 5,4mA. No entanto, quando o sistema opera em LPM, ele consome 7mA após retornar do modo de baixo consumo, representando um aumento de 23,5% em relação ao sistema continuamente em *Active*.

A Tabela 1 apresenta os valores encontrados para os consumos em cada um dos modos, considerando-se apenas os momentos sem execução de tarefas, para comparar os resultados com os informados pelo fabricante.

Tabela 1. Consumo de corrente para os diferentes modos de energia

Modo de energia	**Intervalos dados pelo fabricante**	**Consumo médio medido da placa em *Idle***
Active	3640 a 6150 µA	5382 µA
LPM1	85 a 104 µA	1076 µA
LPM3	1.2 a 3.0 µA	597.8 µA

Comparando os valores obtidos com os dados fornecidos pelo fabricante, percebe-se uma grande discrepância, principalmente para os modos de mais baixo consumo de energia.

De acordo com a Renesas (2013), "não é possível saber se os dados fornecidos pelo fabricante foram determinados executando uma instrução 'NOP', um *loop* infinito ou um algoritmo específico. Além disso, os fabricantes não informam quais periféricos estão ligados ou desligados. Deste modo, o que é fornecido em um *datasheet* representa uma referência, mas não uma realidade absoluta para todo e qualquer projeto".

Baseando-se na equação (6) e nas medições, obtém-se a Tabela 2. De acordo com estes dados, havendo uma carga de processamento de 50%, obtém-se uma economia de apenas 30%, em relação ao modo permanentemente em *Active*.

Tabela 2. Consumo de corrente médio do sistema para diferentes tipos de LPM

Estado	i_{OFF} [µA]	i_{ON2OFF} [µA]	i_{LPM} [µA]	Redução [%]
Active[1]	-	5382	5382	0
LPM1	1076	7035	3717	30,9
LPM3	598		3451	35,9

[1] neste caso, i_{LPM} representa i_{ON}, visto que não há mudança de estado

O consumo médio do sistema depende do modo LPM utilizado e da carga de processamento do sistema. Através de (6) pode-se inferir o ponto de operação C_{MAX}, para o qual o sistema, com LPM, consuma a mesma quantidade de energia do que o sistema permanentemente ligado. Para isto deve-se fazer $i_{LPM}=i_{ON}$ em (6), obtendo (8).

$$C_{MAX}=\frac{i_{ON}-i_{OFF}}{i_{ON2OFF}-i_{OFF}} \quad (8)$$

Para os diferentes modos de LPM do processador estudado, tem-se as curvas de carga de processamento contra o consumo de corrente, exibidas na Figuras 8.

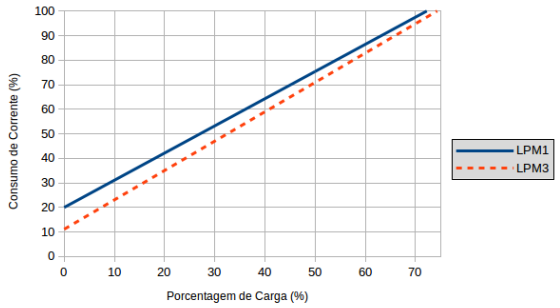


Figura 8. Relação entre carga de processamento e a variação no consumo de corrente, comparado com o sistema utilizando os modos LPM1 e LPM3.

Através da Figura 8, pode-se ver que um sistema que utiliza LPM1 como método de economia de energia dentro tarefa *Idle*, é capaz de reduzir o consumo em até 80%. No entanto, não realiza nenhuma tarefa útil ($C_{\%}=0$). Para LPM3, a diminuição de corrente é de 89%.

Observando o aspecto de carga máxima de processamento, em LPM1, o valor de C_{MAX} é 72,3%, ou seja, apesar de consumir a mesma quantidade de energia que um sistema permanentemente ativo, ele está limitado a 72,3% da capacidade de processamento. Se o sistema necessitar de maior capacidade de processamento, os ganhos com a economia de energia gerados pelo LPM1 são anulados. Para o LPM3, o valor crítico C_{MAX} é dado por 74,3% de capacidade de processamento.

4 Conclusões

Tendo em vista uma ascensão de tecnologias que necessitam minimizar seu consumo de energia, sem interferir na capacidade de processamento, uma alternativa prática é delinear uma estratégia de economia no próprio Sistema Operacional para atingir ambos objetivos. A definição de uma metodologia estática para a escolha dos métodos de economia de energia pode ser feita de maneira simples através da tarefa *Idle* do FreeRTOS.

Pelos resultados obtidos, percebeu-se que o sistema em estudo possui dois níveis distintos de consumo de energia na execução de tarefas. O consumo pós-LPM é superior ao do sistema permanentemente ligado, possivelmente devido ao religamento dos periféricos internos. Sendo assim, utilizando LPM para uma carga de processamento de 75%, consome-se a mesma quantidade de energia que o sistema com 100%, sem modo de baixo consumo de energia habilitado.

Assim, na definição de uma política de gestão energética é importante levar em conta esta diferença nos níveis de corrente, bem como a economia gerada nos modos de baixo consumo. De posse destes valores e da necessidade de processamento do sistema, é possível utilizar a política apresentada em (7) para minimizar o consumo de energia.

Conclui-se que não se deve analisar ou desenvolver uma política de gestão energética se baseando somente nos modos de economia de energia do processador. É necessário realizar uma análise conjunta da questão energética com a capacidade de processamento. Esta análise deve ser pautada em medições reais de consumo, visto que as informações fornecidas pelos fabricantes podem não contemplar todas as características do sistema analisado.

Além das questões abordadas, o modo de economia de energia gera um atraso cada vez que é ativado ou desativado. Este fato, aliado à carga máxima de processamento (C_{MAX}), pode reduzir ainda mais a quantidade de processamento disponível para execução de tarefas.

Para trabalhos futuros, é interessante analisar este comportamento para diferentes taxas de *clock* e modelos de microcontroladores. Outra abordagem seria desenvolver uma metodologia que analise o comportamento dinâmico da carga de processamento, adequando seu consumo de energia, segundo a política apresentada neste trabalho.

5 Agradecimentos

Agradecemos à Capes, Fapemig e CNPq pelo apoio prestado neste projeto.

Referências Bibliográficas

FREERTOS. (2016a) FreeRTOS – Market Leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. [Online] Disponível em: http://www.freertos.org. [Acessado em: 12 de Abril de 2016].

FREERTOS. (2016b) Idle Task and Idle Hook. [Online] Disponível em: http://www.freertos.org/RTOS-idle-task.html. [Acessado em: 03 de Maio de 2016].

JUANG, P. et al. (2002) Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. ACM.

RENESAS (2013) The True Low Power Concept - Implementing Powerful Embedded Controls with Minimum Energy Requirements. *Renesas Electronics America*. [Online] Disponível em: http://am.renesas.com. [Acessado em 12 de Setembro de 2013].

SAHA, S.; SARKAR, A.; CHACKRABARTI, A. (2015) Scheduling Dynamic Hard Real-Time Task Sets on Fully and Partially Reconfigurable Platforms. IEEE Embedded Systems Letters, Vol. 7, NO. 1.

T.I. (2015) MSP430F51x2 and MSP430F51x1 Mixed-Signal Microcontrollers.

T.I. (2016) MSP430F5172 Device Erratasheet.

ZHURALEV, S. et al. (2013) Survey of Energy-Cognizant Scheduling Techniques. IEEE Transactions on Parallel and Distributed Systems. Vol. 24, NO. 7.

6.2 Pedido de Patente da Técnica

INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
PROTOCOLO GERAL
marcos
014140001468
14/08/2014 10:14 DEMG
BR 10 2014 020131 9

< Uso exclusivo do INPI >

Espaço reservado para o protocolo | Espaço reservado para a etiqueta | Espaço reservado para o código QR

INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
Sistema de Gestão da Qualidade
Diretoria de Patentes

DIRPA	Tipo de Documento: Formulário	DIRPA	Página: 1/3
Título do Documento: Depósito de Pedido de Patente		Código: FQ001	Versão: 2
		Procedimento: DIRPA-PQ006	

Ao Instituto Nacional da Propriedade Industrial:
O requerente solicita a concessão de um privilégio na natureza e nas condições abaixo indicadas:

1. Depositante (71):

1.1 Nome: UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI

1.2 Qualificação: Instituição Federal de Ensino Superior

1.3 CNPJ/CPF: 21040001/0001-30

1.4 Endereço Completo: Av. BPS, 1303 - Bairro Pinheirinho - Itajubá/MG

1.5 CEP: 37500-903

1.6 Telefone: (35)3629-1693 1.7 Fax:

1.8 E-mail: nit@unifei.edu.br

☐ continua em folha anexa

2. Natureza: ☒ Invenção ☐ Modelo de Utilidade ☐ Certificado de Adição

3. Título da Invenção ou Modelo de Utilidade (54):

SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO

☐ continua em folha anexa

4. Pedido de Divisão: do pedido Nº **Data de Depósito:**

5. Prioridade: ☐ Interna (66) ☐ Unionista (30)

O depositante reivindica a(s) seguinte(s):

País ou Organização do depósito	Número do depósito (se disponível)	Data de depósito

☐ continua em folha anexa

INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
PROTOCOLO GERAL
marcos
014140001468
14/08/2014 10:14 DEMG
52400.124860/2014-32

"*SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO*".

A presente invenção refere-se a um sistema, possível de ser inserido em equipamentos eletrônicos em geral, mais especificamente um *SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO* que, de acordo com suas características, constitui uma abordagem completa destinada a permitir a redução do consumo de energia sem prejudicar a capacidade de processamento de sistemas eletrônicos microprocessados.

O estudo e desenvolvimento de sistemas eletrônicos microprocessados tem se expandido muito nos últimos anos e integram parte do cotidiano, desde relógios e câmeras até equipamentos móveis e telefones inteligentes. Parte destes são sistemas com funções específicas, cujas rotinas e técnicas de programação diferem daquelas usadas para projetos de aplicativos para computadores de propósito geral possuindo outras preocupações como consumo de energia, dimensão, peso, confiabilidade e portabilidade. Os sistemas eletrônicos com estas características e requisitos, que apresentam funcionalidades especificas são comumente denominados sistemas embarcados.

A questão do consumo de energia por um equipamento é mais crítica em sistemas móveis, pois determina questões de autonomia no funcionamento e impacta diretamente no peso do dispositivo definindo o tamanho e massa da bateria acoplada ao sistema.

É comum fazer uso de diversas ferramentas para atingir um determinado nível de consumo de energia como (1) redução da tecnologia dos transistores utilizados, (2) modos de baixo consumo de energia em microcontroladores e microprocessadores, (3) redução da frequência de execução do sistema, (4) redução da tensão de alimentação ou o (5) desligamento de setores e periféricos de hardware não utilizados. A abordagem 1 é definida no

momento de fabricação do sistema. Já as abordagens 2, 3, 4 e 5, estando disponíveis, podem ser ou não utilizadas, dependendo da necessidade do dispositivo. O problema com as últimas 4 abordagens é que o sistema gasta uma certa quantidade de tempo para fazer o processo de desligamento, depois consome outra quantidade de tempo para fazer o processo de inicialização e religamento em cada opção. Estes são tempos perdidos em que o sistema não consegue realizar nenhuma tarefa útil.

Outro ponto crítico para sistemas embarcados é a capacidade de respeitar prazos temporais de execução bem como tempos máximos de resposta a eventos. Estas características são definidas sob o termo tempo real. O modo para garantir que os dispositivos que necessitem de tempo real não tenham problemas com os prazos envolvidos é fazer uso de um sistema operacional de tempo real, que possui rotinas específicas para garantir que os prazos serão respeitados. No entanto tais sistemas possuem algumas limitações, sendo que a principal delas é que se o sistema precisa de mais de 100% da capacidade de processamento disponível, não é possível realizar a garantia de que as tarefas que exigem tempo real possam ser cumpridas.

Neste contexto, a presente invenção emprega componentes e processos em uma concepção diferenciada e específica, que visa a suplantar as deficiências e necessidades citadas e por meio da qual constitui um sistema capaz de manter a capacidade de resposta em tempo real, exigida por diversos tipos de aplicação, embarcadas ou não, ao mesmo tempo em que permita o dispositivo eletrônico se beneficiar dos sistemas de economia de energia presentes, de modo eficiente.

A presente invenção consiste em um sistema eletrônico composto por unidades de processamento capazes de executar um conjunto de tarefas analisando o consumo temporal da execução destas em um microcontrolador ou microprocessador embarcado. Sua finalidade é detectar dinamicamente a carga de processamento no sistema e selecionar qual o melhor estado eletrônico para maximização do consumo de energia sem afetar a capacidade de processamento da unidade. Tal proposição é formada por um

conjunto de soluções elétricas, eletrônicas e computacionais corretamente incorporadas que possibilitam a tal sistema a capacidade de se adaptar a mudanças no número de processos e tarefas executados pela unidade de processamento, diferenciando esta abordagem em termos de inovação, das atuais soluções estáticas.

A descrição da invenção será feita por meio da enunciação de seus respectivos objetivos, vantagens e demais características, o que ocorre com auxílio e referência aos desenhos apensos, nos quais:

A figura 1 apresenta um diagrama de blocos da arquitetura para o gerenciamento da proposta do sistema objeto desta patente.

Como se infere a partir das figuras em anexo que integram e ilustram a presente invenção, na figura (1) é apresentada a arquitetura do *SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO*.

A arquitetura do sistema proposto constitui-se como um circuito eletroeletrônico que pode ser parte integrante de qualquer outro dispositivo que possua necessidade de processamento. Tal unidade é formada por um módulo de processamento (1A), um módulo de gerenciamento de energia (1B), um módulo de periféricos eletrônicos (1C), um conjunto de tarefas computacionais (1D), um módulo de gerenciamento destas tarefas (1E) e por um circuito de alimentação (1F).

O módulo de processamento (1A) emprega um circuito composto por uma unidade de processamento de dados computacionais, como um microcontrolador ou um microprocessador. Por meio deste módulo as tarefas (1D) podem ser executadas, pausadas ou resumidas de tal forma que consigam ser processadas de modo a atender os seus requisitos.

O módulo de gerenciamento de energia (1B) é composto por circuitos eletrônicos que possuem a capacidade de atuar nos módulos de processamento (1A), para reduzir seu consumo, de periféricos eletrônicos (1C), para ligar ou desligar alguns de seus componentes e de alimentação (1F), para

modificar as tensões envolvidas. Este módulo pode ser interno aos chips ou externo, fazendo uso de circuitos comutadores de energia para ligar e desligar parte dos componentes e periféricos presentes (1C), variação da tensão de alimentação de parte ou de todo o sistema para reduzir a energia consumida, variação da frequência de operação para reduzir a taxa de consumo de energia no tempo (potência elétrica) ou qualquer outra forma que visa reduzir ou parar o consumo de energia em parte do sistema ou no sistema como um todo.

O módulo de periféricos eletrônicos (1C) é composto do conjunto de periféricos eletrônicos que realizam funções de interface de entrada de sinais, interface de saída de sinais, controle armazenamento ou exibição de sinais e informações relevantes. Este módulo é composto por sistemas de interface homem máquina (displays, teclados, etc), interfaces de comunicação (serial, paralela, com ou sem fios, etc), dispositivos de armazenamento de dados (cartões de memória, chips de memória flash, etc) entre outros dispositivos periféricos que o sistema de processamento (1A) tenha acesso direto ou indireto.

O conjunto de tarefas computacionais (1D) reúne todas as tarefas computacionais desenvolvidas que regem o funcionamento do sistema e implementam as funcionalidades desejadas pelo equipamento.

O módulo gestor das tarefas (1E) é o componente computacional responsável por gerenciar a sequência de execução das tarefas (1D) bem como indicar ao sistema de gerenciamento de energia (1B) em qual estado o sistema de economia de energia deve se encontrar. A decisão de qual tarefa deve ser executada é tomada por qualquer sistema de agendamento de tarefas, ao passo que a decisão sobre o modo de energia a ser adotado naquele instante é tomada baseado na quantidade de tarefas (1D), na importância das tarefas (1D), nas necessidades temporais das tarefas (1D) na taxa de ocupação do processador (1A) e na capacidade e variedade de modos de gerenciamento de energia (1B).

Por sua vez, o módulo de alimentação (1F) congrega uma bateria/pilha, ou um conjunto de baterias/pilhas, ou circuitos conversores de tensão, ou qualquer composição dos mesmos, capazes de produzir os sinais de

alimentação requeridos pelos demais módulos do sistema (1).

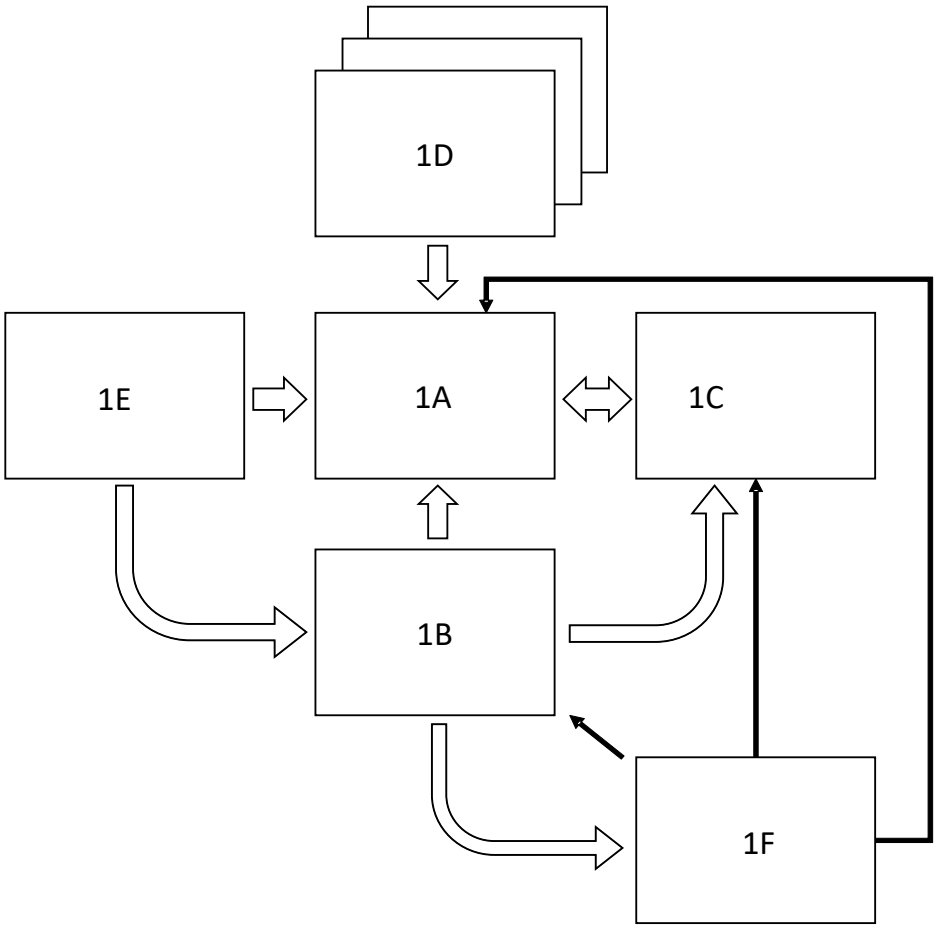
O princípio de operação do *SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO* tem como base a utilização de informações sobre uso de processamento para tomada de decisão sobre o acionamento de recursos para economia de energia. O módulo gestor de tarefas (1E) realiza a medição da taxa de uso do processador. De posse desta taxa, conhecendo os períodos de tempo consumidos pelo módulo de gerenciamento de energia (1B) para ligar e desligar os periféricos eletrônicos (1C), modificar a frequência de execução do módulo de processamento (1A) e gerenciar o módulo de alimentação (1F) para modificação de suas grandezas, o módulo gestor de tarefas (1E) escolhe o melhor conjunto de ações que será implementado pelo módulo de gerenciamento de energia (1B) para minimizar o consumo de energia sem que o tempo disponível para processamento das tarefas (1D) seja prejudicado. Este procedimento é realizado continuamente acompanhando a dinâmica na mudança da carga de processamento de modo a escolher a cada instante o melhor conjunto de ações para minimização do consumo. Desta forma, o sistema fará uso eficiente da energia disponível sem que os atrasos inerentes às ações do módulo de gerenciamento de energia (1B) gerem algum problema no comprimento do prazo de execução das tarefas (1D).

REIVINDICAÇÕES

1- "*SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO*", caracterizado por uma arquitetura de um sistema eletroeletrônico (1), podendo ser parte integrante de qualquer outro dispositivo eletroeletrônico, visando a redução do seu consumo de energia sem afetar ou prejudicar a capacidade do sistema de processar tarefas computacionais.

2- Arquitetura de um sistema de acordo com a reivindicação 1, que é caracterizada por um circuito eletroeletrônico e tarefas computacionais com funcionalidades de propósito geral ou específico. Sua arquitetura é constituída conforme o diagrama de blocos da Figura 1, contendo um módulo de processamento (1A) responsável pela execução de um conjunto de tarefas (1D), um módulo de gestão de energia (1B) responsável por modificar o comportamento do sistema reduzindo seu consumo de energia atuando nos módulos de processamento (1A), alimentação (1F) e nos periféricos eletrônicos de interface (1C); um conjunto de periféricos eletrônicos de interface (1C) responsável por realizar a intermediação do sistema com o ambiente externo; um conjunto de tarefas computacionais (1D); um módulo gestor de tarefas (1E), responsável por gerenciar a execução das tarefas (1D) e informar ao módulo de gerenciamento de energia (1B) qual estado o sistema deve ser encontrar para minimizar o consumo de energia sem impactar na capacidade de execução das tarefas; por fim, por um módulo de alimentação (1F) por meio do qual prover alimentação a todos os demais módulos da unidade.

FIGURA 1



RESUMO

Patente de Invenção "*SISTEMA PARA ECONOMIA DE ENERGIA EM SISTEMAS ELETRÔNICOS MICROPROCESSADOS BASEADO EM RESTRIÇÕES TEMPORAIS DE PROCESSAMENTO*"

A presente invenção refere-se a um sistema que constitui uma abordagem completa destinada a permitir a redução do consumo de energia sem prejudicar a capacidade de processamento de sistemas eletrônicos microprocessados ou microcontrolados, detectando dinamicamente a carga de processamento do sistema e selecionando qual o melhor estado eletrônico para maximização do consumo de energia sem afetar a capacidade de processamento da unidade. Tal proposição é formada por um conjunto de soluções elétricas, eletrônicas e computacionais corretamente incorporadas que cedem a tal sistema a capacidade de se adaptar a mudanças na quantidade de processos e tarefas executados. O princípio de operação tem como base a utilização de informações sobre uso de processamento para tomada de decisão sobre o acionamento de recursos para economia de energia. Este procedimento é realizado continuamente e acompanhando a dinâmica na mudança da carga de processamento de modo a escolher a cada instante o melhor conjunto de ações para minimização do consumo.

6.3 Configuração da SourceMeter 2602A

A SourceMeter foi configurada no modo *2-wire*, o qual é suficiente para leituras de corrente da ordem de uA . Seu funcionamento pode ser visto na figura 27.

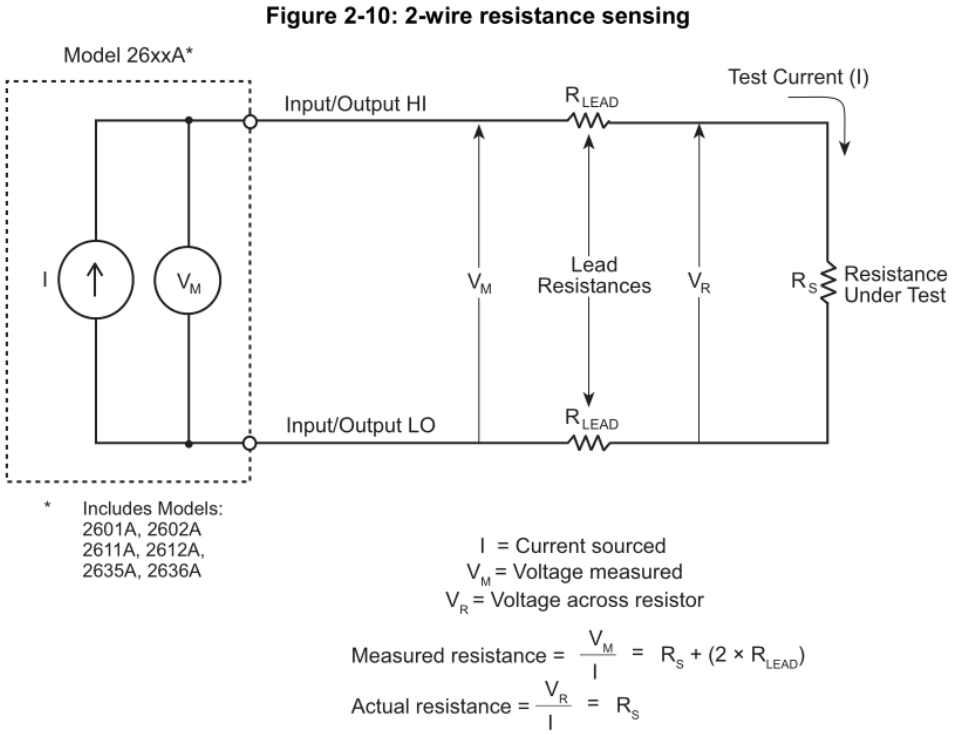


Figura 27 – Funcionamento interno da SourceMeter para a leitura em modo *2-wire*.

Utilizando o software *TSP Express* [59], foi possível configurar o instrumento para realizar leitura com grande quantidade de pontos, com níveis de corrente desejáveis:

ID: localnode.smua
Type: Sweep
Name: Sweep_SMU
Source Function: voltage
Source Mode: normal
Meas. Function: current
Advanced: Sweep_SMU (Sweep) Properties: Sense Mode: Two-Wire
Advanced: Sweep_SMU (Sweep) Properties: Source Limit: 100.0 *mA*
Sweep Segment Device Management: Sweep: localnode.smua (2602*A*)
Timing: NPLC: 0.003
Timing: Auto Zero: ONCE
Timing: Source Delay Type: OFF
Timing: Measure Count (per step): 1
Timing: Measure Delay Type: OFF
Timing: Measure Filter Enable: OFF
Timing: Measure Analog Filter: OFF

Step (outer): # *steps:* 1
Step (outer): Sweep (inner): # *points:* 1000
Step (outer): Sweep (inner): time/point: 110 *us*
Step (outer): Sweep (inner): Name: Sweep_MCU
Step (outer): Sweep (inner): Pulse: NO
Step (outer): Sweep (inner): Source Range: 6 *V*
Step (outer): Sweep (inner): Start: 3.3*V*
Step (outer): Sweep (inner): Stop: 3.3*V*
Step (outer): Sweep (inner): Style: LIN
Step (outer): Sweep (inner): Meas. Range: 100 *mA*
Data: Data Collection Options (set prior to running script): Buffers: AUTO