Mapeamento do isolamento espacial da ARINC 653 para a arquitetura *ARMv7-A*

Luís Fernando Arcaro
Univ. Fed. de Santa Catarina
DAS-CTC-UFSC C.Postal 476
Florianópolis-SC, Brasil
E-mail: luis.arcaro@posgrad.ufsc.br

Rômulo Silva de Oliveira
Univ. Fed. de Santa Catarina
DAS-CTC-UFSC C.Postal 476
Florianópolis-SC, Brasil
E-mail: romulo.deoliveira@ufsc.br

Resumo — A especificação *ARINC 653* define um padrão para a interface funcional e os requisitos temporais de Sistemas Operacionais (SOs) para sistemas de aviação, e baseia-se em partições isoladas temporal e espacialmente nas quais são executados os processos de aplicação. Nesses SOs, o isolamento espacial (proteção de memória) de partições deve garantir que o *software* de aplicação terá acesso restrito às regiões de memória reservadas para a partição, evitando que a execução de uma partição interfira nas demais. Neste artigo é apresentada uma abordagem para implementação do isolamento espacial exigido pela especificação *ARINC 653* em processadores baseados no perfil *ARMv7-A* da arquitetura *ARMv7*.

Abstract — The *ARINC 653* specification defines a standard functional interface and temporal requirements of Operating Systems (OSs) for aviation systems, and is based on partitions temporally and spatially isolated in which are run the application processes. In these OSs, the spatial isolation (memory protection) of partitions must guarantee that the application software will have restricted access to the memory regions that are reserved to the partition, avoiding the execution of one partition to interfere on the others. In this paper we present an approach for implementing the spatial isolation required by the *ARINC 653* specification using processors based on the *ARMv7-A* profile of the *ARMv7* architecture.

I. Introdução

A especificação *ARINC 653* [1] descreve a interface funcional a ser oferecida ao *software* de aplicação e os aspectos temporais a serem atendidos por Sistemas Operacionais (SOs) empregados em sistemas computacionais utilizados em aeronaves. Para operação nesse tipo de sistema são utilizadas implementações comerciais de SOs compatíveis com a *ARINC 653* que, por necessitarem de certificação, apresentam custo financeiro bastante elevado; porém são também necessárias implementações de SOs desse tipo para treinamento de engenheiros aeronáuticos especializados em aviônica e para experimentação de novas aplicações, propósitos para os quais a certificação não é necessária. Um dos pontos centrais da especificação *ARINC 653* é a utilização de partições com isolamento temporal e espacial, permitindo a execução de múltiplas aplicações com diferentes objetivos e criticalidades na mesma plataforma de *hardware*, e ainda garantindo que eventuais falhas ocorridas em uma partição possam ser tratadas de forma a não afetar a execução das demais. Para isso, as partições são executadas em intervalos definidos dentro de uma janela cíclica de tempo (isolamento temporal) e as regiões de memória utilizadas por cada uma delas são protegidas contra acesso das demais (isolamento espacial). As exceções ocorridas durante a execução de uma determinada partição são tratadas através de mecanismos que não violam o escalonamento, garantindo assim a não propagação de faltas entre partições [1]–[4].

SOs compatíveis com a especificação *ARINC 653* são compostos de um módulo sobre o qual são escalonadas múltiplas partições que, do ponto de vista de isolamento, podem ser consideradas análogas aos processos de um SO tradicional. Nessas partições são executados processos, análogos às *threads* de SOs tradicionais desse mesmo ponto de vista, sendo esses utilizados para a implementação do *software* de aplicação [4]. A Figura 1 ilustra a hierarquia de elementos de um SO de acordo com a *ARINC 653*.

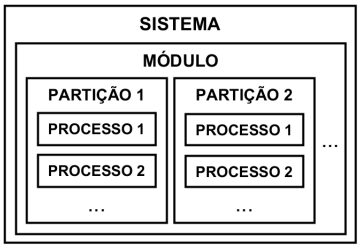

Figura 1. Hierarquia de elementos de um SO

A principal função dos mecanismos de isolamento espacial (proteção de memória) em SOs compatíveis com a *ARINC 653* é a detecção de tentativas de acesso a regiões de memória proibidas, geralmente causadas por erros de programação ou de configuração, permitindo que sejam disparados mecanismos para tratamento desses eventos de forma controlada (sem comprometimento do sistema) [1]. Alguns tipos de erro de acesso à memória, como por exemplo estouros de pilha, podem ser detectados através de mecanismos implementados em *software*, conforme apresentado em [5] e [6]. Porém essas abordagens não oferecem garantias quanto à detecção desses erros, já que empregam verificações periódicas e, portanto, não são capazes de legitimar todas as instruções de acesso à memória. Soluções eficientes para proteção de memória empregam dispositivos de *hardware* dedicados a esse fim, como *Memory Protection Unit*s (*MPU*s) e *Memory Management Unit*s (*MMU*s), que permitem a especificação de permissões de acesso que são verificadas a cada instrução de máquina executada, garantindo assim a detecção de quaisquer acessos indevidos à memória [7].

Este artigo aborda a implementação de mecanismos eficientes de isolamento espacial em uma plataforma de *hardware* de baixo custo financeiro, visando exclusivamente objetivos didáticos e experimentais. A plataforma de *hardware* selecionada é comercialmente conhecida como *BeagleBone*, e consiste de um microcomputador de *hardware* aberto equipado com um processador *Texas Instruments* (*TI*) AM335X (AM3358/AM3359) dotado de um núcleo *Advanced RISC Machines* (*ARM®*) *Cortex™-A8*, que se baseia na arquitetura *ARM®* versão 7 (*ARMv7*) [8], [9].

A arquitetura *ARMv7* possui três diferentes perfis: o *ARMv7-A* para processadores de aplicação, o *ARMv7-R* para processadores de tempo real e o *ARMv7-M*, para microcontroladores. Ela ainda define duas diferentes arquiteturas de gerenciamento de memória: a *Protected Memory System Architecture* versão 7 (*PMSAv7*) – empregada no perfil *ARMv7-R* – que oferece uma *MPU* apenas para proteção de memória, e a *Virtual Memory System Architecture* versão 7 (*VMSAv7*) – utilizada pelo perfil *ARMv7-A* – que possui uma *MMU* com recursos de proteção e virtualização de memória [7].

Enquanto a *VMSAv7* utiliza tabelas de tradução armazenadas em memória e *Translation Lookaside Buffers* (*TLBs*) para definição dos mapeamentos de memória, a *PMSAv7* emprega registradores para fim semelhante. Dessa forma, a *PMSAv7* oferece maior previsibilidade temporal em relação à *VMSAv7* e, assim, é mais adequada para operação nos sistemas críticos aos quais a *ARINC 653* destina-se [7], [10], [11]. O núcleo *Cortex™-A8* utilizado pelo processador da plataforma de *hardware* escolhida baseia-se no perfil *ARMv7-A*, e portanto suporta apenas a arquitetura *VMSAv7*, o que o torna inadequado para operação nesses sistemas mas não o desqualifica para fins experimentais nesse mesmo contexto.

Este artigo apresenta uma abordagem para implementação do isolamento espacial exigido pela especificação *ARINC 653*, empregando para isso o subconjunto mínimo de recursos da *MMU* da arquitetura *VMSAv7* destinado à proteção de memória. Por esse motivo esses recursos serão abordados em certa profundidade, limitando-se porém às funcionalidades efetivamente empregadas nessa abordagem.

II. Contextos de execução

SOs geralmente suportam a execução de diversas tarefas de forma concorrente, ou seja, executam tarefas de forma alternada sobre um núcleo de processamento a fim de dar a impressão de que estão sendo executadas de forma paralela. Essa abordagem exige que a execução de uma tarefa possa ser interrompida a um determinado momento e retomada mais tarde, tornando necessário que as informações relacionadas ao estado do núcleo do processador sejam armazenadas quando de sua interrupção, para que possam ser restauradas no momento em que a execução da tarefa for retomada. A esse conjunto de informações dá-se o nome de **contexto de execução**, e o processo de interrupção de uma tarefa e retomada de outra é chamado de **troca de contexto de execução** [12].

Em SOs tradicionais as tarefas que concorrem por execução no processador são as *threads*, sendo que cada processo possui no mínimo uma delas e pode realizar a criação de outras conforme a aplicação à qual serve [12]. Por sua vez, SOs compatíveis com a especificação *ARINC 653* possuem múltiplas tarefas ligadas aos diversos elementos do sistema, e o escalonamento de cada uma delas varia de acordo com a função à qual servem. A Figura 2 ilustra a disposição dessas tarefas (em linhas tracejadas) em relação aos elementos do sistema aos quais estão relacionadas, e ainda sua cardinalidade (entre parênteses), ou seja, o número de tarefas do tipo em questão que podem ser encontradas no elemento ao qual pertence [1].

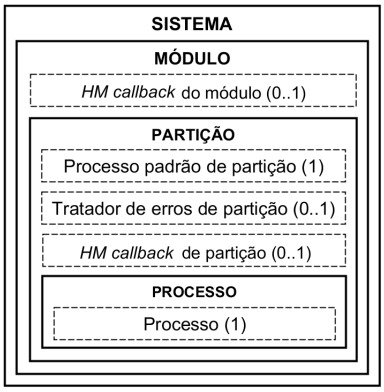

Figura 2. Tarefas de sistemas ARINC 653

Algumas dessas tarefas podem ser implementadas pelo SO tanto em contextos de execução distintos quanto como simples chamadas de procedimento a partir de outros contextos de execução, como é o caso dos *HM callbacks*. Neste trabalho todas essas tarefas serão tratadas como diferentes contextos de execução, permitindo uma análise aprofundada dos requisitos de isolamento espacial impostos por cada uma delas. Apresenta-se a seguir a função de cada um desses contextos de execução, associando a cada um deles uma sigla que será utilizada a seguir para referência resumida.

***HM callback* do módulo (MOD.HMC)** Contexto de execução do mecanismo de monitoramento que é disparado quando da ocorrência de erros mapeados em configuração para o nível de módulo;

***HM callback* de partição (PAR.HMC)** Contexto de execução do mecanismo de monitoramento que é disparado quando da ocorrência de erros mapeados em configuração para o nível de partição, ou na ocorrência de erros mapeados para o nível de processo quando da ausência de um processo tratador de erros na partição [1];

Tratador de erros de partição (PAR.EH) Contexto de execução do processo tratador de erros de uma partição, o qual é disparado quando da ocorrência de erros mapeados em configuração para o nível de processo;

Processo padrão de partição (PAR.DEF) Contexto de execução no qual uma partição é inicializada, ou seja, tem seus processos, artefatos de comunicação e demais elementos criados e inicializados;

Processo (PRO) Contexto de execução de um processo de uma partição, utilizado para execução do *software* de aplicação.

Esses contextos de execução estão sujeitos a restrições de acesso à memória do processador, as quais são aplicadas pela *MMU* empregada. As principais características desse dispositivo serão apresentadas na próxima seção.

III. A *MMU* DA *VMSAv7*

A *MMU* da arquitetura *VMSAv7* faz uso de tabelas de tradução armazenadas na memória principal que permitem o mapeamento de endereços virtuais (*Virtual Addresses* (*VAs*)) a endereços físicos (*Physical Addresses* (*PAs*)), assim como o controle de acesso a todo o espaço de endereços de memória do processador. Essas tabelas de tradução são formadas por um conjunto de descritores, cada um relacionado a um determinado intervalo de endereços virtuais e associado, explícita ou implicitamente, a um intervalo de endereços físicos, a um conjunto de atributos e ainda às permissões de acesso nos diferentes níveis de privilégio suportados pelo processador – PL0 (não privilegiado), utilizado pelo *software* de aplicação, e PL1 (privilegiado), utilizado pelo núcleo do SO [13].

Se essas tabelas de tradução fossem consultadas na memória principal toda vez que uma instrução que realiza acesso a endereços de memória é executada, o tempo de acesso seria proibitivo. Por isso essa *MMU* utiliza uma memória *cache* denominada *TLB*, que armazena os descritores mais comumente utilizados e evita, assim, a consulta frequente às tabelas de tradução [12].

Tabelas de tradução podem ser utilizadas na arquitetura *VMSAv7* em dois formatos distintos, em função da aplicação à qual servirão [7]:

Descritores curtos Possuem descritores de 32 *bits* e mapeiam endereços virtuais de 32 *bits* a endereços físicos de até 40 *bits*, permitindo regiões de 4KB, 64KB, 1MB ou 16MB;

Descritores longos São formadas por descritores de 64 *bits* e mapeiam endereços de até 40 *bits*, tanto virtuais quanto físicos, suportando regiões de 4KB, 2MB e 1GB.

Neste trabalho considera-se que não são empregados endereços virtuais longos (maiores que 32 *bits*), e portanto as informações apresentadas a seguir supõem o emprego de tabelas de tradução de descritores curtos, por serem menores e suficientes nesse contexto. Tabelas desse tipo podem ser de primeiro ou de segundo nível, e são compostas da seguinte forma [7]:

De primeiro nível Possui até 4096 descritores associados a endereços virtuais alinhados a 1MB, e cada um deles pode ser, exclusivamente, um descritor:

- De falha – a região (1MB) não pode ser acessada;
- De propriedades da seção (1MB) associada;
- De propriedades de uma superseção (16MB), sendo que superseções devem estar alinhadas a 16MB e seus descritores devem ser repetidos 16 vezes consecutivas na tabela de tradução;
- De tabela de segundo nível, com um ponteiro para a tabela que define o mapeamento da região (1MB).

De segundo nível Possui 256 descritores associados a endereços base alinhados a 4KB dentro da seção de 1MB apontada pelo registro da tabela de primeiro nível, e cada um deles pode ser, exclusivamente, um descritor:

- De falha – a região (4KB) não pode ser acessada;
- De propriedades da página pequena (4KB) associada;
- De propriedades de uma página grande (64KB), sendo que páginas grandes devem estar alinhadas a 64KB e seus descritores devem ser repetidos 16 vezes consecutivas na tabela de tradução de segundo nível.

Na *MMU* da arquitetura *VMSAv7* estão disponíveis dois registradores apontadores de tabelas de tradução de primeiro nível: *TTBR0* e *TTBR1*. Usualmente, um deles é utilizado para apontar a tabela de tradução que define os acessos a regiões globais (válidas para qualquer contexto de execução do sistema), não sendo portanto alterado durante trocas de contexto, enquanto o outro é utilizado para apontar a tabela de tradução que define as regiões de memória acessíveis ao contexto de execução atual, e que é portanto necessariamente alterado em trocas de contexto. A região de memória mapeada pela tabela de tradução apontada por *TTBR0* inicia no endereço zero (0x00000000) e tem tamanho variável definido através do campo *N* do registrador de controle denominado *TTBCR*. Por sua vez, a tabela de tradução apontada pelo registrador *TTBR1* possui tamanho fixo de 4096 descritores (16KB), cobrindo portanto todo o espaço de endereçamento, porém é utilizada apenas na porção que não é mapeada por *TTBR0* segundo o campo *TTBCR.N*. A Tabela I apresenta como, de acordo com o valor atribuído a *TTBCR.N*, as regiões mapeadas por *TTBR0* e *TTBR1* são delimitadas quando do uso de tabelas de tradução de descritores curtos [7]:

TTBCR.N	Região mapeada por TTBR0	Região mapeada por TTBR1
0b000	0x00000000-0xFFFFFFFF	Nenhuma
0b001	0x00000000-0x7FFFFFFF	0x80000000-0xFFFFFFFF
0b010	0x00000000-0x3FFFFFFF	0x40000000-0xFFFFFFFF
0b011	0x00000000-0x1FFFFFFF	0x20000000-0xFFFFFFFF
0b100	0x00000000-0x0FFFFFFF	0x10000000-0xFFFFFFFF
0b101	0x00000000-0x07FFFFFF	0x08000000-0xFFFFFFFF
0b110	0x00000000-0x03FFFFFF	0x04000000-0xFFFFFFFF
0b111	0x00000000-0x01FFFFFF	0x02000000-0xFFFFFFFF

Tabela I. MAPEAMENTO TTBR0-TTBR1

A Figura 3, adaptada de [7], representa a forma como são apontadas e interpretadas as tabelas de tradução de descritores curtos de primeiro e de segundo nível para o mapeamento de seções (1MB), superseções (16MB), páginas pequenas (4KB) e páginas grandes (64KB).

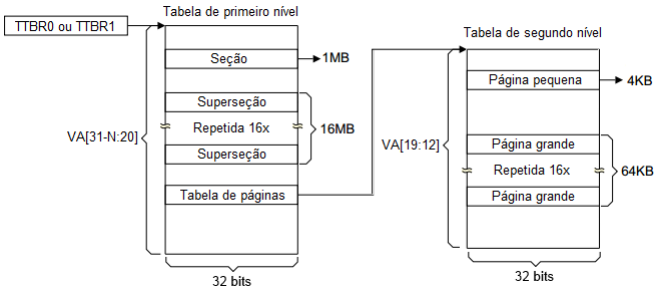

Figura 3. Tabelas de tradução de descritores curtos

Na arquitetura *VMSAv7* as tabelas de tradução suportam a declaração de regiões de memória globais, cujo mapeamento é válido para qualquer contexto de execução, e de regiões de memória não globais, cujo mapeamento é válido apenas para o contexto de execução atual, o que torna necessária a utilização de um mecanismo de distinção de contextos de execução. Essa distinção é realizada através do registrador *CONTEXTIDR* da *MMU*, que deve ser carregado com o identificador do contexto de execução atual e, assim como cada entrada das *TLBs*, possui um campo denominado *Address Space Identifier* (*ASID*). O campo *ASID* das *TLBs* armazenam

o *ASID* contido no registrador *CONTEXTIDR* no momento em que o descritor foi inserido na *TLB*, e uma entrada da *TLB* só é recuperada se, além de descrever os atributos da região de memória que está sendo acessada, possuir o campo *ASID* igual ao do registrador *CONTEXTIDR* no momento do acesso, ou se armazena um descritor global [7]. Dessa forma evita-se que dois descritores pertencentes a diferentes contextos de execução sejam utilizados erroneamente por referirem-se a uma mesma região de memória, além de ser fundamental para que as *TLBs* não precisem ser completamente invalidadas quando da troca de contexto de execução, o que faria com que as tabelas de tradução fossem sempre acessadas nessas ocasiões [12].

Os descritores de seção, superseção, página pequena e página grande das tabelas de tradução de descritores curtos da arquitetura *VMSAv7* suportam, entre outros, os seguintes atributos para cada região mapeada [7]:

Tipo de memória Define propriedades relacionadas ao comportamento do processador no acesso à região;
Permissão de execução Indica se é permitida ou não a execução de instruções na região;
Permissão de acesso Indica o tipo de acesso permitido à região nos diferentes níveis de privilégio suportados pelo processador (PL0 e PL1);
Globalidade Define se o descritor é ou não global, ou seja, se os atributos por ele definidos são ou não válidos para qualquer contexto de execução.

Verifica-se que a *MMU* apresentada oferece os recursos necessários para o controle de acesso a regiões de memória com granularidade de até 4KB. A seção a seguir apresenta a divisão do espaço de endereçamento de memória do processador em regiões, de forma que o acesso a elas possa ser controlado através desse mecanismo.

IV. Regiões de memória

A separação da memória do sistema em regiões é fundamental para tornar possível o mapeamento de permissões de forma maximamente restritiva, explorando os recursos oferecidos pela *MMU* a fim de alcançar um isolamento espacial de partições compatível com as exigências da *ARINC 653*. As regiões de memória apresentadas nesta seção decorrem exclusivamente de características exigidas por essa especificação, porém para a implementação efetiva de um SO em uma plataforma de *hardware* seria possivelmente necessária a definição de outras regiões. Essa necessidade geralmente surge em função de decisões de projeto ou de características específicas da plataforma, e portanto não são previstas neste trabalho. A Tabela II apresenta de forma sucinta as regiões de memória encontradas em sistemas *ARINC 653*, atribuindo a cada uma delas uma sigla para que possam ser referenciadas de forma resumida, e destaca ainda a cardinalidade dessas regiões, ou seja, quantas delas são alocadas no sistema, sendo Pa o conjunto de partições e $Pr(X)$ o conjunto de processos da partição X.

Na próxima seção é apresentado o mapeamento das permissões aplicadas a essas regiões de memória para cada contexto de execução apresentado anteriormente, destacando ainda a função de cada uma delas perante o SO.

Sigla	Descrição	Cardinalidade
MOD.STK	Pilha do módulo	1
MOD.COD	Código do módulo	1
MOD.DAT	Dados do módulo	1
MOD.HMC.STK	Pilha do *HM callback* do módulo	0 a 1
PAR.COD	Código de partição	$\|Pa\|$
PAR.DAT	Dados de partição	$\|Pa\|$
PAR.DEF.STK	Pilha do processo padrão de partição	$\|Pa\|$
PAR.EH.STK	Pilha do tratador de erros de partição	0 a $\|Pa\|$
PAR.HMC.STK	Pilha do *HM callback* de partição	0 a $\|Pa\|$
PRO.STK	Pilha de processo	$\sum \|Pr(X \in Pa)\|$

Tabela II. Regiões de memória

V. Mapeamento de permissões

Cada contexto de execução do SO está associado às regiões de memória apresentadas através de um mapeamento de permissões de acesso, que é empregado para a construção das tabelas de tradução utilizadas pela *MMU* sempre que o contexto de execução correspondente estiver sendo processado. Esse mapeamento define a que regiões de memória cada contexto de execução terá acesso, que operações poderão ser realizadas sobre essas regiões (leitura, escrita e/ou execução), e em que níveis de privilégio essas operações poderão ser executadas.

Uma vez que a especificação *ARINC 653* exige o isolamento espacial a nível de partição, é natural deduzir-se que a construção de uma tabela de tradução por partição deve ser suficiente para o mapeamento de todas as regiões de memória acessíveis pelos contextos de execução de cada partição. Porém, a detecção de determinados tipos de erro de acesso à memória, como a diferenciação entre violações de memória (acesso a endereços inválidos ou proibidos) e estouro de pilha (empilhamento ou desempilhamento de dados além dos limites da região de pilha), pode tornar-se complexa ou inviável se empregada apenas uma tabela de tradução por partição [1]. Por esse motivo, assume-se neste trabalho que é construída uma tabela de tradução para cada contexto de execução do SO, o que gera *overheads* relacionados ao número relativamente grande de tabelas empregadas mas beneficia a detecção de erros.

As permissões utilizadas nos mapeamentos apresentados são expressas no formato **PRV/NPR[*]**, onde **PRV** e **NPR** são, respectivamente, as permissões para modo privilegiado (PL1) e não privilegiado (PL0), e podem assumir os valores **NO** (sem permissão), **RO** (somente leitura) ou **RW** (leitura e escrita). O indicador opcional **[*]** denota que a execução de código é proibida na região em questão. Três diferentes permissões são utilizadas (padrão, partição e processo), e são aplicadas da seguinte forma:

1) Se o contexto de execução pertence ao mesmo processo ao qual a região de memória pertence e a permissão de processo não é vazia, é aplicada a permissão de **processo**;
2) Se a condição anterior não foi satisfeita e o contexto de execução pertence à mesma partição à qual a região de memória pertence e a permissão de partição não é vazia, é aplicada a permissão de **partição**;
3) Se as condições anteriores não foram satisfeitas e a permissão padrão não é vazia, é aplicada a permissão **padrão**;
4) Se nenhum caso anterior foi satisfeito, o acesso à região não é permitido.

Apresenta-se a seguir a função e o mapeamento de permissões atribuído a cada região de memória, assim como uma descrição justificada dessas permissões.

A. Pilha do módulo (MOD.STK)

A pilha do módulo é utilizada tanto durante a inicialização do sistema quanto pelas rotinas de tratamento de exceções. Uma vez que exceções utilizam esta região como pilha e já que essas executam em modo privilegiado e podem ocorrer em qualquer contexto de execução, esta região possui permissão de leitura e escrita para todos os contextos de execução, porém apenas em modo privilegiado. O contexto de execução de inicialização do sistema (não tratado neste artigo) pode necessitar permissão de leitura e escrita também em modo não privilegiado. Por tratar-se de uma região de pilha, a execução não é permitida. A Tabela III apresenta o mapeamento de permissões desta região.

Contexto de execução	Padrão
MOD.HMC	RW/NO*
PAR.DEF	RW/NO*
PAR.EH	RW/NO*
PAR.HMC	RW/NO*
PRO	RW/NO*

Tabela III. PERMISSÕES PARA A REGIÃO MOD.STK

B. Código do módulo (MOD.COD)

Nesta região localiza-se o código do núcleo do SO e do contexto de execução **MOD.HMC**. O código do núcleo do SO deve ser acessível a partir de todos os contextos de execução, uma vez que é nesta região que encontra-se o código dos serviços fornecidos ao *software* de aplicação. Os serviços do SO são geralmente executados em modo privilegiado, porém algumas porções desse código podem precisar ser executadas em modo não privilegiado. Em virtude disso, atribui-se a esta região permissões de leitura em qualquer nível de privilégio, com permissão de execução por tratar-se de uma região de código. A Tabela IV apresenta o mapeamento de permissões desta região.

Contexto de execução	Padrão
MOD.HMC	RO/RO
PAR.DEF	RO/RO
PAR.EH	RO/RO
PAR.HMC	RO/RO
PRO	RO/RO

Tabela IV. PERMISSÕES PARA A REGIÃO MOD.COD

C. Dados do módulo (MOD.DAT)

Nesta região localizam-se os dados manipulados pelo núcleo do SO e pelo contexto de execução **MOD.HMC**. Uma vez que o código do SO pode ser executado a partir de qualquer contexto de execução e assumindo que todo código do SO que manipula dados executa em modo privilegiado, atribui-se a esta região permissões de leitura e escrita apenas em modo privilegiado para todos os contextos de execução. Por manipular dados desta região e executar em modo não privilegiado, o contexto de execução **MOD.HMC** possui, ainda, permissão padrão de leitura e escrita em modo não privilegiado. Por tratar-se de uma região de dados, a execução é proibida. A Tabela V apresenta o mapeamento de permissões desta região.

Contexto de execução	Padrão
MOD.HMC	RW/RW*
PAR.DEF	RW/NO*
PAR.EH	RW/NO*
PAR.HMC	RW/NO*
PRO	RW/NO*

Tabela V. PERMISSÕES PARA A REGIÃO MOD.DAT

D. Código de partição (PAR.COD)

Armazena o código dos contextos de execução **PAR.DEF**, **PAR.EH**, **PAR.HMC** e **PRO** de uma determinada partição. Portanto, esses são os únicos contextos de execução que têm permissão de acesso a esta região, e desde que pertencentes à partição à qual a região pertence. É fornecida permissão para leitura e execução em qualquer nível de privilégio. A Tabela VI apresenta o mapeamento de permissões desta região.

Contexto de execução	Partição
PAR.DEF	RO/RO
PAR.EH	RO/RO
PAR.HMC	RO/RO
PRO	RO/RO

Tabela VI. PERMISSÕES PARA A REGIÃO PAR.COD

E. Dados de partição (PAR.DAT)

Esta região armazena os dados manipulados pelos contextos de execução **PAR.DEF**, **PAR.EH**, **PAR.HMC** e **PRO** de uma determinada partição. Por isso, esses contextos de execução têm permissão de leitura e escrita em qualquer nível de privilégio sobre ela, desde que pertencentes à partição em questão. Esta região pode ainda precisar ser restaurada pelo SO quando da reinicialização da partição associada, portanto é fornecida ainda permissão padrão de leitura e escrita para qualquer contexto de execução, desde que em modo privilegiado. Por tratar-se de uma região de dados, a execução é proibida. A Tabela VII apresenta o mapeamento de permissões da região.

Contexto de execução	Padrão	Partição
MOD.HMC	RW/NO*	-
PAR.DEF	RW/NO*	RW/RW*
PAR.EH	RW/NO*	RW/RW*
PAR.HMC	RW/NO*	RW/RW*
PRO	RW/NO*	RW/RW*

Tabela VII. PERMISSÕES PARA A REGIÃO PAR.DAT

F. Pilha de processo (PRO.STK)

Utilizada como pilha por um processo de uma determinada partição. Por utilizar esta região como pilha, o contexto de execução **PRO** do processo ao qual o contexto de execução pertence possui permissão total sobre ela. O contexto de execução **PRO** dos demais processos da partição à qual o processo pertence possuem permissão de leitura e escrita apenas em modo privilegiado, a fim de viabilizar otimizações empregadas pelos mecanismos de comunicação intrapartição, especificamente *buffers*, nos quais a cópia de mensagens pode ocorrer diretamente entre variáveis locais (alocadas na pilha) de dois diferentes processos da partição [1]. Por tratar-se de uma região utilizada como pilha, a execução é proibida. A Tabela VIII apresenta o mapeamento de permissões desta região.

Contexto de execução	Partição	Processo
PRO	RW/NO*	RW/RW*

Tabela VIII. PERMISSÕES PARA A REGIÃO PRO.STK

G. Outras regiões de pilha

São regiões utilizadas como pilha pelos contextos de execução **MOD.HMC** do módulo e **PAR.DEF**, **PAR.EH** e **PAR.HMC** de uma determinada partição. Por utilizar estas regiões como pilha, esses contextos de execução possuem permissão total sobre aquelas às quais correspondem. Para os contextos de execução **PAR.DEF**, **PAR.EH** e **PAR.HMC**, essa permissão só se aplica aos contextos de execução da partição à qual a região pertence. Por tratar-se de regiões utilizadas como pilha, a execução é proibida. A Tabela IX apresenta o mapeamento de permissões destas regiões.

Região de memória	Contexto de execução	Padrão	Partição
MOD.HMC.STK	MOD.HMC	RW/RW*	-
PAR.DEF.STK	PAR.DEF	-	RW/RW*
PAR.EH.STK	PAR.EH	-	RW/RW*
PAR.HMC.STK	PAR.HMC	-	RW/RW*

Tabela IX. PERMISSÕES PARA OUTRAS REGIÕES DE PILHA

VI. CONCLUSÃO

Neste artigo foi apresentada uma abordagem para atendimento aos requisitos de isolamento espacial (proteção de memória) exigidos pela especificação *ARINC 653* quando da utilização de processadores baseados na arquitetura *ARMv7* perfil *ARMv7-A*. Para isso foi apresentado o subconjunto mínimo de recursos da *MMU* oferecida por essa arquitetura destinado a esse fim, assim como os contextos de execução encontrados em SOs compatíveis com essa especificação, as regiões de memória que devem ser fundamentalmente gerenciadas por esses SOs e, finalmente, o mapeamento de permissões a ser utilizado para efetivação desse isolamento.

Observa-se a compatibilidade da abordagem apresentada com as definições da *ARINC 653* no que se refere ao isolamento espacial, evidenciada pelo mapeamento de permissões, que indica por exemplo que toda região de memória só pode ser escrita por contextos de execução pertencentes a no máximo uma partição [1]. A restrição de operações de leitura de memória, apesar de não ser exigida pela especificação, também é realizada por essa abordagem, o que é evidenciado pela inexistência de permissões padrão em modo não privilegiado para quaisquer contextos de execução específicos de partições, ou seja, toda partição tem acesso para leitura apenas em sua própria memória.

As maiores dificuldades encontradas na elaboração da abordagem apresentada neste artigo estão relacionadas à definição das permissões de acesso para cada região de memória em cada contexto de execução. Isso se deve à necessidade de prever as regiões de memória que devem ser acessíveis por cada um desses contextos de execução em ambos os níveis de privilégio suportados pela plataforma, ou seja, tanto durante seu processamento normal quanto durante serviços do SO invocados a partir deles. Alguns serviços do SO exigem permissões especiais para seu correto funcionamento, como é o caso dos serviços de *buffers* com relação às pilhas de processos, tornando assim necessária uma análise minuciosa do funcionamento desses serviços.

Na continuação deste trabalho será realizado o desenvolvimento de um SO compatível com a especificação *ARINC 653* para execução na plataforma *BeagleBone*, o qual terá finalidades didáticas e de experimentação para sistemas de aviônica. Na implementação desse SO será utilizada a abordagem proposta neste artigo como base para a implementação do isolamento espacial e, portanto, da detecção de erros de acesso à memória.

REFERÊNCIAS

[1] ARINC, "Avionics application software standard interface part 1 - required services (arinc specification 653p1-2)," Airlines Electronic Engineering Committee - Aeronautical Radio, Inc. (ARINC), 2551 Riva Road, Annapolis, Maryland 21401-7435, Tech. Rep., Mar. 2006.

[2] P. Prisaznuk, "Arinc 653 role in integrated modular avionics (ima)," in *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, 2008, pp. 1.E.5–1–1.E.5–10.

[3] S. VanderLeest, "Arinc 653 hypervisor," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 2010, pp. 5.E.2–1–5.E.2–20.

[4] S. Han and H.-W. Jin, "Resource partitioning for integrated modular avionics: comparative study of implementation alternatives," *Software: Practice and Experience*, 2013. [Online]. Available: http://dx.doi.org/10.1002/spe.2210

[5] R. T. E. Ltd. (2014, May) Freertos documentation. [Online]. Available: http://www.freertos.org/RTOS.html

[6] O. Corporation. (2014, May) Rtems on-line library. [Online]. Available: http://www.rtems.org/onlinedocs/doc-current/share/rtems/html/

[7] *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition*, ARM, Jul. 2012, issue C.b, ARM DDI 0406C.b (ID072512).

[8] G. Coley, *BeagleBone Rev A6 System Reference Manual*, beagleboard.org, May 2012, revision 0.0, reference BBONE_SRM.

[9] *AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual*, Texas Instruments, Oct. 2011, revised August 2013, literature number SPRUH73I.

[10] *ARM Cortex-A Series Programmers Guide*, ARM, Jan. 2014, version 4.0, issue D, ARM DEN 0013D (ID012214).

[11] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[12] R. Arpaci-Dusseau and A. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 2012. [Online]. Available: http://pages.cs.wisc.edu/ remzi/OSTEP/

[13] *Cortex-A8 Technical Reference Manual*, ARM, May 2010, revision r3p2, ARM DDI 0344K (ID060510).