Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem para a Estimação do Consumo de Energia em Modelos de Simulação Distribuída

Helder Fernando de Araújo Oliveira

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação

Elmar Uwe Kurt Melcher
(Orientador)

Joseana Macêdo Fechine Régis de Araújo
(Orientadora)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

O48a Oliveira, Helder Fernando de Araújo.

Uma abordagem para a estimação do consumo de energia em modelos de simulação distribuída / Helder Fernando de Araújo Oliveira. – Campina Grande, 2015.

137 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2015.

"Orientação: Prof. Dr. Elmar Uwe Kurt Melcher, Profa. Dra. Joseana Macêdo Fechine Régis de Araújo".

Referências.

1. Energia - Consumo - Estimação. 2. Simulação Distribuída. 3. Arquitetura de Alto Nível (HLA). 4. ESL. 5. RTL. I. Melcher, Elmar Uwe Kurt. II. Araújo, Joseana Macêdo Fechine Régis de. III. Título.

CDU 004:621.31(043)

Dedico este trabalho às pessoas que mais amo nesta vida: Minha mãe, meu pai e meu irmão.

Agradecimentos

Agradeço primeiramente a Deus, por ter me guiado e iluminado em todos os momentos de minha vida.

Agradeço em especial a minha mãe, por todo seu amor e por sempre acreditar e me incentivar a lutar para que chegasse onde estou. A meu pai, por todo seu apoio, incentivo e amor. A meu irmão por toda força, amor e motivação para que eu seguisse em frente com meus objetivos. A minha namorada Alessandra Santiago, por sua paciência, atenção e amor durante o tempo em que estamos juntos.

Aos meus orientadores, professor Elmar Melcher e professora Joseana Araújo, por sua amizade, apoio e contribuições em minha vida pessoal e acadêmica.

Ao professor e colega Alisson Brito, por toda sua ajuda durante o tempo do doutorado. Aos meus amigos e colegas que participaram da equipe do LINCS-CG, professora Edna Barros, Dalton, Fabrício, Ezequiel, Maria, Isaac, George, Henrique, Fagner, Bruno e Lenilson. Aos meus colegas do LAD e integrantes do curso de pós-graduação.

Ao professor Helmut Neff, por toda sua amizade, atenção e apoio durante o tempo em que passei estudando na Alemanha.

À equipe do KIT que me recebeu de forma muito acolhedora durante o período em que realizei o doutorado sanduíche.

Às funcionárias da COPIN e aos coordenadores do PPGCC que durante esse tempo sempre estiveram dispostos a ajudar.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de estudos concedida.

E, finalmente, a todos aqueles que, de alguma forma, contribuíram para que eu pudesse atingir meus objetivos.

Resumo

Consumo de energia é um grande desafio durante o projeto de um SoC (*System-on-a-Chip*). Dependendo do projeto, para garantir maior precisão na estimação do consumo de energia, pode ser necessário estimar o consumo de energia do sistema ou parte dele utilizando diferentes elementos: diferentes abordagens de estimação, ferramentas ou, até mesmo, modelos descritos em variadas linguagens e/ou níveis de abstração. Porém, consiste em um desafio incorporar tais elementos para criação de um ambiente de simulação distribuído e heterogêneo, o qual permita que estes se comuniquem e troquem informações de modo sincronizado. Diante do exposto, a presente pesquisa tem como objetivo desenvolver uma abordagem, utilizando-se *High Level Architecture* (HLA), a fim de permitir a criação de um ambiente de simulação distribuído e heterogêneo, composto por diferentes ferramentas e modelos. Estes modelos podem ser descritos em diversas linguagens e/ou níveis de abstração, como também podem utilizar diferentes abordagens a estimação do consumo de energia. O uso da HLA permite que os elementos que compõem este ambiente heterogêneo possam ser simulados de maneira sincronizada e distribuída. A abordagem deve proporcionar a coleta e o agrupamento de dados de estimação de consumo de energia de modo centralizado. Para realização dos estudos de caso, foi utilizado um *benchmark* composto por um conjunto escalável de MPSoC (*MultiProcessor System-on-Chip*) descrito em C++/SystemC e o arcabouço Ptolemy. Um projeto em SystemVerilog/Verilog também foi utilizado para validar a coleta de dados de estimação de consumo de energia de modelos descritos nessas linguagens, por meio da abordagem proposta. Resultados experimentais demonstraram a flexibilidade da abordagem e sua aplicabilidade para a criação de um ambiente de simulação síncrono e heterogêneo, o qual promove uma visão integrada dos dados de energia estimados.

Palavras-chave: Energia, consumo, estimação, simulação distribuída, arquitetura de alto nível, HLA, ESL, RTL, MPSocBench, Ptolemy.

Abstract

Energy consumption is a big challenge in SoC (System-on-a-Chip) design. Depending on the project requirements, to guarantee a better accuracy in power estimation, it might be necessary to estimate the power consumption of a system or part of it using different elements: different power estimation approaches, tools or, even, models described in different languages and/or abstraction levels. However, it is a challenge to incorporate these elements to create a simulation environment distributed and heterogeneous, which allows these elements to communicate and exchange information synchronously. In view of what has been exposed, the present research aims to develop an approach using HLA (High Level Architecture), enabling the creation of an environment distributed and heterogeneous, composed by different tools and models. These models can be described in different languages and/or abstraction levels, as well as use different power estimation approaches. The use of HLA enables the synchronized and distributed simulation of the elements that compose the simulation environment. The approach must allow the collecting and grouping of power estimation data in a centralized manner. As a case study, it has been used a benchmark composed of a scalable set of MPSoCs (*MultiProcessor System-on-Chip*) which is described in C++/SystemC and the Ptolemy framework. A project in SystemVerilog/Verilog was also used to validate the power estimation data collected from models described in these languages, through the proposed approach. The experimental results show the approach flexibility and its applicability on creation of a distributed and synchronous simulation environment, which promotes an integrated view of power estimation data.

Keywords: Power estimation, distributed simulation, High Level Architecture, HLA, ESL, RTL, MPSocBench, Ptolemy.

Sumário

Lista de Siglas e Abreviações

AOP - *Aspect-Oriented Programming.*
API - *Application Programming Interface.*
CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior.
CI - Circuito Integrado.
CMOS - *Complementary Metal-Oxide-Semiconductor.*
DAAD - *Deutscher Akademischer Austausch Dienst.*
DDM - *Data Distribution Management.*
DPM - *Dynamic Power Management.*
DM - *Declaration Management.*
DMSO - *Defense Modeling and Simulation Office.*
DoD - *US Department of Defence.*
DUV - *Design Under Verification.*
DVFS - *Dynamic Voltage and Frequency Scaling.*
EDA - *Electronic Design Automation.*
EMDMA - *Enhanced Multimedia Direct Memory Access.*
ESL - *Electronic System Level.*
FDD - *FOM Document Data.*
FED - *Federation Execution Data.*
FIFO - *First In First Out.*
FOM - *Federation Object Model.*
FPGA - *Field-Programmable Gate Array.*
FSDB - *Fast Signal Database.*
GALT - *Greatest Available Logical Time.*
HDL - - *Hardware Description Language.*
HLA - *High Level Architecture.*
IEEE - *Institute of Electrical and Electronics Engineers.*
IP - *Intellectual Property.*
ISA - *Instruction Set Architecture.*
KIT - *Karlsruhe Institute of Technology.*
LAD - Laboratório de Arquiteturas Dedicadas.
LBTS - *Lower Bound Time Stamp.*
LITS - *Least Incoming Time Stamp.*
LUT - *Look-Up Table.*
MDE - *Model Driven Engineering.*
MIPS - *Microprocessor without Interlocked Pipeline Stages.*
MNET - *Minimum Next Event Time.*

MPSoC - *MultiProcessor System-on-Chip*.
OMT - *Object Model Template.*
PAC - *Parallel Architecture Core.*
PGFB - *Power Gated Functional Block.*
RO - *Receive Order.*
RSSF - Redes de Sensores Sem Fio.
RSoC - *Reconfigurable System-on-a-chip.*
RTI - *Run-Time Infrastructure.*
RTL - *Register Transfer Level.*
SoC - *System-on-a-Chip.*
SOM - *Simulation Object Model.*
SPARC - *Scalable Processor ARChitecture.*
TLM - *Transaction-Level Modeling.*
TSO - *Time Stamped Order.*
UFCG - Universidade Federal de Campina Grande.
UVM - *Universal Verification Methodology Connect.*
V2SC - *VHDL-to-SystemC-Converter.*
XML - *EXtensible Markup Language.*

Lista de Códigos

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de Registros

Capítulo 1

Considerações Iniciais

Neste capítulo, será apresentada uma introdução para contextualização inicial da pesquisa, assim como a motivação para seu desenvolvimento. Também será apresentada a definição do problema, serão apresentados os objetivos da pesquisa e suas contribuições.

1.1. Introdução

O consumo de energia se torna, cada vez mais, uma das principais preocupações durante o projeto de um SoC (*System-on-a-Chip*). Atualmente, diversas pesquisas (AHUJA, 2010; BOUHADIBA; MOY; MARANINCHI, 2013; CONTI et al., 2011; GIAMMARINI; CONTI; ORCIONI, 2011; HSIEH; YEH; HUANG, 2010; KLEIN et al., 2007a; KUEHNLE; WAGNER; BECKER, 2011; LIU et al., 2010; MCCULLOUGH et al., 2011; OST et al., 2009; RETHINAGIRI et al., 2014a, 2014b; TRABELSI et al., 2011) são dedicadas à estimação de consumo de energia em projetos de SoC. Essas pesquisas operam em diferentes níveis de abstração e abordam diferentes técnicas de estimação (TRABELSI et al., 2011).

Dentre os níveis de abstração, podem ser citados o ESL (*Electronic System Level*), o RTL (*Register Transfer Level*), o nível de portas lógicas (*gate level*) e o nível de *transistores* (*transistor-level*) (CALDARI et al., 2003). Os projetistas devem considerar a questão de energia o mais cedo possível para reduzir os ciclos de projeto (HSIEH; YEH; HUANG, 2010).

Para um nível de abstração mais alto, os projetistas podem obter informação de energia consumida no início da fase de projeto, sendo menos oneroso melhorá-lo na visão de *hardware* ou *software* do que mais tarde no ciclo de projeto. Além disso, especificações em alto nível permitem ao projetista ignorar várias questões de codificação RTL, tais como sincronização e agendamento de operações, enquanto captura funcionalidades do projeto alvo (AHUJA, 2010).

Neste contexto, metodologias de estimação de consumo de energia no nível de sistema (ESL) têm se tornado um tópico de pesquisa relevante (AHUJA, 2010; HSIEH; YEH; HUANG, 2010; KUEHNLE; WAGNER; BECKER, 2011). Diversas metodologias e ferramentas ESL (AHUJA, 2010; BELTRAME; SCIUTO; SILVANO, 2007; CONTI et al., 2011; GIAMMARINI; CONTI; ORCIONI, 2011; HSIEH; YEH; HUANG, 2010; IKHWAN et al., 2006; KLEIN et al., 2007b; MCCULLOUGH et al., 2011; OST et al., 2009; RETHINAGIRI et al., 2014b, 2014a; TRABELSI et al., 2011) vêm sendo desenvolvidas nos últimos anos. Muitas dessas metodologias focam na relação entre sua precisão e seu custo computacional.

1.2. Motivação

Devido à complexidade dos projetos e à pressão devido ao tempo entre a análise do produto e sua disponibilização para a venda (*time-to-market*), nem sempre é possível realizar a estimação de consumo de energia no nível RTL (*Register Transfer Level*) ou no nível de portas lógicas para um sistema inteiro, visto que, nesses níveis, o tempo requerido pelas simulações é frequentemente longo.

Assim, diante da preocupação atual com o consumo de energia em dispositivos eletrônicos, tornou-se importante estimá-lo o mais cedo possível dentro do ciclo de desenvolvimento de projetos de circuitos digitais.

Diversos autores (HSIEH et al., 2011; MARIANI et al., 2009; RETHINAGIRI et al., 2014a; SCHÜRMANS et al., 2013; TRABELSI et al., 2011) concordam que o principal desafio para ferramentas de estimação de consumo de energia no nível de sistema é alcançar melhor equilíbrio entre desempenho da simulação e sua precisão.

Nos últimos anos, diversas pesquisas se propõem a prover uma estimativa de consumo de energia no nível de sistema (AHUJA, 2010; BANSAL et al., 2005; CALDARI et al., 2003; DAMAŠEVIČIUS, 2006; GIAMMARINI; CONTI; ORCIONI, 2011; GREAVES; YASIN, 2014; HSIEH; YEH; HUANG, 2010; IKHWAN et al., 2006; KLEIN et al., 2007a; KUEHNLE; WAGNER; BECKER, 2011; PARK et al., 2007; SCHÜRMANS et al., 2013; TRABELSI et al., 2011).

Dependendo do projeto, para garantir maior precisão na estimação do consumo de energia, pode ser necessário estimar o consumo de energia do sistema ou parte dele utilizando diferentes elementos: diferentes abordagens de estimação, ferramentas ou, até mesmo, modelos descritos em variadas linguagens e/ou níveis de abstração.

Porém, consiste em um desafio incorporar tais elementos para criação de um ambiente de simulação heterogêneo e distribuído, o qual permita que estes se comuniquem e troquem informações de modo sincronizado.

1.3. Definição do Problema

A arquitetura de alto nível (*High Level Architecture – HLA*) (IEEE COMPUTER SOCIETY, 2010a) fornece meios para que aplicações/ferramentas/simulações troquem informações de maneira sincronizada. A HLA é definida como uma arquitetura comum para modelagem e simulação distribuída. Ela define uma abordagem integrada que provê um *framework* comum para interconexão de simulações interativas (IEEE COMPUTER SOCIETY, 2010b).

Em nenhuma das pesquisas relacionadas sobre estimação de consumo de energia (apresentadas no Capítulo 3) foram encontrados relatos sobre o uso de uma arquitetura de propósito geral para modelagem e simulação distribuída, tal qual a HLA, a fim de permitir a criação de um ambiente de simulação distribuído e heterogêneo, composto por diferentes ferramentas, modelos descritos em diferentes linguagens e/ou níveis de abstração, que proporcione o uso de diferentes abordagens de estimação de consumo de energia para cada modelo durante o processo de simulação.

Em uma aplicação compatível com a arquitetura de alto nível, qualquer número de simulações fisicamente distribuídas pode ser reunido dentro de um ambiente de simulação unificado para atender a necessidade de novas aplicações.

A criação de uma abordagem baseada na HLA (IEEE COMPUTER SOCIETY, 2010a, 2010b) proporcionará:

- Simulação combinada e sincronizada de modelos em diferentes níveis de abstração (ESL, RTL, etc.);
- Uso de diferentes abordagens de estimação de consumo de energia para cada modelo durante o processo de simulação.
- Reuso de modelos descritos em diversas linguagens (e.g., C++, SystemC, SystemVerilog, Verilog, Java, etc.);
- Uso sistemático, combinado e sincronizado de diversas ferramentas (e.g., simulador SystemC, ferramentas de simulação para modelos SystemVerilog/Verilog, Ptolemy[1], etc.).

1.4. Objetivos da Pesquisa

Nesta seção, será apresentado o objetivo geral da pesquisa, assim como seus objetivos específicos. Na seção 1.5, serão apresentadas as principais contribuições da pesquisa ora descrita.

1.4.1. Objetivo Geral

A presente pesquisa tem como objetivo geral desenvolver uma abordagem utilizando *High Level Architecture* (HLA) para permitir a criação de um ambiente de simulação distribuído e heterogêneo, composto por diferentes ferramentas e modelos. Estes modelos podem ser descritos em diversas linguagens e/ou níveis de abstração, como também utilizar diferentes abordagens a estimação do consumo de energia. O uso da HLA permite que os elementos que compõe este ambiente heterogêneo possam ser simulados de maneira sincronizada e distribuída. A abordagem deve proporcionar a coleta e o agrupamento de dados de estimação de consumo de energia de modo centralizado.

[1] Ptolemy (BERKELEY EECS, 2015b) é um *framework* de código aberto que suporta experimentos com projetos orientados a atores. Atores são componentes de *software* que executam concorrentemente e se comunicam por meio de mensagens enviadas via portas interconectadas.

1.4.2. Objetivos Específicos

Em relação aos objetivos específicos, a abordagem deve considerar os aspectos descritos a seguir.

- Viabilizar a comunicação e troca de dados entre os modelos, de maneira sincronizada e distribuída.
- Permitir o uso de modelos descritos em diferentes níveis de abstração (ESL ou RTL) e linguagens (C++, SystemC, SystemVerilog, Verilog, Java), os quais poderão utilizar diferentes abordagens de estimação de consumo de energia e ferramentas durante o processo de simulação.
- Viabilizar a coleta de dados de estimação de consumo de energia durante a simulação distribuída de maneira centralizada.
- Adotar a Arquitetura de Alto Nível (HLA) no intuito de viabilizar a troca de dados entre os modelos, assim como envio de informações de consumo de energia durante a simulação. O uso da HLA permitirá que, em pesquisas futuras, a estratégia possa ser estendida para agregação de simulações de modelos descritos em outros níveis de abstração (e.g., nível de portas lógicas) e linguagens (e.g., VHDL).
- Definir uma arquitetura para a implementação da estratégia proposta contemplando o uso da HLA.
- Ser validada por meio de estudos de caso.
    - Utilizar a plataforma MPSoCBench (DUENHA et al., 2014) para construção de modelos C++/SystemC de MPSoC como parte dos estudos de caso.
    - Criar um modelo de simulação Ptolemy que se comunique, via abordagem desenvolvida, com um modelo de um MPSoC construído por meio da plataforma MPSoCBench, de modo que ambos os modelos possam trocar dados e se comunicarem de maneira sincronizada.
    - Escolher e utilizar, dentre os métodos e técnicas investigados para estimação de consumo de energia em projetos de circuitos digitais, alguma(s) que se adéque(m) aos estudos de caso.

- Utilizar um modelo RTL de um DPCM (*Differential Pulse-CodeModulation*), descrito em SystemVerilog/Verilog, demonstrando a viabilidade da simulação de modelos descritos nessas linguagens por meio da abordagem desenvolvida.

1.5. Contribuições

A pesquisa ora descrita apresenta uma abordagem que permite distribuir e simular diferentes modelos, de maneira sincronizada, viabilizando comunicação entre os mesmos, como também o envio de dados de consumo de energia para uma unidade coletora. Esses modelos podem ser descritos em diferentes níveis de abstração e/ou diferentes linguagens, e podem utilizar diferentes ferramentas de simulação e metodologias de estimação de consumo de energia. A abordagem promove a coleta e agrupamento de dados de estimação de consumo de energia de maneira centralizada durante o processo de simulação.

Para cada componente monitorado (e.g., um módulo funcional) de cada um dos modelos é necessário a inserção de linhas de código responsáveis pela chamada de métodos para o envio de dados intrínsecos ao modelo, assim como a inserção de linhas de código para chamada de métodos responsáveis por acumular valores de energia durante o tempo de simulação do modelo. Visto que os métodos para comunicação entre os modelos são específicos para cada um deles, estes métodos deverão ser definidos pelo usuário da abordagem de acordo com cada modelo simulado. Cada modelo simulado terá um elemento responsável por coordenar a sincronização e enviar de dados de energia.

Outro aspecto inovador da pesquisa é a utilização da arquitetura de alto nível (IEEE COMPUTER SOCIETY, 2003, 2010a, 2010b, 2010c) como meio de distribuição e comunicação entre as simulações, garantindo uma maneira consolidada e bem documentada para sua extensibilidade. Nenhuma das pesquisas correlatadas avaliadas (ver Capítulo 3) utiliza HLA como meio de interconexão para distribuir e simular seus modelos.

Conforme já mencionado, dependendo do projeto, para garantir maior precisão na estimação do consumo de energia, pode ser necessário estimar o consumo de energia do sistema ou parte dele utilizando diferentes

elementos: diferentes abordagens de estimação, ferramentas ou, até mesmo, modelos descritos em variadas linguagens e/ou níveis de abstração.

O modo como a abordagem foi desenvolvida permite que ela possa facilmente ser entendida para trabalhar com outras linguagens e também outras ferramentas (e.g., simuladores). Pesquisas futuras poderão ser realizadas no âmbito de extensão da pesquisa ora apresentada.

A abordagem foi validada por meio de estudos de caso de modelos MPSoC descritos em C++/SystemC, assim como por um estudo de caso de um DPCM (*Differential Pulse-Code Modulation*) descrito em SystemVerilog/Verilog. Os resultados experimentais obtidos demonstram a flexibilidade e sua aplicabilidade na distribuição e simulação sincronizada de diferentes modelos *Electronic System Level* (ESL), como também modelos *Register Transfer Level* (RTL), provendo uma visão unificada dos dados de energia estimados.

1.6. Organização do Documento

O restante deste documento foi estruturado conforme descrição a seguir.

Capítulo 2 – Neste capítulo, são apresentados conceitos fundamentais para o entendimento da pesquisa realizada: conceitos sobre estimação de consumo de energia, SystemC, SystemVerilog e arquitetura de alto nível.

Capítulo 3 – Neste capítulo, são apresentadas as pesquisas relacionadas ao tema proposto.

Capítulo 4 – Neste capítulo, a abordagem desenvolvida é apresentada em detalhes.

Capítulo 5 – Neste capítulo, são apresentados os resultados obtidos durante o desenvolvimento e avaliação da abordagem desenvolvida.

Capítulo 6 – Neste capítulo, são apresentadas as considerações finais sobre a pesquisa, limitações e sugestões para pesquisas futuras.

Capítulo 2

Fundamentação Teórica

Neste capítulo, serão apresentados os principais conceitos para contextualização da pesquisa: técnicas para a estimação do consumo de energia, da arquitetura de alto nível e da plataforma MPSoCBench.

2.1. Contextualização

A indústria de semicondutores tem investido fortemente no desenvolvimento de sistemas complexos em um único chip, conhecidos como SoC (*System-on-a-Chip*), bastante empregados em dispositivos portáteis. Com os diversos recursos adicionados aos SoC, ocorreu um aumento da complexidade no fluxo de desenvolvimento e um aumento do consumo energético destes sistemas (GIRARD, NICOLICI, WEN, 2009). Nos últimos anos, aumentou a importância da restrição da energia consumida pelos dispositivos eletrônicos, em especial dos SoC (TRABELSI et al., 2011; YANG et al., 2015).

Diante disto, estimação de consumo de energia tem sido um tópico com grande ênfase no fluxo de desenvolvimento de projetos de circuitos digitais (ANDERSON; MEMBER; NAJM, 2004; BENINI; HODGSON; SIEGEL, 1998; DING; TSUI; PEDRAM, 1998; GREAVES; YASIN, 2014; HSIEH; YEH; HUANG, 2010; IKHWAN et al., 2006; KLEIN et al., 2007a; KUEHNLE; WAGNER; BECKER, 2011; RETHINAGIRI et al., 2014a, 2014b). A estimação pode ser necessária nos diversos níveis de abstração.

Dentre os níveis de abstração, podem ser citados o ESL (*Electronic System Level*), o RTL (*Register Transfer Level*), o nível de portas lógicas (*gate level*) e o nível de *transistores* (*transistor-level*). Porém, estimar o consumo de energia de dispositivos o quanto antes no fluxo do projeto (i.e., nos níveis de abstração mais altos) possibilita uma análise antecipada e de menor custo das funcionalidades do sistema que são consideradas mais críticas sob o ponto de vista energético (CALDARI et al., 2003;

RETHINAGIRI et al., 2014a). Assim, pode-se concentrar os esforços na melhoria desses pontos durante o fluxo de desenvolvimento do projeto visando à obtenção dos melhores resultados em termos de consumo de energia.

No decorrer dos últimos anos, diversos trabalhos (AHUJA, 2010; CALDARI et al., 2003; DAMAŠEVIČIUS, 2006; GIAMMARINI; CONTI; ORCIONI, 2011; GREAVES; YASIN, 2014; HSIEH; YEH; HUANG, 2010; IKHWAN et al., 2006; KLEIN et al., 2007a, 2007b; KUEHNLE; WAGNER; BECKER, 2011; LIU et al., 2010; PARK et al., 2007; TRABELSI et al., 2011) se propõem a prover uma estimativa de consumo de energia no nível de sistema, também conhecido como ESL.

O ESL é considerado um alto nível de abstração, no qual o comportamento de um sistema pode ser modelado utilizando-se linguagens de alto nível como, por exemplo, C++. A maioria dos trabalhos realizados no âmbito da estimação do consumo de energia no nível de sistema utiliza SystemC como mecanismo de especificação.

A seguir, serão apresentados os principais conceitos para contextualização da pesquisa, assim como uma visão geral das pesquisas relacionadas.

2.2. Estimação de Consumo de Energia

A partir da década de 90, houve uma preocupação crescente relacionada ao aumento do consumo de energia dos circuitos digitais (DING; TSUI; PEDRAM, 1998; LANDMAN, 1996; PEDRAM, 1996, 1997). A restrição do consumo de energia sobre os dispositivos eletrônicos aumentou a medida que a tecnologia do processo de fabricação dos circuitos eletrônicos avançou (HUANG et al., 2010; KEATING et al., 2007; MATTOS; JR; PILLA, 2009; MOHANTY et al., 2008; TRABELSI et al., 2011).

São várias as razões para buscar a redução da energia consumida pelos SoC: Redução do desgaste térmico; redução de custos com resfriamento do dispositivo eletrônico; aumento do tempo de funcionamento com uma carga da bateria; entre outras (HUANG et al., 2010; JIANG et al., 2009; KAWA, 2008; LANDMAN, 1996; MOHANTY et al., 2008).

Uma razão diretamente ligada ao meio ambiente é o impacto causado pelo processo de geração de energia para suprir as necessidades dos

dispositivos eletrônicos. Esse processo tem como principal consequência a emissão de gases na atmosfera (*carbon footprint*[2]), podendo acelerar o processo do efeito estufa e da poluição do ar.

O consumo da energia total de um circuito integrado é um somatório do consumo da energia dinâmica e da energia estática (ABRIL et al., 2005; KEATING et al., 2007). Técnicas para reduzir a potência dinâmica[3] (*dynamic power*) se tornaram comum. Com tecnologias de processo CMOS inferiores a 90 nm, a potência estática[4] (*leakage power*) se tornou relevante e, em muitos casos, se tornou uma restrição dominante no projeto (IEEE COMPUTER SOCIETY, 2009).

Técnicas para redução do consumo de energia podem ser aplicadas em diversos níveis de abstração, durante o desenvolvimento de um SoC, desde a concepção e especificação do sistema até a fase da geração do *layout* (KEATING et al., 2007). Uma síntese das principais técnicas para redução de consumo de energia pode ser encontrada em Silveira (2011).

2.2.1. Níveis de Abstração para Estimação de Consumo de Energia

Assim como as técnicas para redução de consumo de energia, a estimação de consumo de energia em SoC pode ser realizada em diversos níveis de abstração. De acordo com Talarico et al., (2005), a energia consumida por um circuito digital pode ser estimada em quatro diferentes níveis de abstração: nível de transistores (*transistor-level*), nível de portas (*gate-level*), RTL (*Register Transfer Level*) e nível de sistema ou ESL (*Electronic System Level*). A seguir, será apresentado um breve resumo de cada uma delas.

[2] *Carbon footprint* é a quantidade total de gases de efeito estufa produzido diretamente ou indiretamente por atividades humanas. Geralmente, é expressa em toneladas equivalentes de dióxido de carbono (GOMBINER, 2011).

[3] Potência dinâmica é energia consumida durante o chaveamento do transistor (KIM et al., 2003).

[4] Potência estática é energia desperdiçada por causa das correntes de "fuga" (KIM et al., 2003).

2.2.2. Nível de Transistores

No nível de transistores (ou nível de circuito), a estimação de consumo pode ser calculada a partir das correntes elétricas que percorrem os transistores, o que requer uma descrição do SoC no nível de transistores (TRABELSI et al., 2011). A diminuição do consumo de energia neste nível pode ser realizada pelo redimensionamento do transistor e rearranjo do *layout* dependendo das estimativas obtidas. Estimativas de consumo neste nível podem ser obtidas por meio de simuladores de circuitos, como por exemplo, SPICE (BERKELEY EECS, 2015a).

Dentre os níveis de abstração, o nível de transistores é o que fornece maior precisão nas estimativas de consumo do circuito digital, porém simulações nesse nível consomem muito tempo e, normalmente, não são aplicadas a um SoC inteiro, mas apenas a partes dele, por exemplo, *standard cells*. Uma *Standard cell* é um grupo de transistores e estruturas interconectadas que fornece uma função lógica booleana, como por exemplo, AND, OR, XOR, XNOR, inversores, ou uma função de armazenamento, tal como *flip-flop* ou *latch* (ZHANG, 2010).

2.2.3. Nível de Portas

No nível de portas lógicas a estimação de consumo é baseada nos modelos de consumo de energia[5] (*power models*) das células (*standard cells*), de acordo com sua tecnologia de fabricação (TRABELSI et al., 2011). A potência consumida por uma célula está correlacionada com seus dados de entrada. Também depende de diversos outros parâmetros como, por exemplo, tensão de alimentação e frequência.

De acordo com Trabelsi *et al.* (2011), diminuir o consumo de energia de um sistema inteiro neste nível requer muitos experimentos, a fim de otimizar os parâmetros de consumo. O projeto é simulado no nível de portas lógicas e a potência é calculada utilizando atividades de chaveamento e capacitância do nó (DAMAŠEVIČIUS, 2006; TALARICO et al., 2005). Na década de 90, para calcular o consumo de energia neste nível, ainda eram ignoradas correntes de curto-circuito, corrente de fuga e outros

[5] Um modelo de consumo de energia pode ser definido como um modelo que captura a dependência de dissipação de potência de um bloco do projeto sobre certos parâmetros, tal como atividade de chaveamento, capacitância, etc. (AHUJA, 2010).

efeitos considerados insignificantes em termos de energia (BRAND; VISWESWARIAH, 1996).

Como já mencionado, com tecnologias de processo CMOS inferiores a 90 nm a potência estática (*leakage power*) se tornou relevante e, em muitos casos, até passou a ser uma restrição dominante no projeto (IEEE COMPUTER SOCIETY, 2009). Realizar estimação de consumo de energia no nível de portas é mais rápido do que executá-la no nível de transistores (BRAND; VISWESWARIAH, 1996; TALARICO et al., 2005).

São duas as técnicas mais difundidas para estimação de consumo de energia no nível de portas: dinâmica (ou simulativa) e estática (ou não-simulativa) (ANDERSON; MEMBER; NAJM, 2004; DING; TSUI; PEDRAM, 1998; KUEHNLE; WAGNER; BECKER, 2011). De acordo com o autor, técnicas de estimação dinâmicas simulam explicitamente o circuito sob um fluxo "típico" de vetores de entrada. A principal deficiência dessa técnica é sua lentidão em comparação com a estimação estática. Além disto, os resultados são muito dependentes da sequência simulada.

Para produzir estimativas de potência significativas, faz-se necessário um número grande de vetores simulados. As técnicas de estimação estáticas contam com informação estatística sobre o vetor de entrada para estimar a atividade de chaveamento interna do circuito. Estas informações podem ser, por exemplo, a média das atividades dos sinais de entrada e suas correlações.

2.2.4. *Register Transfer Level* (RTL)

No *Register Transfer Level* (RTL) os sistemas são descritos usando blocos mais abstratos como multiplicadores, somadores, controladores. De acordo com Ahuja (2010), na indústria, estimação de consumo de energia com precisão é realizada, em sua maioria, no nível de abstração RTL e no nível de portas. Estimação de consumo de energia no nível RTL requer geralmente as seguintes entradas (AHUJA, 2010):

- Uma descrição em uma linguagem de descrição de *hardware* (*hardware description language*);

- Registros da simulação RTL no formato VCD[6] (*Value Change Dump*), *Fast Signal Database*[7] (FSDB), ou outro;
- Bibliotecas com caracterização de potência (*power characterized libraries*), como, por exemplo, bibliotecas de potência das *standard cell*.

Landman (1996) e Schürmans et al.(2013) afirmam que a dificuldade em estimar energia neste nível deriva-se do fato de que os detalhes no nível de porta, circuito e *layout* podem não ter sido especificados. Além disto, o *floorplan*[8] do circuito integrado pode não estar disponível, o que torna difícil a análise de interconexões e rede de distribuição de *clock*. De acordo com Trabelsi *et al*. (2011), a estimação de consumo de energia no nível RTL é em torno de 10 vezes mais rápida que no nível de portas lógicas.

2.2.5. *Electronic System Level* (ESL)

O *Electronic System Level* (ESL) é tipicamente definido acima do RTL. Quando aplicado a projetos de *hardware*, pode ser definido como um processo de descrição das funcionalidades do *hardware* em alto nível de abstração para aumentar a produtividade do projetista e permitir maior grau de exploração (BAILEY; MARTIN; PIZIALI, 2007).

Os detalhes da comunicação entre os módulos são separados da implementação das unidades funcionais ou da arquitetura de comunicação. Mecanismos de comunicação como FIFO (*First In First Out*) ou barramentos são modelados como canais (GRÖTKER et al., 2002).

No nível de sistema, a ênfase é à funcionalidade da transferência dos dados e não à implementação, tornando mais fácil para o projetista, no nível de sistema, a tarefa de realizar a exploração arquitetural. De acordo com Caldari et al., (2003), o processo de estimação de consumo de energia no ESL não precisa necessariamente conduzir a um alto grau de precisão, visto a dificuldade de atingi-lo neste nível.

[6] *Value Change Dump* ou VCD é um arquivo ASCII comumente gerado por ferramentas de simulação capturando mudanças de valores nas variáveis selecionadas em uma simulação. O formato VCD é definido pelo padrão IEEE 1364-2001.

[7] *Fast Signal Database* ou FSDB é um arquivo que captura e armazena resultados de simuladores, emuladores e ferramentas formais que produzem sequências de tempo/valor.

[8] *Floorplan* de um circuito integrado é uma representação esquemática da alocação seus principais blocos funcionais em uma determinada área.

A meta consiste em obter uma indicação antecipada e de menor custo das porções do projeto que são consideradas mais críticas sobre o ponto de vista energético. Desta maneira, podem ser concentrados esforços na melhoria desses pontos durante o fluxo de desenvolvimento do projeto para obter resultados requeridos pela especificação em termos de consumo de energia.

Geralmente, a estimação de consumo de energia no ESL é baseada em simples descrições do sistema no alto nível utilizando modelos abstratos de capacitância e chaveamento.

2.2.6. Técnicas para a Estimação do Consumo de Energia ESL

O principal desafio para ferramentas de estimação de consumo de energia é alcançar um melhor equilíbrio entre desempenho e precisão (HSIEH et al., 2011; MARIANI et al., 2009; RETHINAGIRI et al., 2014a; SCHÜRMANS et al., 2013; TRABELSI et al., 2011). Para reduzir o consumo de energia em um produto final de um projeto de *hardware* é vantajoso que possa ser realizada a estimação de consumo de energia desde o início do fluxo do projeto (TRABELSI et al., 2011).

Ferramentas para estimação de consumo de energia nos níveis mais baixos de abstração permitem modelagem de energia precisa, porém resultam em baixo desempenho de simulação e alto custo em torno do projeto (HSIEH; YEH; HUANG, 2010; LIU et al., 2010).

Métodos para estimação de consumo de energia em um baixo nível de abstração (por exemplo, nível de *layout*, nível de portas e RTL) levam em conta muitos detalhes do SoC simulado, tornando as simulações lentas e, consequentemente, aumentando o tempo do projeto. A lentidão desses métodos pode ser considerada um obstáculo para produtividade (TRABELSI et al., 2011). Desta maneira, técnicas de estimação em um nível de abstração mais alto (por exemplo, ESL) se fazem necessárias.

As técnicas de estimação de consumo de energia ESL executam suas estimativas baseadas em descrições de sistema no alto nível de abstração utilizando modelos de consumo de energia (DAMAŠEVIČIUS, 2006).

Um modelo de consumo de energia (*power model*) pode ser definido como um modelo que captura a dependência de dissipação de potência de

um bloco do projeto sobre certos parâmetros, tais como atividade de chaveamento, capacitância, etc. Sua precisão é muito dependente do modelo de computação[9], atividade de entra/saída, capacitância, etc. (AHUJA, 2010). De acordo com Hsieh, Yeh e Huang (2010), existem dois tipos de metodologia para implementação dos modelos de consumo de energia: a metodologia *top-down* e a metodologia *bottom-up*. O tipo de metodologia a ser utilizado dependerá se o projeto do circuito integrado está pronto ou não.

A metodologia *top-down* é comumente utilizada quando o sistema está em desenvolvimento. Nela, os modelos de consumo de energia são geralmente criados a partir de máquinas de estados que representam os possíveis estados de operação[10] do circuito em questão. Para cada estado e transições de estado é determinado um valor de potência. Embora o projeto ainda não esteja pronto, o uso deste tipo de valor de potência relativa pode ajudar o projetista a melhorar o sistema em termos de consumo de energia.

Na metodologia *bottom-up*, os modelos de consumo de energia podem ser construídos precisamente utilizando um processo de simulação de energia de baixo nível de abstração, denominado processo de caracterização. Uma vez realizado o processo, o modelo de consumo de energia pode ser utilizado para fornecer um valor de potência do circuito de acordo com seu estado de operação.

O processo de caracterização (ou de caracterização de potência) pode ser definido como um processo de atribuição de valores de potência para algum comportamento, componente, operação, etc., do bloco do projeto de *hardware,* via medições diretas viabilizadas por instrumentação física ou por meio de modelagem baseada nos contadores de desempenho do *hardware* (MCCULLOUGH et al., 2011).

[9] Modelo de computação (MoC) é uma definição formal do conjunto de operações permitidas usadas em uma computação e seus respectivos custos. Isso define o custo de um sistema em certo nível de abstração para refletir as características essenciais do sistema (CHEN; DOEMER; CENTER, 2009).

[10] Um estado de operação do circuito representa o estado o qual ele se encontra. Por exemplo, se o circuito em questão for um barramento, seu estado pode ser ativo ou inativo. Caso seja uma memória, seu estado pode ser inativo, escrita e leitura. O estado de operação vai depender do tipo de IP em questão.

A partir dos valores mensurados são criados os macromodelos. De maneira formal, um macromodelo pode ser definido como um modelo abstrato obtido pela medição da dissipação de potência de implementações existentes com a ajuda de métodos em baixo nível (KLEIN et al., 2007b; LIU et al., 2010). Macromodelagem de consumo energia é derivada da estimação de consumo de energia no RTL (LIU; PAPAEFTHYMIOU, 2004; MACII, 1998).

Um resumo de técnicas para macromodelagem pode ser visto no trabalho de Macii (1998). De acordo com Liu e Papaefthymiou (2004) a ideia básica por trás da macromodelagem é gerar um mapeamento entre a dissipação de potência de um circuito e certas estatísticas de seus sinais de entrada.

Os mesmos autores afirmam que o processo de macromodelagem pode ser dividido em duas etapas: etapa de caracterização e etapa de avaliação. Na fase de caracterização, o circuito é simulado sob amostras de fluxos de entrada com várias estatísticas de sinais para obter valores de dissipação de potência. O mapeamento é utilizado na fase de avaliação para predizer a dissipação de potência do circuito baseada em estatísticas de seus sinais de entrada atuais.

A maioria das técnicas de estimação de consumo de energia utilizada pelas ferramentas EDA (*Electronic Design Automation*) é baseada no conceito de macromodelos (KLEIN et al., 2007b). Klein et al. (2007b) também afirma que, comumente, as propriedades do macromodelo de consumo de energia são armazenadas em LUT (*Look-Up Table*).

De acordo com Liu et al. (2010), a técnica de macromodelagem de consumo de energia tem atraído considerável atenção para estimação de consumo de energia em alto nível de abstração. Porém, cada técnica de macromodelagem faz algumas suposições as quais deixam algum tipo de limitação intrínseca, afetando sua precisão.

Os mesmos autores afirmam que a seleção adequada de um conjunto de macromodelos suportados poderia ser vista como uma forma de aperfeiçoar a precisão da estimação de consumo de energia total. Esta técnica é denominada de estimação de consumo de energia multi-modelos (*multi-models power estimation*). Também é destacada a ideia de estimação de consumo de energia multi-precisão (*multi-accuracy power estimation*).

Esta última abordagem fornece diferentes combinações de modelos de consumo de energia que variam em relação à sua precisão. Com o uso da estimação de consumo de energia multi-precisão é possível a troca desses modelos durante a simulação, em tempo de execução, permitindo ao projetista decidir entre precisão de estimação dos valores de consumo de energia e desempenho da simulação. Isto é particularmente útil durante a fase de exploração de um projeto, quando o projetista altera as características ou os parâmetros do projeto, tentando satisfazer às suas restrições.

Se o projetista modifica as características ou parâmetros do projeto, geralmente somente partes limitadas do sistema são afetadas pela mudança de um único parâmetro (BELTRAME; SCIUTO; SILVANO, 2007). Neste sentido, a ideia dessa técnica consiste em não estimar partes do sistema que não são afetadas pela mudança de um dado parâmetro (LIU et al., 2010).

Beltrame, Sciuto e Silvano (2007) afirmam que uma estimativa aproximada (ou uma não estimativa dessas partes) pode tornar as simulações mais rápidas e com precisão relativamente boa. A precisão da estimação de consumo de energia é aumentada quando existe uma análise das entradas de dados para predizer a frequência de chaveamento sobre a estrutura de *hardware* sintetizada esperada (KÜHNLE *et al.*, 2011).

Essencialmente, técnicas de macromodelagem podem ser divididas em duas categorias: sensível à atividade (*activity-sensitive*) e sensível à transição (*transition-sensitive*) (KLEIN et al., 2007b). De acordo com Landman (1996) e Klein et al. (2007b), a macromodelagem sensível à atividade leva em consideração a influência das estatísticas da atividade dos dados na dissipação de potência. Em outras palavras, assume que os dados de entrada afetam a atividade do componente, tendo assim influência na dissipação de potência.

Nesta categoria de macromodelagem, durante a simulação RTL, utilizando valores de entrada típicos, a atividade dos sinais é monitorada e uma análise de dados é realizada, a fim de gerar estatísticas. Essas estatísticas são, então, usadas para alimentar os modelos de consumo de energia de cada componente (e.g., memória, unidade de controle), permitindo, assim, uma estimativa da potência total.

A macromodelagem sensível à transição é baseada nas transições de entradas atuais, em oposição à macromodelagem sensível à atividade, a qual depende das estatísticas sobre as entradas. A macromodelagem sensível à transição assume que os dados de entrada não afetam a atividade do componente, ou seja, o valor da potência é uma constante independente da entrada.

Muitas pesquisas focam em técnicas de modelagem de consumo de energia para componentes individuas de um sistema como, por exemplo, processadores, memórias, barramentos, periféricos, lógica definida pelo usuário, etc (BANSAL et al., 2005). Esses modelos de consumo de energia podem ser integrados dentro de *frameworks* de simulação no nível de sistema para fornecer meios de estimação de consumo de energia.

De acordo com Bansal *et al.* (2005), devido à diversidade inerente de componentes de um SoC e seus estilos de especificação, simulação no nível de sistema é tipicamente realizada utilizando uma coleção de modelos de simulação heterogêneos para os diferentes componentes. Consequentemente, modelos de consumo de energia para os diferentes componentes de sistema são também heterogêneos por natureza.

Técnicas de modelagem de consumo de energia no nível de instruções que podem ser utilizadas, por exemplo, por um processador diferem significativamente de modelos de consumo de energia analíticos que podem ser utilizados por um *chip* de memória, e de modelos de consumo de energia no nível de transação usados em um barramento.

Uma das técnicas de modelagem comumente utilizadas consiste em associar constantes de dissipação de potência para cada componente do projeto. A dissipação total de potência é computada pela soma da dissipação de todos os componentes. Este tipo de abordagem é denominada *constant additive model*. Benini, Hodgson e Siegel (1998) apontam como principal problema desta abordagem o fato dela deixar grande fração do processo de estimação para o projetista.

Muitas pesquisas (CALDARI et al., 2002, 2003; GIVARGIS; VAHID, 2002; TALARICO et al., 2005; TIWARI; MALIK; WOLFE, 1994) utilizam instruções como unidade para cobrir o conjunto inteiro de comportamentos de um bloco funcional de um projeto de circuito digital. Nesses é atribuído um valor absoluto ou relativo de energia para cada instrução. Essa

atribuição acontece da seguinte maneira: Quando o conjunto de instruções for identificado, cada uma delas será caracterizada em termos de energia.

2.3. Linguagens de Programação para Projeto de Circuitos Digitais

Nesta seção, serão apresentados conceitos sobre as linguagens SystemC e SystemVerilog, simulação paralela e simulação distribuída, assim como HLA, apresentando os principais conceitos dos elementos que a compõem e noções básicas de gerenciamento de tempo e sincronização.

2.3.1. A Linguagem SystemC

SystemC (IEEE COMPUTER SOCIETY, 2012) é considerada uma das linguagens mais promissoras para projetos no nível de sistema (CONTI et al., 2011). Como uma linguagem para modelagem de alto nível, SystemC fornece uma variedade de construtores *hardware*-orientado e um núcleo de simulação baseado em eventos (LIU et al., 2010). SystemC estende a capacidade de C++. Para viabilizar a modelagem de *hardware*, SystemC adiciona conceitos de concorrência, eventos e tipos de dados apropriados para *hardware*, tais como portas, sinais e módulos.

Por permitir descrições de projetos de *hardware* em múltiplos níveis de abstração (por exemplo, nível de portas, RTL e, principalmente, ESL), SystemC tem se tornado importante na comunidade de projetistas (KLEIN et al., 2007a).

Liu et al. (2010), Conti et al. (2011) e Klein et al. (2007a) citam diversas vantagens da linguagem SystemC, porém apontam a falta de uma semântica para capturar consumo de energia como uma carência desta linguagem. Trabalhos como o de Xanthos, Chatzigeorgiou e Stephanides (2003), Damaševičius (2006), Trabelsi et al. (2011) Greaves e Yasin (2014) têm como objetivo a modificação ou extensão das bibliotecas SystemC, e/ou modificação do núcleo do simulador SystemC para permitir estimação de consumo de energia de sistemas digitais escritos nesta linguagem. Abordagens as quais não modificam o núcleo SystemC, criam algumas interfaces de informações de energia, cujas API podem ser chamadas no

modelo de funcionalidade SystemC para realizar estimação de consumo de energia (KLEIN et al., 2007a, 2007b).

De acordo com Liu et al. (2011), em alguns casos é necessário um núcleo SystemC sem modificações para realização das etapas de verificação funcional e síntese. Modificar o núcleo SystemC ou adicionar chamadas de API irá "manchar" o modelo de funcionalidade SystemC. Modificação no núcleo SystemC de acordo com alguma técnica especial de estimação, não suportará outras. Para chamadas de API, quando o projetista modifica as características do projeto, muito esforço deve ser gasto para lidar com códigos relacionados.

Silveira (2011) aponta outra carência da linguagem SystemC: a falta de uma semântica para modelagem de projetos de circuitos digitais de baixo consumo (*low power*). Neste contexto, o autor adiciona novas funções ao núcleo de um simulador SystemC para ligar e desligar submódulos durante simulações RTL e ESL, permitindo proporcionar ao fluxo de desenvolvimento a verificação funcional do comportamento da técnica de *power gating*[11] no nível de sistema.

A linguagem SystemC tem sido amplamente usada para escrever representações TLM (*Transaction-Level Modeling*) e modelos de referência. TLM é uma abordagem para modelagem de sistemas digitais na qual detalhes de comunicação entre os módulos são separados dos detalhes de implementação de unidades funcionais ou da arquitetura de comunicação. Mecanismos de comunicação como barramentos e FIFO são modelados como canais e são apresentados aos módulos usando interfaces de classes SystemC (GRÖTKER et al., 2002).

No nível de transações é dada maior ênfase nos dados que são transferidos, sua origem e seu destino, em vez de detalhes de implementação do protocolo utilizado para transferência dos dados. TLM em SystemC envolve comunicações entre processos SystemC usando chamada de funções.

[11] A técnica de *power gating* tem como foco a redução de consumo de energia estática em circuitos digitais. Esta técnica é baseada na suspensão da alimentação de submódulos funcionais do SoC que podem ser desligados enquanto outros continuam em funcionamento (EISNER; NAHIR; YORAV, 2009; HENZLER, 2006; HSIEH; YEH; HUANG, 2010; KIM et al., 2009).

Atualmente, a versão mais recente da biblioteca TLM disponibilizada é a 2.0.1. O foco do OSCI TLM-2.0 consiste na modelagem de barramentos mapeados em memória[12]. Porém, isto não significa que o OSCI TLM-2.0 tenha sido desenvolvido exclusivamente para este propósito. Contudo, a maioria de suas características é focada neste tipo de modelagem. TLM 2.0 possui uma estrutura em camadas, com as camadas mais baixas sendo mais flexíveis e gerais, e camadas superiores sendo específicas para modelagem de barramentos. Mais detalhes sobre TLM 2.0 podem ser encontrados em Aynsley (2009).

2.3.2. A Linguagem SystemVerilog

Nos meados de 1990, Verilog tornou-se a linguagem de descrição de *hardware* mais utilizada para simulação e síntese (SPEAR, 2008). Porém, em suas duas versões padronizadas pelo IEEE, esta linguagem oferecia recursos para criar apenas testes simples, não atendendo à parte de verificação para circuitos mais complexos. Assim, surgiram algumas linguagens de verificação de *hardware* (HVL – *Hardware Verification Language*) comerciais, como OpenVera (SYNOPSYS, 2015) e "e" (CADENCE, 2015). Porém, algumas empresas preferiam criar ferramentas personalizadas em vez de utilizar uma destas linguagens (SPEAR, 2008).

Diante dos fatos, um consórcio de companhias de EDA (*Electronic Design Automation*) e usuários uniram-se com intenção de impulsionar o desenvolvimento e o uso de padrões exigidos pela indústria de semicondutores. Tal união deu origem à organização Accellera (ACCELERA, 2015), a qual decidiu criar uma nova geração da linguagem Verilog. A doação da linguagem OpenVera, por parte da Synopsys, formou a base para as características de verificação de *hardware* dessa nova linguagem, a qual foi denominada SystemVerilog (IEEE COMPUTER SOCIETY, 2013).

[12] Um barramento mapeado em memória estende logicamente o espaço de endereçamento de memória do processador para incluir dispositivos neste barramento. Isto permite que dispositivos conectados a este barramento sejam mapeados no espaço de memória da CPU e sejam acessados como memória utilizando ciclos *load* e *store* para endereçar estes dispositivos diretamente (ORACLE, 2015).

Adotada como padrão IEEE em 2005, SystemVerilog foi a primeira linguagem padrão da indústria a cobrir RTL, *assertions, transaction-level modeling* e *coverage-driven constrained random verification*. Em outras palavras, é uma linguagem unificada que abrange modelagem no nível de sistema, RTL e verificação. Spear (2008) afirma que um dos maiores benefícios de SystemVerilog é permitir a criação de ambientes de verificação confiáveis e repetíveis, com sintaxe consistente, que podem ser usados em vários projetos.

Assim como SystemC, a linguagem SystemVerilog incorpora programação orientada a objetos, com algumas diferenças e semelhanças, citadas a seguir.

- SystemC é uma linguagem mais adequada para simulações de *hardware* em alto nível. Diferentemente de SystemVerilog, qualquer projeto de *hardware* baseado em código SystemC será, provavelmente, ineficiente e volumoso. De acordo com (GOERING, 2001), um grupo de projetistas da Motorola desenvolveu um projeto de *hardware* em SystemC, no qual foi verificada uma dificuldade compreensão em relação ao código Verilog do mesmo, além de apresentar menor desempenho, difícil depuração e ineficiência para reúso.

- SystemC estende a capacidade de C++. Para viabilizar a modelagem de *hardware*, SystemC adiciona conceitos de concorrência, eventos e tipos de dados apropriados para *hardware*, tais como portas, sinais e módulos. Desta maneira, para criar modelos (i.e representações de um circuito digital ou parte dele) em SystemC, é necessário que o engenheiro aprenda C++ e tais conceitos (IEEE COMPUTER SOCIETY, 2012). De maneira oposta, SystemVerilog já fornece recursos nativos para *modelagem de hardware* (IEEE COMPUTER SOCIETY, 2013).

- SystemC não possui recursos em sua linguagem para modelar cobertura (SILVEIRA, 2011). De modo contrário, SystemVerilog possui características nativas para medição da cobertura funcional (IEEE COMPUTER SOCIETY, 2013).

- Tanto em SystemC quanto em SystemVerilog é possível criar modelos no nível de transação com eficiência comparável (GOERING, 2001).

- Os simuladores SystemVerilog necessitam de licenças pagas (e.g., simuladores da Cadence, Mentor ou Synopsys) enquanto os simuladores SystemC não necessitam, caso todos os modelos estejam disponíveis em SystemC.

- SytemVerilog incorpora programação orientada a objetos, porém SystemC tem um suporte mais rico para este tipo de abordagem, já que suporta herança múltipla e sobrecarga de funções (IEEE COMPUTER SOCIETY, 2013).

- SystemVerilog destaca-se na fusão de diferentes tecnologias de verificação funcional: *Assertions*, randomicidade direcionada e cobertura funcional (IEEE COMPUTER SOCIETY, 2013).

Uma característica importante da linguagem SystemVerilog é que seus ambientes permitem chamadas de funções em C/C++ por meio de DPI (*Direct Programming Interface*), possibilitando que modelos de referência escritos nessas linguagens consigam interoperar com esse ambiente. Um exemplo de uso de DPI na indústria EDA (*Electronic Design Automation*) é a biblioteca UVM *Connect (Universal Verification Methodology Connect)*(MENTOR, 2015a) da empresa Mentor (MENTOR, 2015b). Por meio de DPI, essa biblioteca provê conectividade TLM1 e TLM2 e passagem de objetos entre modelos e componentes SystemC e SystemVerilog.

2.4. Simulação Paralela e Simulação Distribuída

Plataformas de computação paralela e distribuídas são diferenciadas pela área física ocupada pelo computador (FUJIMOTO, 2000). De acordo com o mesmo autor, pode-se dizer que uma simulação paralela ocorre em um conjunto de computadores confinados em um pequeno espaço geográfico, tais como um único gabinete ou um conjunto pequeno de gabinetes em uma sala, fazendo com que seus processadores estejam fisicamente

próximos. Geralmente, são utilizadas máquinas homogêneas, dispondo de processadores do mesmo fabricante.

Essas máquinas normalmente fornecem *hardware* de comutação adequado para computação paralela, permitindo a transmissão de mensagens de um computador para o outro com latência relativamente pequena.

De maneira antagônica, simulação distribuída acontece em máquinas que estão geograficamente distribuídas, podendo estar em um único prédio ou até mesmo em continentes diferentes. Diferentemente da simulação paralela, na simulação distribuída cada um dos nós de processamento geralmente é uma máquina completa que incluí sua própria memória e seus dispositivos de entrada e saída. Outra característica marcante é que devido ao uso de redes LAN ou WAN para troca de mensagens entre esses processadores, a latência torna-se bem maior em comparação com uma simulação paralela.

Simulações executadas em computadores com vários processadores utilizando memória compartilhada, multicomputadores com memória distribuída ou máquinas SIMD (*Single Instruction, Multiple Data*) são definidas como programas de simulação paralela. Simulações executadas em computadores distribuídos são definidas como simulações distribuídas.

A arquitetura de alto nível (HLA), adotada pela abordagem desenvolvida como meio para viabilizar a troca de dados entre os modelos e envio de informações de consumo de energia durante a simulação destes, é definida como uma arquitetura para modelagem e realização de simulação distribuída.

2.5. *High Level Architecture* (HLA)

A *High Level Architecture* (HLA) foi originalmente desenvolvida pelo Escritório de Modelagem e Simulação (DMSO - *Defense Modeling and Simulation Office*) para o Departamento de Defesa dos Estados Unidos (DoD - *US Department of Defence*). Seu campo original de aplicação são simulações de treinamento militar com milhares de participantes militares interagindo dentro de um exercício de treinamento compartilhado (ROTH et al., 2011a). A primeira versão completa do padrão, conhecida como HLA 1.3, foi publicada em 1998 (DOD, 1998).

No ano de 2000, HLA tornou-se um padrão IEEE, desenvolvido para fornecer uma arquitetura comum para modelagem e simulação distribuída. A HLA define uma abordagem integrada que fornece um *framework* comum para interconexão de simulações interativas (IEEE COMPUTER SOCIETY, 2010b). Kuhl, Weatherly e Dahmann (1999) definem HLA como sendo uma "cola" que permite a combinação de simulações de computador em uma simulação maior.

Utilizando HLA, simulações podem interagir com outras, independente das plataformas em que estão sendo executadas. A arquitetura de alto nível (HLA) facilita o reúso e a interoperação de sistemas de simulação e sistemas que podem ser mantidos ou controlados para produzir algum valor, como por exemplo, gerenciadores da simulação, coletores de dados, sistemas de apresentação de dados, sistemas do mundo real, etc (IEEE COMPUTER SOCIETY, 2010a).

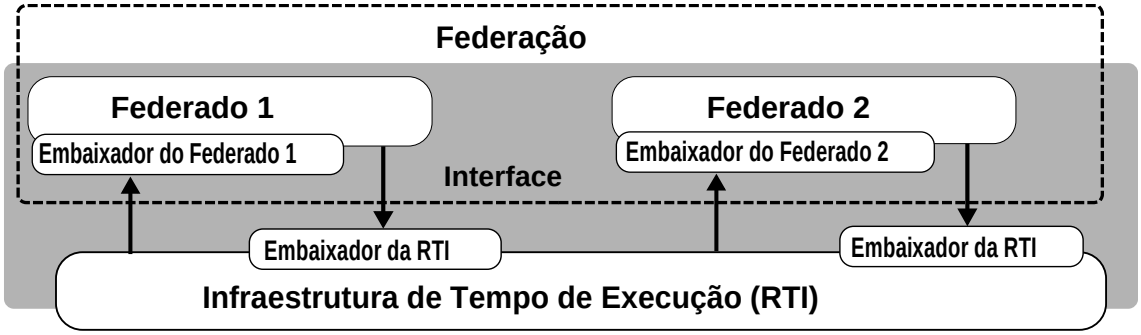
Existem diversas implementações comerciais e de código aberto que viabilizam o uso da arquitetura HLA. Essas implementações obedecem as regras e especificações contidas no padrão a fim de prover uma interface de serviços comum para sincronização e troca de dados durante a execução das simulações. Para a elaboração da abordagem desenvolvida, foi utilizada uma implementação de código aberto denominada CERTI (NOULARD; ROUSSELOT; SIRON, 2009).

2.5.1. Estrutura Geral da HLA

A arquitetura de alto nível provê meios para que simulações e/ou aplicações possam interagir. Cada uma dessas simulações/aplicações que interagem representa uma entidade intitulada de federado (*federate*). Esta combinação de simulações/aplicações é denominada de federação (*federation*). Federados podem coordenar operações e trocar dados durante a execução da federação por meio de uma estrutura denominada Infraestrutura de Tempo de Execução (*Run-Time Infrastructure* ou RTI). A Infraestrutura de Tempo de Execução é o *software* que fornece uma interface de serviços comum para sincronização e troca de dados durante a execução de uma federação HLA (IEEE COMPUTER SOCIETY, 2010b).

A estrutura genérica dos principais elementos que compõem a HLA é apresentada na Figura 1.

Figura 1 - Estrutura genérica de elementos da arquitetura HLA.



Fonte: O autor, 2015.

Toda informação trocada entre os federados deverá ocorrer via RTI. A RTI oferece uma interface "Embaixador da RTI" (*RTI ambassador)* para cada um dos federados que participam da federação. O Embaixador da RTI é a interface a partir da qual o federado invoca operações para requisitar serviços da RTI. Cada um dos federados deverá possuir uma interface de comunicação para receber chamadas da RTI. Essa interface é denominada Embaixador do Federado (*Federate Ambassador)*. Mais detalhes serão apresentados na Seção 2.5.2.7.

A arquitetura de alto nível é composta por três documentos (IEEE COMPUTER SOCIETY, 2010a):

- *Framework* e regras HLA (*HLA rules*): Uma definição da HLA, seus componentes e um conjunto de dez regras básicas que juntas descrevem os princípios gerais desta arquitetura. Este componente é definido pelo padrão IEEE 1516 (IEEE COMPUTER SOCIETY, 2010a);
- Interface de especificação HLA (*HLA interface specification*): Uma descrição da interface funcional entre simulações (federados) e RTI. Este componente é definido pelo padrão IEEE 1516.1 (IEEE COMPUTER SOCIETY, 2010b). A RTI provê serviços para federados de maneira análoga à forma como um sistema operacional distribuído provê serviços para suas aplicações. As implementações comerciais e de código aberto que viabilizam o uso da arquitetura HLA são baseadas no documento Interface de especificação HLA;
- *HLA Object Model Template* (OMT): Uma especificação para definir o formato e sintaxe de modelos de objetos HLA. O conceito de objetos HLA assemelha-se ao conceito de objetos em programação orientada a objetos. O HLA OMT é definido pelo padrão IEEE 1516.2 (IEEE COMPUTER SOCIETY, 2010c).

2.5.2. Conceitos Gerais da HLA

Para o entendimento da pesquisa ora descrita, alguns conceitos sobre HLA devem ser discutidos. A seguir, serão definidos, de maneira mais detalhada, os principais elementos da arquitetura de alto nível. As definições apresentadas a seguir são baseadas principalmente nas definições dos padrões IEEE (IEEE COMPUTER SOCIETY, 2010a, 2010b, 2010c) que definem HLA.

2.5.2.1. Federado (*Federate*)

Federado é uma simulação compatível com a arquitetura HLA (KUHL; WEATHERLY; DAHMANN, 1999). De acordo com o padrão IEEE (IEEE COMPUTER SOCIETY, 2010a), um federado pode ser entendido como uma aplicação que pode ser ou está conectada a outra aplicação, sob um *Federation Object Model* (FOM) *Document Data/Federation Execution Data* (FDD/FED) e uma RTI. Neste grupo podem ser incluídos gerenciadores de federações, coletores de dados, sistemas do mundo real, simulações, sistemas de apresentação de dados, e outros utilitários.

2.5.2.2. Federação (*Federation*)

Múltiplas entidades de simulação são conectadas via RTI usando um OMT (*Object Model Template*) comum (KUHL; WEATHERLY; DAHMANN, 1999). O padrão IEEE (IEEE COMPUTER SOCIETY, 2003) define federação como sendo um conjunto de federados interagindo via serviços da RTI utilizando um Modelo de Objeto da Federação (FOM - *Federation Object Model*). Todo federado que participa da execução da federação é denominado federado afiliado (*joined federate*).

2.5.2.3. *Object Model Template* (OMT)

O OMT define o formato e sintaxe (mas não o conteúdo) para gravação de informações nos modelos de objetos HLA, para incluir objetos, atributos, interações e parâmetros. O *Federation Object Model* (FOM) e *Simulation Object Model* (SOM) são documentados de acordo com o HLA OMT. O padrão IEEE 1516.2 (IEEE COMPUTER SOCIETY, 2010c) define o HLA OMT. O padrão HLA OMT não define os conteúdos de um *Simulation Object Model*

(SOM) (Seção 2.5.2.5) ou FOM (Seção 2.5.2.4), porém define um formato e sintaxe apropriados para suas documentações.

2.5.2.4. *Federation Object Model* (FOM)

O FOM descreve o conjunto de classes de objetos, atributos, classes de interação e parâmetros que são compartilhados durante a execução da federação (DUMOND; LITTLE, 2003). De acordo com o padrão IEEE 1516.1 (IEEE COMPUTER SOCIETY, 2010b), o FOM é uma especificação que define a informação trocada em tempo de execução para alcançar um conjunto de objetivos da federação. O FOM inclui:

- O conjunto classes de objetos escolhidos para representar a informação trocada em uma federação;
- O conjunto de classes de interação escolhidas para representar a ação combinada entre objetos criados;
- Atributos de classes de objetos e parâmetros de classes de interação, e outras informações relevantes.

Os componentes de um FOM estabelecem um "modelo de contrato de informação" necessário para garantir a interoperabilidade entre federados. O FOM pode ser também definido como o conjunto de SOM dos federados que participam da federação (KIM; CHOI; KIM, 2013). Na HLA todo objeto é uma instância de uma classe de objeto encontrada no FOM.

2.5.2.5. *Simulation Object Model* (SOM)

O SOM (*Simulation Object Model*) é uma especificação dos tipos de informação[13] que um federado pode prover para as federações, como também a informação que um federado pode receber de outros dentro da federação (IEEE COMPUTER SOCIETY, 2010a). O SOM é distinto de informações internas pertinentes ao federado (DUMOND; LITTLE, 2003). A descrição das classes do federado, dos objetos e atributos são dadas no SOM do federado. A intenção do SOM é descrever uma interface pública do federado em termos de um conjunto identificado de objetos e interações suportados. O conjunto de SOM dos federados que compõem a federação definem o FOM (*Federation Object Model*) (KIM; CHOI; KIM, 2013).

[13] Os tipos de informação referem-se às classes de objetos, atributos, classes de interação e parâmetros.

2.5.2.6. *FOM Document Data* (FDD)

O FDD contém informações derivadas do FOM (classes, atributos, parâmetros, nomes, etc.) utilizadas pelo RTI durante o tempo de execução. Cada execução de federação necessita de um arquivo FDD. Na versão HLA 1.3 em vez de se ter um arquivo FDD, tem-se um arquivo FED (*Federation Execution Data*). Uma das diferenças entre o padrão IEEE 1516 e o padrão HLA 1.3 é que o primeiro utiliza um formato XML (*EXtensible Markup Language*) para definir seu FOM e a versão 1.3 utiliza a linguagem LISP (GRAHAM, 1995).

2.5.2.7. Infraestrutura de Tempo de Execução (RTI)

A Infraestrutura de Tempo de Execução (*Run-Time Infrastructure* ou RTI) pode ser definida como um *software* que provê uma interface de serviços comuns para sincronização e troca de dados durante a execução de uma federação. É a implementação dos serviços iniciados pelo federado (*Federate Initiated Services*). Cada federado deve acessar os serviços disponíveis na RTI via interface "Embaixador da RTI" (*RTI Ambassador)*.

A interface "Embaixador da RTI" é parte da implementação da RTI. De maneira análoga, quando a RTI desejar se comunicar com o federado, seja em resposta a um serviço solicitado pelo federado (*callbacks*) ou simplesmente para informá-lo de algum evento pelo qual ele esteja interessado, ela deve fazê-lo via interface "Embaixador do Federado" (*Federate Ambassador)*. A interface "Embaixador do Federado" deve ser implementada pelo usuário da HLA.

São exemplos de serviços iniciados pelo federado: *publish, subscribe, register, update*, etc. São exemplos de serviços iniciados pela RTI (RTI *Initiated Services*): *discover*[†] [14]*, reflect, time advance grant*[†], etc. Os principais serviços utilizados na pesquisa ora descrita serão apresentados ao longo deste documento. O conjunto de serviços RTI oferecidos pelo IEEE 1516 é definido no padrão IEEE 1516.1 - *Federate Interface Specification* (IEEE COMPUTER SOCIETY, 2010b). No Quadro 1 é apresentado um resumo da finalidade de cada grupo de serviços oferecidos pela RTI.

[14] A adaga indica que este é um método de retorno (*callback*).

Quadro 1 - Grupo de serviços oferecidos pela RTI.

Grupo de serviço	Definição
Federation management	Grupo de serviços para criação, controle dinâmico, modificação e finalização de uma execução de federação.
Declaration management (DM)	Federados afiliados utilizam esses serviços para declarar sua intenção em gerar informação.
Object management	Grupo de serviços para lidar com o registro, modificação, e finalização de instâncias de objetos e envio e recebimento de interações[15].
Ownership management	Grupo de serviços usados pelos federados afiliados e RTI para a transferência de posse de atributos de instância de objetos entre esses federados.
Time management	Esses serviços e outros mecanismos associados fornecem meios para ordenar a entrega de mensagens durante a execução da federação. Os serviços de gerenciamento de tempo também são utilizados para controlar o avanço de federados ao longo do eixo de tempo da federação durante sua execução.
Data distribution management (DDM)	Federados afiliados utilizam esses serviços para reduzir a transmissão e recepção de dados irrelevantes. Enquanto que serviços DM fornecem informação na relevância de dados no nível de atributo de classe, serviços DDM adicionam capacidade para refinar ainda mais os requisitos de dados no nível de instância.
Support services	Definem vários serviços para recuperar informação sobre a federação, tais como classes e interações.

Fonte: IEEE COMPUTER SOCIETY (2010b).

2.5.3. Regras HLA

O padrão IEEE 1516 (IEEE COMPUTER SOCIETY, 2010a) fornece um conjunto de dez regras que definem os princípios da HLA em termos de responsabilidades dos federados e das federações.

2.5.3.1. Regras para Federações

A seguir, as regras para as federações.

1) Federações devem possuir um HLA FOM documentado de acordo com o HLA OMT (IEEE COMPUTER SOCIETY, 2010c).
2) Em uma federação, todas as instâncias de objetos associados a uma simulação devem estar nos federados e não na RTI.
3) Durante a execução da federação, todas as trocas de dados FOM entre os federados devem ocorrer via RTI.
4) Durante a execução de uma federação, os federados devem interagir com o RTI de acordo com a especificação de interface HLA (IEEE COMPUTER SOCIETY, 2010b).

[15] Uma interação é uma ação explícita realizada por um federado que pode ter algum efeito ou impacto em outro federado dentro da execução da federação.

5) Durante a execução de uma Federação, um atributo de uma instância de objeto deve ser de propriedade de, no máximo, um federado em qualquer dado instante.

2.5.3.2. Regras para Federados

A seguir, as regras para os federados.

1) Federados devem possuir um HLA SOM documentado de acordo com o HLA OMT (IEEE COMPUTER SOCIETY, 2010c).
2) Federados devem estar aptos a atualizar e/ou refletir quaisquer atributos de seus objetos e enviar e/ou receber interações, conforme especificado em seus SOM.
3) Todos os Federados devem estar aptos a transferir ou aceitar a posse de atributos dinamicamente durante uma execução de federação, conforme especificado em seus SOM.
4) Federados devem estar aptos a variar as condições (e.g., limites) sob as quais eles fornecem atualizações de atributos, conforme especificado em seus SOM.
5) Federados devem estar aptos a gerenciar o tempo local de maneira a permitir a troca de dados coordenada com outros membros da federação.

2.5.4. Simulação Contínua *versus* Simulação de Evento Discreto

Um dos conceitos-chave sobre simulação é a forma como o tempo avança. Com respeito ao avanço de tempo, existe uma distinção básica entre simulações baseadas em modelos contínuos e simulações baseadas em modelos de eventos discretos.

Modelos contínuos assumem o contínuo avanço de tempo e mudança de estado. Em termos computacionais, isto é representado por uma variável de tempo, a qual avança em incrementos pequenos o bastante para simular precisamente o comportamento contínuo do modelo.

Atualizações nas variáveis que representam o estado do sistema são tipicamente computadas para cada passo no tempo (CSU, 2001). Em contraste, simulações de eventos discretos são baseadas em modelos que assumem um sistema de estado imutável até que ocorra um evento que

produza uma mudança instantânea no estado do sistema. Neste tipo de simulação o tempo avança de evento para evento, na ordem cronológica de tempo.

No Quadro 2, são apresentadas as principais diferenças entre esses dois tipos de simulação. Na Figura 2, é apresentada a maneira como ocorre os passos de tempo (*timesteps*) em cada um desses tipos de simulação. Um federado que implementa simulação de evento discreto não possui um *timestep* conhecido, ou previsível.

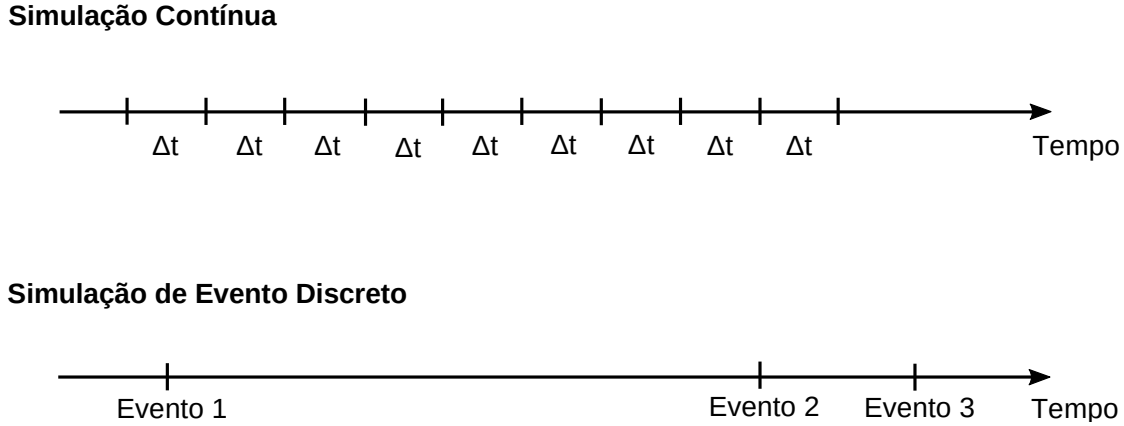
Na simulação contínua, o tempo de simulação é dividido em uma sequência de *timesteps* de igual tamanho (Δt, Figura 2). O tempo avança de um *timestep* para o próximo. Na simulação de eventos discretos, o tempo da simulação não avança de um *timestep* para o próximo, mas avança do tempo que ocorre o evento para o tempo do próximo evento (ROTH et al., 2011a).

Quadro 2 – Quadro comparativo entre simulação contínua e simulação de evento discreto.

Simulação Contínua	Simulação de Evento Discreto
Avança o tempo e o estado do sistema continuamente.	O estado do sistema muda apenas quando ocorrem eventos.
O tempo avança em incrementos pequenos o bastante para garantir precisão.	O tempo avança de evento em evento.
O estado das variáveis é atualizado a cada passo de tempo (*timestep*).	O estado das variáveis é atualizado à medida que ocorre cada evento.

Fonte: California State University, Chico (CSU, 2001).

Figura 2 - Passos de tempo (*timesteps*) das simulações.



Fonte: O autor, 2015.

2.5.5. Ordenação de Mensagens

Durante a execução da federação, mensagens devem ser trocadas entre os federados via RTI. As mensagens trocadas por esses federados podem ou

não ter um rótulo de tempo (*timestamp*) associado a cada uma delas. Este tempo pode ser utilizado para ordenar a maneira como essas mensagens são recebidas. Existem dois tipos de ordem de recebimento de mensagens: *Receive Order* (RO) e *Time Stamped Order* (TSO).

2.5.5.1. Receive Order (RO)

Uma mensagem para a qual não há garantia de ordem de entrega é dita RO (*Receive Order*). Mensagens que são recebidas como RO serão recebidas em uma ordem arbitrária pelo respectivo federado afiliado. Caso um valor de *timestamp* seja fornecido com este tipo de mensagem, esse *timestamp* não terá influência na ordem de recebimento da mensagem.

2.5.5.2. *Time Stamped Order* (TSO)

Mensagens que possuem um *timestamp* (rótulo de tempo) associado e devem ser entregues a um federado receptor na ordem correta, com respeito ao seu *timestamp,* são chamadas de TSO (*Time Stamped Order*).

A RTI ordena mensagens de federados afiliados que fazem uso dos serviços de gerenciamento de tempo (*time management services*) e mensagens contendo rótulos de tempo (*timestamps*). Mensagens contendo *timestamps* diferentes são ditas entregues em TSO se para quaisquer duas mensagens, M1 e M2 (rotuladas com tempo T1 e T2, respectivamente), que são entregues a um único federado, em que T1<T2, M1 é entregue antes de M2. Mensagens que possuem o mesmo *timestamp* serão entregues em uma ordem arbitrária.

Federados que fazem uso dos serviços de gerenciamento de tempo (*time management services*) e fazem uso de mensagens contendo *timestamps* nunca receberão mensagens do passado, ou seja, mensagens com o *timestamp* menor do que o tempo lógico (*logical time*) atual do federado. De acordo com Fujimoto (1998), tempo lógico é o tempo no federado durante o qual este é capaz de enviar ou receber mensagens ordenadas por rótulos de tempo.

2.5.6. Tipos de Federados

Um federado pode ser "regulador de tempo", "de tempo restringido", "regulador e restringido", como também "nem regulador nem restringido". Quando um federado é criado, por padrão, ele deverá ser definido como "nem regulador nem restringido".

De acordo com o padrão IEEE 1651.1(IEEE COMPUTER SOCIETY, 2010c), um federado de tempo restringido (*time-constrained federate*) é um federado afiliado que pode receber mensagens TSO e para os quais o avanço de tempo é restringido por outros federados afiliados dentro da execução da federação. O avanço de tempo é restringido por um limite colocado em cada um desses federados, denominado GALT (*Greatest Available Logical Time*) (Figura 3). O GALT será explicado em detalhes nas próximas seções.

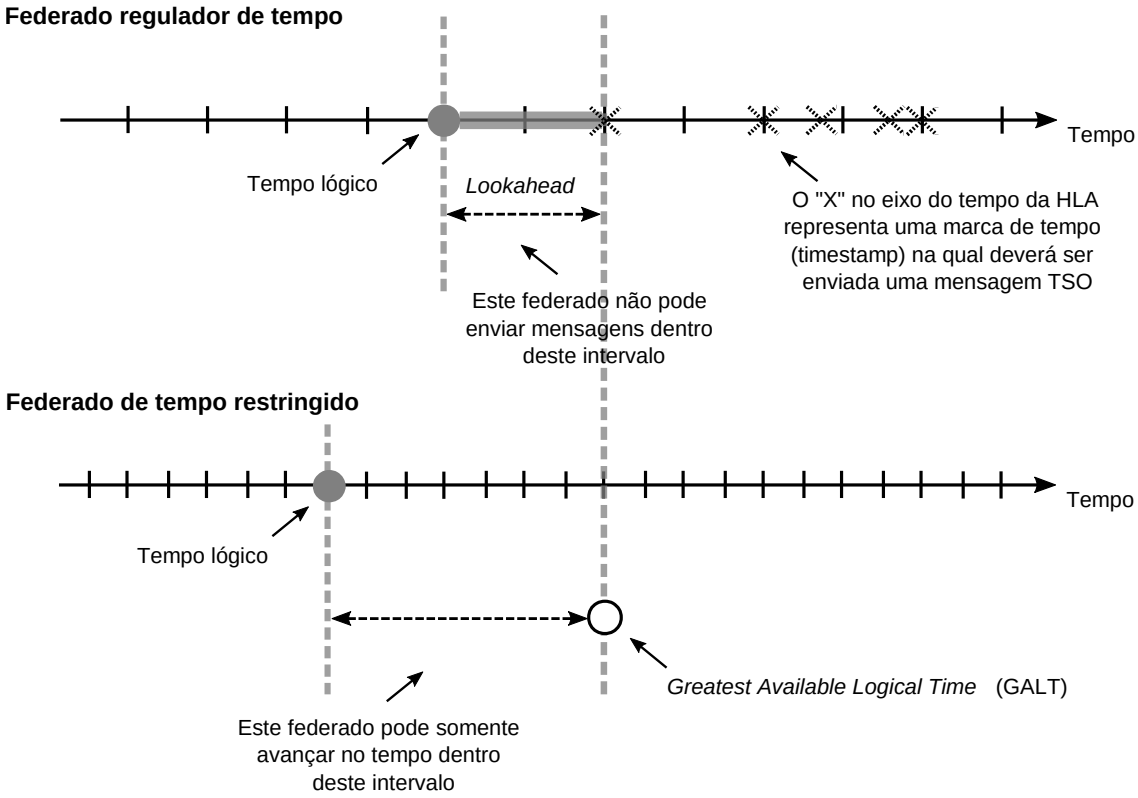
O mesmo padrão IEEE define que um federado regulador de tempo (*time-regulating federate*) é um federado afiliado que pode enviar mensagens TSO e que restringe o avanço de tempo de outros federados afiliados dentro da execução da federação. Todo federado regulador de tempo deve prover um valor de *lookahead* quando se torna regulador de tempo. Uma vez estabelecido, este valor só pode ser modificado a partir da chamada do serviço *Modify Lookahead*.

O *lookahead* é um valor não negativo que estabelece o menor valor dos *timestamps* para os quais podem ser enviadas mensagens TSO pelo federado regulador de tempo. Um federado regulador de tempo não pode enviar mensagens TSO que contenham *timestamp* menor que o valor do seu tempo lógico[16](*logical time*) somado ao valor do seu *lookahead* (Figura 3).

Existe uma restrição adicional para federados reguladores de tempo que possuem um *lookahead* de valor zero. Neste tipo de situação, o federado regulador de tempo não deve enviar mensagens TSO que contenham um timestamp menor ou igual que o seu tempo lógico (somado ao valor do *lookahead*, o qual é zero).

[16] O ponto atual no eixo do tempo da HLA (IEEE COMPUTER SOCIETY, 2010b).

Figura 3 - Estrutura genérica de um federado regulador de tempo e um federado de tempo restringido.



Fonte: O autor, 2015.

2.5.7. Noções Básicas de Gerenciamento de Tempo

Diferentes políticas de gerenciamento de tempo podem ser adotadas com o uso da HLA. A RTI disponibiliza serviços opcionais de gerenciamento de tempo para coordenar a troca de eventos entre os federados. Esses eventos podem ser associados com um ponto no tempo e o RTI pode assegurar comportamento causal. Também é possível que um ou mais federados em uma federação ignorem o tempo.

Por padrão, o RTI não tenta coordenar o tempo entre os federados. Uma das vantagens proporcionadas pelo uso da HLA é que ela além de suportar uma variedade de políticas de gerenciamento de tempo, também permite a interoperabilidade entre federados com diferentes políticas (IEEE COMPUTER SOCIETY, 2010b).

Em uma federação o tempo sempre se move para frente. Porém, a percepção de tempo atual pode ser diferente entre os participantes da federação. O gerenciamento de tempo tem como objetivo controlar o avanço de tempo de cada um dos federados ao longo do eixo do tempo da

federação (PATERSON; HOUGLAND; SANMIGUEL, 2000). Existem situações em que é apropriado restringir o progresso de um federado baseado no progresso de outro.

Como já mencionado, federados que restringem o progresso de avanço do tempo de outro federado são denominados federados reguladores de tempo (*time-regulating federates*). De maneira oposta, federados que possuem seu progresso de tempo determinado por federados reguladores são denominados federados de tempo restringido (*time-constrained federates*).

Um federado pode ser "regulador", "restringido", "regulador e restringido", como também "nem regulador nem restringido". Quando um federado é criado, por padrão, ele deverá ser definido como "nem regulador nem restringido".

Todo federado possui um tempo lógico (*logical time*). O tempo lógico de um federado pode ser definido como o ponto atual no eixo do tempo da HLA (IEEE COMPUTER SOCIETY, 2010c). Um federado afiliado que se torna regulador de tempo[17] pode associar alguma de suas atividades (e.g., envio de interações e atualização de valores de atributos de instância de objetos) com pontos no eixo do tempo da HLA.

Rótulos de tempo (*timestamps*) devem ser atribuídos para mensagens que representam essas atividades. Esses *timestamps* correspondem aos pontos do eixo do tempo da HLA com os quais as atividades estão associadas.

Um federado afiliado de tempo restringido está interessado em receber essas mensagens representando essas atividades (e.g., recebimento de interações e reflexão de valores de atributos de instância de objetos) em uma federação nas quais federados enviam mensagens TSO.

O tempo lógico de federados afiliados reguladores de tempo deve ser usado para restringir o avanço de tempo lógico dos federados que são restritos ao tempo (IEEE COMPUTER SOCIETY, 2010b).

Os serviços de gerenciamento de tempo (*time management services*) e mecanismos associados fornecem meios para ordenar a entrega de mensagens durante a execução da federação. O uso desses mecanismos

[17] Ao afiliar-se à federação, um federado pode fazer a requisição a RTI para tornar-se *time-regulating* ou *time-constrained* pela chamada dos serviços gerenciamento de tempo *Enable Time Regulation* ou *Enable Time Constrained*, respectivamente.

permite que mensagens enviadas por diferentes federados afiliados sejam entregues em ordem consistente para qualquer federado afiliado que deseje receber essas mensagens durante a execução da federação.

O uso dos serviços de gerenciamento de tempo permite a coordenação de troca de mensagens entre federados afiliados reguladores e federados afiliados de tempo restringido. Federados que não são reguladores de tempo, nem de tempo restringido, não precisam fazer uso de nenhum dos serviços de gerenciamento de tempo (IEEE COMPUTER SOCIETY, 2010b).

2.5.8. Sincronização

O RTI somente concede o avanço de tempo por meio do serviço *Time Advance Grant*† (*callback*) quando ele garante que nenhuma mensagem TSO será posteriormente entregue com o *timestamp* menor do que o avanço de tempo concedido. Assim, a RTI garante que os federados nunca receberão mensagens com *timestamp* menor do que seu tempo lógico atual (GUPTA, 2007).

Um limite é colocado em cada um dos federados afiliados (*joined federates*) de tempo restringido que limita o quanto eles podem avançar no tempo, garantindo que esses não avançarão o tempo e ultrapassarão um ponto no qual mensagens TSO poderiam ainda ser enviadas por outros federados afiliados[18]. Caso um desses federados requisite um avanço de seu tempo lógico além desse limite, o avanço de tempo não será concedido até que este limite tenha aumentado além do tempo lógico a ser concedido (IEEE COMPUTER SOCIETY, 2010b). Este limite no avanço é expresso em termos de um valor denominado GALT (*Greatest Available Logical Time*) (Figura 3, Seção 2.5.6).

Cada federado afiliado tem um GALT que expressa o maior tempo lógico para qual a RTI garante que pode conceder um avanço de tempo sem ter que esperar o avanço de outros federados da simulação. O GALT é calculado pela RTI baseado em fatores tais como o tempo lógico, *lookahead* e solicitações de avanço no tempo lógico de federados afiliados reguladores de tempo.

[18] Razão pelo qual se usa o termo "restringido" (*constrained*).

Caso não existam federados reguladores de tempo afiliados, o GALT do federado afiliado é indefinido, significando que esse federado poderá avançar para qualquer ponto sem ter que esperar pelo avanço de tempo de outros federados afiliados. O GALT é usado pelo RTI para limitar o avanço de tempo de federados afiliados de tempo restringido. Porém, federados afiliados que não são de tempo restringido também possuem um GALT. Para esses, o GALT expressa o limite que será aplicado ao federado caso esse se torne de tempo restringido (*time-constrained*).

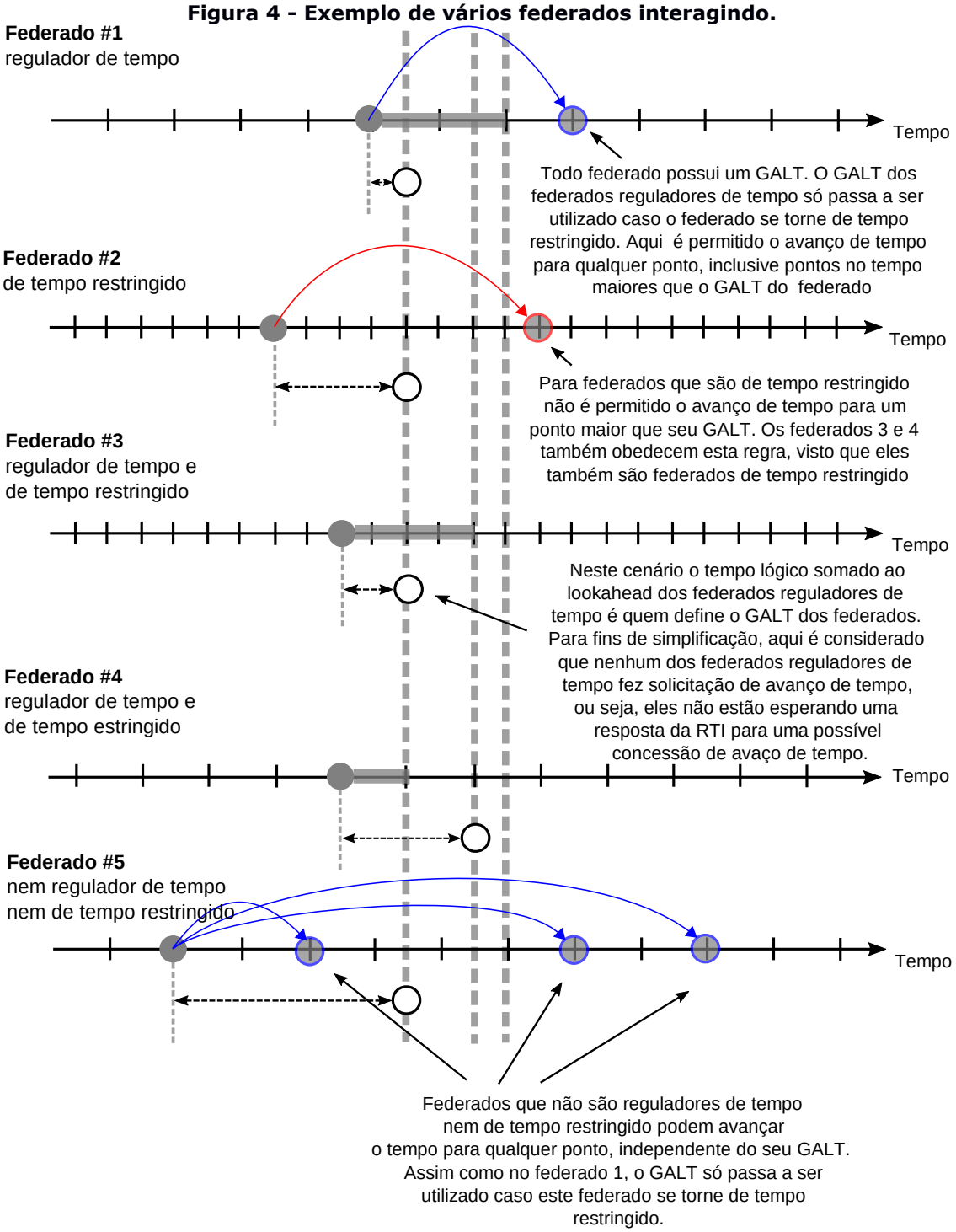
Para a contextualização dos conceitos, na Figura 4 é apresentado um cenário no qual vários federados interagem. Nela é mostrado que o eixo do tempo de alguns dos federados possui *timesteps* espaçados diferentemente. Por exemplo, no Federado 1, cada *timestep* é equivalente a dois *timesteps* do Federado 2. Esse espaçamento diferenciado significa que os federados podem avançar seu tempo local com graduações distintas. Porém, a HLA permite que federados com diferentes graduações de tempo local possam coordenar a troca de dados com outros membros da federação (ÇAYIRCI; MARINCIC, 2009).

As linhas e círculos azuis da Figura 4 significam um avanço de tempo lógico permitido, ou seja, ele será concedido pela RTI, caso seja solicitado pelo federado. De maneira oposta, as linhas e círculos vermelhos significam um avanço de tempo que não é permitido, ou seja, um avanço de tempo lógico que não será concedido pela RTI, caso seja solicitado pelo federado.

Baseado na ideia do GALT cada federado afiliado possui um valor denominado LITS (*Least Incoming Timestamp*). Este valor expressa o menor *timestamp* em que um federado afiliado poderia (porém, não necessariamente) receber no futuro uma mensagem TSO, ou seja, o *timestamp* da próxima mensagem que o federado pode ter que processar.

O LITS do federado afiliado é calculado pela RTI e é baseado no seu GALT e também em quaisquer mensagens enfileiradas (mensagens que estejam no *buffer* da RTI) que possam ser posteriormente recebidas pelo federado afiliado. Caso o GALT do federado afiliado seja indefinido e também não existam mensagens TSO enfileiradas, que o federado afiliado possa receber, o LITS desse federado também será indefinido.

Figura 4 - Exemplo de vários federados interagindo.



Fonte: O autor, 2015.

O padrão IEEE 1516 introduz algumas modificações em sua terminologia em relação ao padrão HLA 1.3. No padrão IEEE 1516, como já mencionado, o termo GALT é denominado LBTS (*Lower Bound Time Stamp*) no padrão HLA 1.3. O LITS do padrão IEEE 1516 é denominado MNET (*Minimum Next Event Time*) no padrão HLA 1.3.

2.6. Considerações Finais do Capítulo

Neste capítulo, foi apresentada a fundamentação teórica para a contextualização da pesquisa. No capítulo foram definidos os principais conceitos relacionados à estimação de consumo de energia, apresentando todos os seus níveis de abstração, com maior foco no nível de sistema. Também foram apresentados conceitos sobre as linguagens SystemC e SystemVerilog, simulação paralela e simulação distribuída, assim como HLA, apresentando os principais conceitos dos elementos que a compõem e noções básicas de gerenciamento de tempo e sincronização.

No Capítulo 3 serão apresentados as pesquisas relacionadas analisadas para o desenvolvimento da abordagem, destacando suas principais contribuições e características.

Capítulo 3

Pesquisas Relacionadas

Neste capítulo, serão apresentados as pesquisas relacionadas analisadas para o desenvolvimento da abordagem. Serão destacadas suas principais contribuições e características. O capítulo tratará das abordagens para estimação de consumo de energia estudadas. Uma abordagem será eleita para validação da pesquisa ora apresentada, considerando os estudos de casos relativos aos modelos MPSoC C++/SystemC, criado a partir da plataforma MPSoCBench (DUENHA et al., 2014).

3.1. Abordagens para estimação de Consumo de Energia

Uma das primeiras pesquisas para estimação de consumo de energia em um nível mais alto que o nível de circuitos ou nível de portas foi realizada por Tiwari, Malik e Wolfe (1994). É a primeira pesquisa que foca na modelagem de consumo de energia de processadores (IKHWAN et al., 2006). Na época, o uso de sistemas embarcados estava crescendo muito, assim como as aplicações que eram executadas nesses sistemas. Muitas aplicações de sistemas embarcados são críticas em termos de consumo de energia, ou seja, em termos de restrições de consumo de energia, formam uma parte importante da especificação do projeto.

Na década de 90, poucas ferramentas estavam disponíveis para medição de consumo de energia de projetos de sistemas embarcados nos níveis mais altos que o nível de portas lógicas. Tais ferramentas eram muito lentas e impraticáveis para avaliação de consumo de energia de sistemas embarcados e, frequentemente, não podiam ser aplicadas devido à falta de disponibilidade de informação no nível de circuito e de portas lógicas dos processadores embarcados. Neste sentido, os autores apresentam uma metodologia para desenvolver e validar um modelo de consumo de energia para qualquer processador. O modelo pode ser fornecido pelo fabricante do

processador, tanto para processadores de prateleira quanto para processadores embarcados.

A pesquisa tem como objetivo viabilizar a avaliação do custo energético do software embarcado, da mesma maneira que, na época, era avaliado o custo energético no nível de portas. A técnica foi aplicada em dois microprocessadores comerciais da época, Intel 486DX2 e o Fujitsu SPARClite 934. Cada um dos processadores foi caracterizado em termos de instruções. Um valor de consumo energia foi atribuído a cada uma das instruções dos processadores.

Caldari et al. (2003) apresentam um conjunto de orientações para a integração de informação de consumo de energia em projetos ESL parametrizados. Como estudo de caso foi utilizada uma descrição em SystemC de um barramento que utiliza o protocolo AMBA (ARM LTD, 2015). As orientações descritas foram aplicadas sobre este IP (*Intellectual Property*). Um conjunto de instruções foi identificado para cobrir toda funcionalidade do barramento. Cada uma das instruções foi caracterizada para extrair um modelo de consumo de energia com base no uso específico de sub-blocos previamente determinados. A partir de simulações no nível de sistema pôde-se estimar a energia consumida por este barramento.

Damaševičius (2006) apresenta um *framework* para estimação de características do projeto no RTL. Nele, a biblioteca SystemC foi estendida com novas classes descrevendo a computação de área, atraso e características de potência dos modelos SystemC no nível de abstração RTL. De acordo com o autor, o *framework* permite estimar as características do projeto em um nível de abstração mais alto, viabilizando modelagem e teste mais rápidos.

O *framework* também permite estimar as características do projeto nos estágios iniciais do fluxo de desenvolvimento, diminuindo o custo e a dificuldade na escolha de implementações mais eficientes ou na modificação da arquitetura do projeto caso essa não satisfaça as restrições, diminuindo, consequentemente, o *time-to-market* e aumentando a produtividade global do projetista. Os resultados são obtidos a partir de três diferentes implementações de somadores completos de 1 bit.

Ikhwan et al. (2006) apresentam um *framework* para estimação de consumo de energia em SoC, no nível de sistema, denominado PowerViP. O

PowerViP foi construído sobre um *framework* de simulação no nível de sistema também desenvolvido pelos autores, o Vip (SAMSUNG, 2004). No PowerViP diferentes técnicas de modelagem de consumo de energia foram empregadas para cada componente: núcleos de processador, barramentos, blocos de IP personalizados e memórias.

A construção do PowerViP se dá em três etapas. A primeira consiste em estabelecer um ambiente de análise de consumo de energia no nível de portas ou RTL, por componente IP, para extrair (caracterizar) seus valores de potência. O segundo passo consiste em construir um modelo de consumo de energia com os números de potência extraídos. O último passo consiste em integrar (*annotate*) o modelo de consumo de energia dentro do modelo ESL do componente IP para gerar valores de potência durante a simulação no nível de sistema.

Klein et al. (2007a, 2007b) apresentam um *framework* para estimação de consumo de energia denominado PowerSC. Esse *framework* instrumenta SystemC para caracterização de potência, modelagem e estimação do consumo de energia em múltiplos níveis de abstração. A ideia básica é a extensão da biblioteca SystemC adicionando classes C++ que viabilizem a estimação do consumo de energia. De acordo com os autores, PowerSC permite modelagem de energia consistente desde o mais alto até o mais baixo nível de abstração.

PowerSC fornece uma API que viabiliza a integração de técnicas macromodelagem alternativas. Para utilizar o *framework* são necessárias apenas duas modificações na descrição SystemC do módulo que irá utilizá-lo. As modificações consistem na adição de duas linhas de código: Uma para incluir o cabeçalho principal, “powersc.h”, e outra para chamada de uma macro PowerSC que imprime os resultados da estimação.

Quando um modelo (e.g., uma classe) é compilado utilizando a biblioteca PowerSC, em vez de ser gerado um modelo SystemC executável convencional, é gerado um modelo executável aumentado, o qual é instrumentado para coletar informações de estatísticas de sinais durante a simulação. Assim, por meio de uma instrumentação adequada, os elementos do projeto são monitorados e informações de consumo de energia são dinamicamente registradas.

Os autores ressaltam que os tipos de dados, sinais e módulos SystemC não precisam ser manualmente modificados, visto que PowerSC os modificam automaticamente. No Código 1 são apresentadas as modificações necessárias, descritas pelos autores, para o uso deste *framework*. A linha 2 do Código 1 é referente à inclusão do cabeçalho principal da abordagem PowerSC. A linha 17 é referente à macro PowerSC que imprime os resultados da estimação.

Código 1 – Modificações necessárias para o uso do *framework* PowerSC.

```
1  #include <systemc.h>
2  #include<powersc.h> //Modificação obrigatória
3  #include "muls32.h"
4  ...
5  SC_MODULE (rtl_example){
6    sc_in_clk clk;
7    ...
8    sc_signal<sc_uint<2>> sig1, sig2;//Sinais
9    ...
10   MulS32 *mult;
11   ...
12  };
13  ...
14  int sc_mais (int argc, char **argv){
15    ... //Instanciação dos módulos
16    sc_start();
17    PSC_REPORT_POWER;//Modificação obrigatória
18    return(0);
19  }
```

Fonte: KLEIN et al., 2007b.

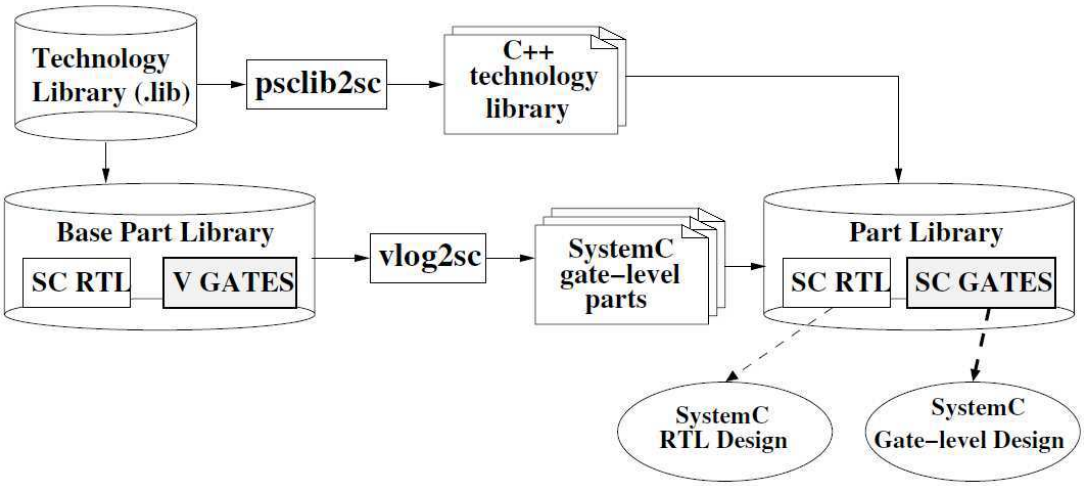
Os modelos de consumo de energia são construídos com informações de níveis mais baixos. Por exemplo, os autores criam modelos de consumo de energia no nível de abstração RTL com informações de consumo de energia no nível de abstração de portas.

Para validar o *framework*, os autores criaram modelos de consumo de energia (macromodelos) por meio de um processo de caracterização de energia (processo de macromodelagem) utilizando informações do nível de abstração de portas lógicas. A criação desses macromodelos se deu da seguinte forma (Figura 5):

Uma biblioteca denominada *Base Part Library* (contendo somadores e multiplicadores) foi construída a partir de uma biblioteca de tecnologia de fabricação no formato *Liberty* com uma ferramenta do pacote Forte's Cynthesizer da empresa Cadence (CADENCE, 2015). Os componentes

criados foram descritos em SystemC no nível RTL (SC_RTL) e Verilog no nível de portas lógicas (V_GATES).

Figura 5 - Processo de caracterização no nível de abstração de portas lógicas para o *framework* PowerSC.



Fonte: KLEIN et al., 2007b.

Porém, como os autores pretendiam uma representação unificada utilizando a linguagem SystemC dentro do *framework*, foram implementados dois conversores:

- vlog2sc: Teve como objetivo traduzir cada *netlist*[19] encontrado em V_GATES para uma descrição SystemC equivalente no nível de portas;
- psclib2sc: Teve como objetivo converter a biblioteca de fabricação do formato *Liberty* em uma representação C++.

Os arquivos produzidos por estes dois conversores resultaram na *SystemC gate-level part library* (SC_GATES), a qual substitui a biblioteca Verilog (V_GATES).

Desta maneira, o usuário poderá selecionar a estimação do consumo de energia entre o nível de abstração RTL ou de portas lógicas, apenas escolhendo entre as bibliotecas SC_GATES e SC_RTL.

Para estender a biblioteca PowerSC com uma nova técnica de macromodelagem, duas classes devem ser definidas: `psc_macromodel` e `psc_macromodel_parms` (Código 2).

[19] Uma *netlist* pode ser definida como uma representação do circuito, em termos de conexões entre portas lógicas.

Código 2 – Principais aspectos da classe psc_macromodel.

```
1  class psc_macromodel {
2  ...
3  public :
4    virtual void init_power_map( ) ;
5    virtual double get_power ( const psc_macromodel_parms & );
6  ...
7  };
```

Fonte: KLEIN et al., 2007b.

O usuário deverá criar o código das funções virtuais, as quais são utilizadas internamente pelo PowerSC para avaliar o consumo de energia. São elas:

- `init_power_map`: Esta função inicializa a estrutura interna, definida pelo usuário, com informações de energia oriundas da etapa de caracterização.
- `get_power`: Esta função possui os detalhes de como computar a dissipação de potência baseada no parâmetro `psc_macromodel_params`. Por exemplo, atributos advindos deste parâmetro poderiam ser estatísticas de sinais. PowerSC chama esta função internamente para gerar relatórios de dissipação de potência de um modelo de consumo de energia específico.

Um `psc_macromodel` é automaticamente criado para cada componente da biblioteca, visto que, técnicas para macromodelagem geralmente necessitam de um comportamento específico de cada componente (e.g., tabelas inicializadas com diferentes valores de dissipação de potência).

Park et al. (2007) apresentam uma metodologia para gerar uma hierarquia de modelos de consumo de energia para estimação em blocos de *hardware* personalizados, permitindo um *trade-off* entre precisão de estimação de consumo de energia, esforço de modelagem e velocidade de estimação. De acordo com o autor, sua abordagem permite diversas explorações no nível de sistema, tal como observar o efeito da técnica de *clock gating*[20] e efeitos de ajustes de parâmetros no nível de aplicação no

[20] A técnica de *clock gating* baseia-se na interrupção do sinal de *clock* em submódulos funcionais ociosos do SoC. Essa técnica é utilizada para redução de energia dinâmica em circuitos digitais (VARMA et al., 2008; VISWANATH et al., 2009).

sistema de energia. O autor também afirma que a precisão da estimação utilizando sua metodologia é próxima a uma estimação no nível de portas.

O autor implementou sua metodologia sobre um IP de decodificação de predição de vídeo H.264. Em sua metodologia os modelos de consumo de energia são baseados em árvores. Cada nó da árvore representa um estado de energia do IP. Para cada nó é atribuído um valor de potência. O nível zero (também conhecido como raiz da árvore) representa o IP em questão.

O nível 1 representa o IP dividido em dois nós. Estes nós representam o IP nos estados Ativo ou Ocioso. Ao descer na árvore (ou seja, aumentando seu nível) os modelos de consumo de energia possuem mais nós, consequentemente uma estimação de consumo de energia que leva em consideração mais detalhes. Assim, nós no nível N-1 podem ser decompostos em modelos de consumo de energia detalhados (nível N) que consideram o efeito de diferentes características dos dados de entrada para o cálculo da potência.

Estes efeitos podem ser, por exemplo, a distância Hamming[21] e o número de bits 1 no dado. A metodologia engloba várias granularidades de modelagem de consumo de energia e oferece ao projetista um *trade-off* entre a precisão da estimativa, o esforço de modelagem do *design* e tempo de simulação.

Hsieh, Yeh e Huang (2010) desenvolveram um *framework* para estimação de consumo de energia ESL, utilizando interfaces de modelo de consumo de energia. De acordo com o autor, utilizando a interface de modelo de consumo de energia proposta, pode-se facilmente integrar vários modelos de consumo de energia na plataforma. Neste sentido, os projetistas podem escolher em utilizar modelos de consumo de energia "grosseiros" (*coarse-grained*) ou refinados (*fine-grained*), de acordo com a relação custo-benefício entre precisão e custo computacional.

Uma das vantagens da pesquisa é que utilizando a interface de modelo de consumo de energia definida, pode-se facilmente integrar vários modelos de consumo de energia sem modificar o projeto original no nível de sistema. Como estudo de caso foi utilizado PAC (*Parallel Architecture Core*) *Duo system* (LIN et al., 2008).

[21] Número de bits invertidos entre dois dados consecutivos (PARK et al., 2007).

Ahuja (2010), em suas soluções, inclui a utilização de registros de simulações ESL e o seu mapeamento para níveis de abstração mais baixos para estimação de consumo de energia. Assim, o mecanismo para estimação de consumo de energia RTL pode ser reusado e executado no nível de sistema. De acordo com o autor, a metodologia proposta pode ser dividida em cinco passos:

1. Converter um modelo de alto nível, descrito em C++ ou SystemC, para um RTL equivalente. Para tanto, o autor recomenda o uso da ferramenta Catapult (CALYPTO, 2015);
2. Simular o modelo alto nível e gerar seu VCD (*Value Change Dump*);
3. Aplicar um algoritmo de extração de atividades e ciclos de trabalho (*duty-cycle*) de várias variáveis do projeto no arquivo VCD gerado durante a simulação realizada no passo 2;
4. Gerar o mapeamento de variáveis no alto nível para sinais RTL e usar as saídas do algoritmo mencionado no passo 3 para criar entradas apropriadas para realização da análise de potência probabilística[22] usando um estimador de consumo de energia RTL, tal como a ferramenta PowerTheater (ARM, 2012) e
5. Coletar os números de potência relatados pela ferramenta de estimação de consumo de energia RTL.

Uma análise feita por Hsieh, Yeh e Huang (2010) aponta como um dos problemas desta metodologia o fato de que, embora registros de simulações possam ser obtidos rapidamente no nível de sistema, utilizar um mecanismo de estimação RTL é ainda muito lento para estimação de consumo de energia do sistema completo. Um resumo dos resultados obtidos com essa abordagem pode ser visto em (AHUJA et al., 2009).

Liu et al. (2010) apresentam um *framework* de modelagem e estimação de consumo de energia em alto nível baseada em uma integração coordenada de SystemC e AOP (*Aspect-Oriented Programming*). A pesquisa mantém o modelo SystemC "limpo", evitando modificações no núcleo SystemC ou adição de chamadas de API dentro do modelo de funcionalidade

[22] Na análise de potência probabilística, um sinal lógico é visto como processo de "zeros e uns" aleatórios, com certas características estatísticas. Não se sabe exatamente o momento em que ocorre os chaveamentos do sinal. Em vez disso, são prescritas ou derivadas várias características estatísticas numéricas do sinal são. A dissipação de potência do circuito é derivada de quantidades estatísticas (YEAP, 1998) .

SystemC. Na pesquisa, AspectC++ é usado para definir aspectos especiais de informação de potência.

De acordo com os autores, esses aspectos podem ser vistos como arquivos de configuração para integrar API de informação de potência e o modelo de funcionalidade SystemC. Um ponto de destaque da pesquisa é que o modelo suporta estimação de consumo de energia utilizando múltiplos macromodelos e estimação de consumo de energia de múltipla precisão. Os experimentos foram realizados sobre um somador de n bits como estudo de caso, sendo apresentados resultados da eficácia do método.

Trabelsi et al. (2011) apresentam uma abordagem que permite levar em consideração o consumo de energia logo no início do fluxo de projeto durante a cossimulação do sistema. Em sua abordagem é destacado, como sendo original, o fato de permitir estimação de energia tanto para IP *white-box* como para IP *black-box*. Para IP *white-box* são inseridos contadores de atividades dentro do código dos IP, a fim de detectar ocorrências de atividades e determinar o consumo de energia dessas atividades durante a simulação do sistema. Para IP *black-box*, módulos de estimação de consumo de energia são conectados entre os IP a fim de detectar suas atividades por meio dos sinais que eles trocam durante a simulação.

As simulações foram realizadas no nível CABA (*cycle-accurate bit-accurate*), utilizando SystemC, a fim de se obter estimativas de consumo precisas. O nível CABA é um nível de abstração mais alto que o RTL, permitindo simulações mais rápidas do que aquelas realizadas usando RTL.

De acordo com o autor, para migrar do RTL para o nível CABA, detalhes de implementação de *hardware* são escondidos da parte de processamento do sistema, enquanto se preserva o comportamento do sistema no nível de ciclo de *clock (cycle-acurate)*. *Bit-accurate,* do termo CABA *(cycle-accurate bit-accurate),* significa que um protocolo de comunicação é usado entre os componentes no nível de *bit.*

A implementação da abordagem consiste em uma extensão do *framework* Gaspard2 (WEST TEAM, 2015). Gaspard2 é um *framework*, baseado em MDE[23] (*Model Driven Engineering*), de *codesign* de SoC para

[23] MDE (*Model Driven Engineering*) é uma metodologia de desenvolvimento de *software* o qual foca na criação e exploração de representações abstratas de conhecimentos e atividades que governam um particular domínio de aplicação em vez de algoritmos (SCHMIDT, 2006).

descrever tanto partes da arquitetura como partes da aplicação de um sistema em um alto nível de abstração. A partir de um sistema modelado nesse *framework* pode ser gerado o código SystemC correspondente.

A contribuição da pesquisa de Trabelsi *et al.* (2011) no *framework* consistiu em integrar estimação de consumo de energia no nível de modelagem do Gaspard2, como também no nível de simulação, permitindo a automação da estimação de energia no fluxo de projeto do Gaspard2. Na implementação foram gerados módulos de estimação de consumo de energia para IPs SystemC na biblioteca de IP Gaspard2. A abordagem MDE do Gaspard2 permite a geração de código SystemC do SoC inteiro com os estimadores integrados. O código pode ser integrado dentro do simulador SystemC e estimativas de energia podem ser apresentadas durante a simulação.

De acordo com Kuehnle, Wagner e Becker (2011), ferramentas convencionais de análise de consumo de energia RTL precisam, principalmente, de dois tipos de informação de entrada: de um lado, um modelo do projeto que já passou pela etapa de "*place and route*", permitindo assim calcular as cargas capacitivas de cada *net*, avaliando os números de *fan-out*[24] e as características das primitivas (LUTS para FPGA) e, assim, o consumo de energia por *net* e por atividade. Do outro lado, atividades dinâmicas são coletadas durante a simulação pós "*place and route*" no arquivo *Value Change Dump* (VCD).

A pesquisa utiliza uma metodologia de estimação de consumo de energia baseada na atividade do projeto do CI. A atividade de chaveamento é estimada de acordo com uma computação da distribuição de probabilidade do conjunto de dados de entrada e a distribuição de probabilidade derivada de conjuntos de dados internos e de saída. De acordo com o autor, a análise dos dados de entrada para predizer a frequência de chaveamento na estrutura de *hardware* sintetizada esperada aumenta a precisão da estimação de consumo.

Como base para sua pesquisa, Kuehnle, Wagner e Becker (2011) utilizaram uma ferramenta de tradução de VHDL para SystemC, denominada V2SC (MAZDAK AND ALBORZ DESIGN AUTOMATION, 2015). A

[24] Em circuitos integrados o *fan-out* de uma porta lógica define o número máximo de entradas que a saída dela pode alimentar.

parte de estimação de consumo foi implementada como uma extensão da ferramenta V2SC. Ao traduzir um código VHDL para SystemC, a ferramenta converte, por exemplo, um tipo inteiro VHDL por um tipo sc_int_power e não pelo tipo padrão SystemC sc_int. O sc_int_power é um novo tipo que foi construído no projeto para realizar todo cálculo e estimação de consumo de energia. A abordagem calcula e estima potência sobrecarregando os operadores aritméticos (+,-,*./) e quaisquer operadores e funções que possam atuar com o tipo sc_int_power.

Outra pesquisa projetou o Powersim (GIAMMARINI; CONTI; ORCIONI, 2011; GIAMMARINI; ORCIONI, 2013). Powersim é uma bilbioteca C++ desenvolvida para ser usada dentro de uma implementação SystemC (CONTI et al., 2011). De acordo com os autores, seu principal propósito é a simulação da complexidade computacional e consumo de energia de sistemas digitais descritos no ESL. Algumas de suas principais características são:

- Dispensa a necessidade de mudanças do código fonte que descreve o sistema;
- Possibilita a atribuição de diferentes modelos de energia para cada operação realizada em cada tipo de dado;
- Possibilita a adição de novos modelos de consumo de energia;
- Possibilita a extensão de sua funcionalidade por meio de uma implementação orientada a objetos baseada em C++.

Em seu desenvolvimento, algumas classes SystemC foram modificadas, como também foram adicionadas novas classes à biblioteca SystemC. A biblioteca Powersim estima o consumo de energia de um sistema interagindo com operadores, funções matemáticas e funções lógicas dos módulos que constituem o sistema. Por meio do monitoramento de operadores e funções matemáticas a biblioteca Powersim pode fornecer uma estimação da complexidade de um sistema.

Greaves e Yasin (2014) desenvolveram uma biblioteca que estende a capacidade de SystemC para agregação de dados de consumo de energia e *layout* físico (e.g., área dos componentes). A biblioteca, denominada TLM POWER3, fornece meios para cálculo do consumo de energia dinâmica. O trabalho desenvolvido dá suporte à modelagem de consumo de energia *phase /mode*. Na modelagem fase/modo (*phase /mode)* a dissipação de

potência de um bloco IP é determinada a partir do seu estado atual. O estado do bloco é caracterizado por ambos, sua fase (*phase*) e seu modo (*mode*). Um modo é um modo DPM particular (*Dynamic Power Management*), por exemplo, *on, sleep, off*. Uma fase é caracterizada por sua potência e seu tempo de duração, por exemplo, *wait, read, compute*.

Além do suporte à modelagem fase/modo, o trabalho dá suporte ao registro de dados de energia por transação em modelos TLM. Seu trabalho estende as funcionalidades das bibliotecas PKtool (VECE; CONTI, 2009) e TLM POWER2 (MOY, 2010). Porém, existem algumas diferenças a serem consideradas entre as funcionalidades das bibliotecas TLM POWER3 e TLM POWER2. A biblioteca TLM POWER2 define unidades físicas para potência e energia da mesma maneira que SystemC define unidades físicas para tempo. Os operadores aritméticos são sobrecarregados para obter um comportamento esperado. Por exemplo, potência multiplicada por um tempo resultará em energia. No TLM POWER3 também foram adicionadas unidades físicas para tensão, distância e área.

No âmbito de estimação de consumo em alto nível de abstração, a maioria das pesquisas relacionadas investigadas implementaram suas abordagens de estimação de consumo utilizando SystemC. No Quadro 3 essas pesquisas são resumidas de acordo com seu nível de abstração para estimação de consumo, como também por suas contribuições e características.

Quadro 3 - Quadro comparativo das pesquisas relacionadas sobre abordagens para estimação de consumo de energia. (continua)

Autor(es)	Ano	Nível de abstração	Contribuições e características	Limitações
Tiwari, Malik e Wolfe	1994	ESL	Apresenta uma técnica para a modelagem do consumo de energia de processadores. Primeira pesquisa que foca na modelagem de consumo de energia de processadores	A técnica descrita pelos autores pode não ser aplicável para processadores onde o número de instruções sejam alguns milhares. O processo de caracterização pode levar muito tempo. Não fornece uma implementação da técnica.
Caldari et al.	2003	ESL	Apresenta um conjunto de orientações para a integração de informação de consumo de energia em modelos de sistemas descritos no ESL.	Baseia-se na criação de macromodelos a partir do conhecimento das possíveis implementações. Não fornece uma implementação do arcabouço.
Damaševičius	2006	RTL	Um arcabouço a para estimação do consumo de energia que consiste na extensão e modificação da biblioteca SystemC. O processo de modelagem de área, atraso e características de consumo de energia utilizando SystemC para RTL, torna-se mais rápido do que usando, por exemplo, a linguagem VHLD para obter tais características utilizando um sintetizador comercial.	Os modelos SystemC devem ser modificados para a introdução de novos sinais e métodos. A simulação de modelos SystemC que utilizam o arcabouço é, em média, 68% mais lenta do que modelos SystemC que não o utilizam. Possui foco apenas em SystemC. Não fornece a implementação do arcabouço.

Quadro 3 - Quadro comparativo das pesquisas relacionadas sobre abordagens para estimação de consumo de energia. (continua)

Autor(es)	Ano	Nível de abstração	Contribuições e características	Limitações
Ikhwan et al.	2006	ESL	Um arcabouço para a estimação do consumo de energia em SoC no nível de sistema denominado PowerViP. O arcabouço permite a utilização de diferentes modelos de consumo de energia para os diferentes tipos de elementos que compõem o SoC (e.g., memórias, processadores, barramentos).	Não fornece uma implementação do arcabouço. Apesar de o arcabouço prover heterogeneidade em termos de uso de modelos de consumo de energia, melhorando a precisão da estimação, o arcabouço não provê flexibilidade para que se utilize outras técnicas para a estimação do consumo de energia de maneira concomitante, durante a simulação.
Klein et al.	2007	ESL, RTL e portas lógicas	Um arcabouço para estimação de consumo de energia que consiste na extensão da biblioteca SystemC. Permite o uso de diferentes macromodelos. Fornece uma implementação do arcabouço. Fornece documentação e informações necessárias para uso da abordagem.	Em seus experimentos, o arcabouço foi validado apenas para criação de modelos de energia no nível RTL e no nível de portas. Possui foco apenas em SystemC.
Park et al.	2007	ESL	Uma metodologia para gerar uma hierarquia de modelos de consumo de energia para estimação de consumo de energia de blocos de *hardware* personalizados. Permite diversas explorações no nível de sistema, tal como observar o efeito da técnica de *clock gating* e efeitos de ajustes de parâmetros no nível de aplicação no sistema de energia.	Possui foco apenas para a estimação do consumo de energia de blocos de *hardware* personalizados. Não fornece uma implementação da metodologia.

Quadro 3 - Quadro comparativo das pesquisas relacionadas sobre abordagens para estimação de consumo de energia. (continua)

Autor(es)	Ano	Nível de abstração	Contribuições e características	Limitações
Ahuja et al.	2010	ESL	Um arcabouço para estimação de consumo de energia que atua em conjunto com diversas outras ferramentas.	Não fornece uma implementação do arcabouço. Embora registros de simulações possam ser obtidos rapidamente no nível de sistema, utilizar um mecanismo de estimação de consumo de energia baseado no mapeamento para níveis de abstração mais baixos, tal qual o RTL, é ainda muito lento para estimação de consumo de energia do sistema completo.
Liu et al.	2010	ESL	Um arcabouço para estimação de consumo de energia que combina SystemC e AOP. O arcabouço suporta estimação de consumo de energia utilizando múltiplos macromodelos e estimação de consumo de energia de múltipla precisão. Não modifica a biblioteca SystemC.	Apesar de permitir o uso de múltiplos macromodelos, estes devem estar descritos no mesmo nível de abstração. Possui foco apenas em SystemC. Não fornece a implementação do arcabouço.
Hsieh, Yeh e Huang	2010	ESL	Um arcabouço que disponibiliza uma interface de energia que pode integrar vários modelos de consumo de energia em uma plataforma virtual ESL. O arcabouço prevê o uso de modelos de consumo de energia "grosseiros" (*coarse-grained*) ou refinados (*fine-grained*)	Não fornece uma implementação do arcabouço. Não permite a simulação de modelos em múltiplos níveis de abstração.

Quadro 3 - Quadro comparativo das pesquisas relacionadas sobre abordagens para estimação de consumo de energia. (continua)

Autor(es)	Ano	Nível de abstração	Contribuições e características	Limitações
Trabelsi et al.	2011	CABA	Integração de estimação de consumo de energia no nível de modelagem do Gaspard2 como também no nível de simulação. Permite estimação de energia tanto para *IP white-box* como para *IP black-box*. Modifica o núcleo do simulador SystemC	Possui foco apenas em SystemC. Não fornece a implementação do arcabouço.
Kuehnle, Wagner e Becker	2011	ESL	Extensão da ferramenta V2SC para suportar estimação de consumo de energia. Ao traduzir um código VHDL para SystemC, a ferramenta converte tipos VHDL para novos tipos SystemC, projetados para realizar todo cálculo e estimação de consumo de energia. Não modifica a biblioteca SystemC	Os modelos devem ser escritos em VHDL para que sejam convertidos em modelos SystemC. Possui foco apenas em SystemC. Não fornece a implementação do arcabouço.

Quadro 3 - Quadro comparativo das pesquisas relacionadas sobre abordagens para estimação de consumo de energia. (conclusão)

Autor(es)	Ano	Nível de abstração	Contribuições e características	Limitações
Giammarini, Conti e Orcioni	2011	ESL	Extensão e modificação da biblioteca SystemC para suportar estimação de energia. Monitora os operadores SystemC a fim de calcular o consumo de energia dos modelos SystemC. Por exemplo, uma operação de soma de entre dois inteiros de 8 bits consome 30.45 nJ. Não requer mudanças no código da aplicação. Fornece uma implementação da abordagem.	Possui foco apenas em SystemC.
Greaves e Yasin	2014	ESL	Extensão e modificação da biblioteca SystemC para agregação de dados de consumo de energia e *layout* para o acúmulo e estimação do uso de energia dinâmica. Tem como foco modelos SystemC TLM. Fornece um esboço de documentação da abordagem.	Possui foco apenas em SystemC. Fornece, mediante solicitação, uma versão não concluída da implementação do arcabouço.

Dependendo do projeto, para garantir maior precisão na estimação do consumo de energia, pode ser necessário estimar o consumo de energia do sistema ou parte dele utilizando diferentes elementos: diferentes abordagens de estimação, ferramentas ou, até mesmo, modelos descritos em variadas linguagens e/ou níveis de abstração.

Por exemplo, Ikhwan et al. (2006) destacam a importância do uso de diferentes modelos de consumo de energia para os diferentes blocos de *hardware* que compõem o projeto de um SoC. Os autores alertam que, para os diferentes tipos de componentes de um SoC, tais como processadores, memórias, barramentos, etc, geralmente se utiliza diferentes abordagens de estimação para o consumo de energia.

Liu et al. (2010) e Hsieh, Yeh e Huang (2010) também mencionam a importância da estimação de consumo de energia utilizando múltiplos modelos de consumo de energia (macromodelos). Os autores afirmam que o uso de diferentes macromodelos podem melhorar a precisão da estimação do consumo de energia, visto que partes do sistema podem utilizar modelos com diferentes precisões.

Todos os autores mencionados têm como foco a tentativa de garantir uma boa precisão na estimação do consumo de energia, apresentando um bom desempenho durante a simulação. Alguns deles, tais quais Ikhwan et al. (2006), Hsieh, Yeh e Huang (2010), Liu et al. (2010) e Klein et al. (2007a, 2007b), também tentam focar na criação de abordagens que permitam uma heterogeneidade em termos de macromodelos, para realização mais precisa da estimação do consumo de energia.

Porém, mesmo permitindo o uso de diferentes macromodelos para a estimação do consumo de energia, nenhuma das abordagens apresentadas permite que dois modelos distintos (e.g., dois modelos SystemC) possam ser simulados concomitantemente utilizando diferentes abordagens de estimação. No geral, isto acontece porque na maioria das abordagens, quando implementadas, modificam a biblioteca SystemC ou a estende, impossibilitando o uso de duas abordagens diferentes durante a simulação.

Além disso, para todas as abordagens pesquisadas, a estimação do consumo de energia é realizada em apenas um nível de abstração específico por vez. Por exemplo, Klein et al. (2007a, 2007b) permite a estimação do consumo de energia em mais de um nível de abstração, porém, o usuário

deve escolher um único nível de abstração para a estimação do consumo de energia do sistema inteiro.

Uma solução para os problemas acima mencionados, seria desenvolver uma abordagem que permitisse que esses modelos SystemC fossem distribuídos e simulados, de maneira sincronizada, utilizado seus diferentes simuladores SystemC. Porém, consiste em um desafio a criação de um ambiente de simulação heterogêneo e distribuído, o qual permita que estes se comuniquem e troquem informações de modo sincronizado. O uso de uma arquitetura de propósito geral para modelagem e simulação distribuída viabilizaria a criação desse ambiente integrado e heterogêneo.

Em nenhuma das pesquisas relacionadas sobre estimação de consumo de energia foram encontrados relatos do uso de uma arquitetura de propósito geral para modelagem e simulação distribuída, tal qual a HLA, a fim de permitir a criação de um ambiente de simulação integrado e heterogêneo, composto por diferentes ferramentas (e.g., simuladores), modelos descritos em diferentes linguagens e/ou níveis de abstração, que proporcione o uso de diferentes abordagens de estimação de consumo de energia para cada modelo durante o processo de simulação.

Neste sentido, a presente pesquisa tem como objetivo geral desenvolver uma abordagem utilizando *High Level Architecture* (HLA) para permitir a criação de um ambiente de simulação integrado e heterogêneo, composto por diferentes ferramentas e modelos. Estes modelos podem ser descritos em diversas linguagens e/ou níveis de abstração, como também utilizar diferentes abordagens a estimação do consumo de energia. O uso da HLA permite que os elementos que compõe este ambiente heterogêneo possam ser simulados de maneira sincronizada e distribuída. A abordagem deve proporcionar a coleta e o agrupamento de dados de estimação de consumo de energia de modo centralizado.

Para validar a abordagem desenvolvida, foi necessário a escolha de, pelo menos, uma das abordagens de estimação de consumo de energia pesquisadas. Dentre as pesquisas apresentadas destacam-se as pesquisas de Klein et al. (2007a, 2007b), Giammarini, Conti e Orcioni (2011) e Greaves e Yasin (2014). devido à facilidade de uso de suas abordagens, como também devido a disponibilização de suas implementações.

Qualquer uma dessas pesquisas poderia ser utilizada para a validação da presente abordagem. Porém, visto que parte dos estudos de caso seriam realizados utilizando a plataforma MPSoCBench (DUENHA et al., 2014), e, o *framework* PowerSC, desenvolvido por de Klein et al. (2007a, 2007b), foi utilizado com sucesso para modelagem do consumo de energia dos processadores MIPS e SPARC disponibilizados por essa plataforma, seu *framework* foi escolhido para a validação da presente pesquisa.

A plataforma MPSoCBench (detalhada na Seção 4.6) foi utilizada como ferramenta para criação de um modelo utilizado no estudo de caso para a validação da presente pesquisa.

3.2. Considerações Finais do Capítulo

Neste capítulo, foram apresentados as pesquisas relacionadas para fundamentação da presente pesquisa, destacando suas principais contribuições, características e limitações, com destaque para a contribuição da abordagem desenvolvida.

No Capítulo 4 será apresentada a abordagem desenvolvida, assim como aspectos essenciais de sua implementação. Serão abordados detalhes de implementação sobre a sincronização dos federados, publicação e assinatura de objetos, como também detalhes de configuração do ambiente de simulação. Também serão apresentadas as plataformas MPSoCBench e Ptolemy, as quais foram utilizadas na construção de parte dos estudos de caso.

Capítulo 4

Abordagem para a Estimação do Consumo de Energia – PowerHLA

Neste capítulo, será apresentada a abordagem desenvolvida, denominada PowerHLA. Também serão apresentadas as plataformas MPSoCBench e o arcabouço Ptolemy, os quais foram utilizados na construção de parte dos estudos de caso para a validação da pesquisa ora apresentada. Os elementos que compõem a abordagem PowerHLA serão descritos nas próximas seções.

4.1. Considerações Iniciais

A abordagem fará uso da HLA RTI (*Run-Time Infrastructure*) de código aberto CERTI (NOULARD; ROUSSELOT; SIRON, 2009), como meio de interconexão entre as simulações.

A abordagem, além de permitir a simulação distribuída e sincronizada de modelos heterogêneos, gerará uma saída com valores de energia amostrados em tempos pré-definidos pelo usuário para cada um dos desses. Essas saídas obedecerão ao formato CSV (*comma separated values*), tornando possível a geração de gráficos de maneira fácil, por meio de ferramentas como o Openoffice Calc ou Microsoft Excel. Todos os elementos da PowerHLA foram desenvolvidos utilizando C++. Um desses elementos também faz uso de SystemC.

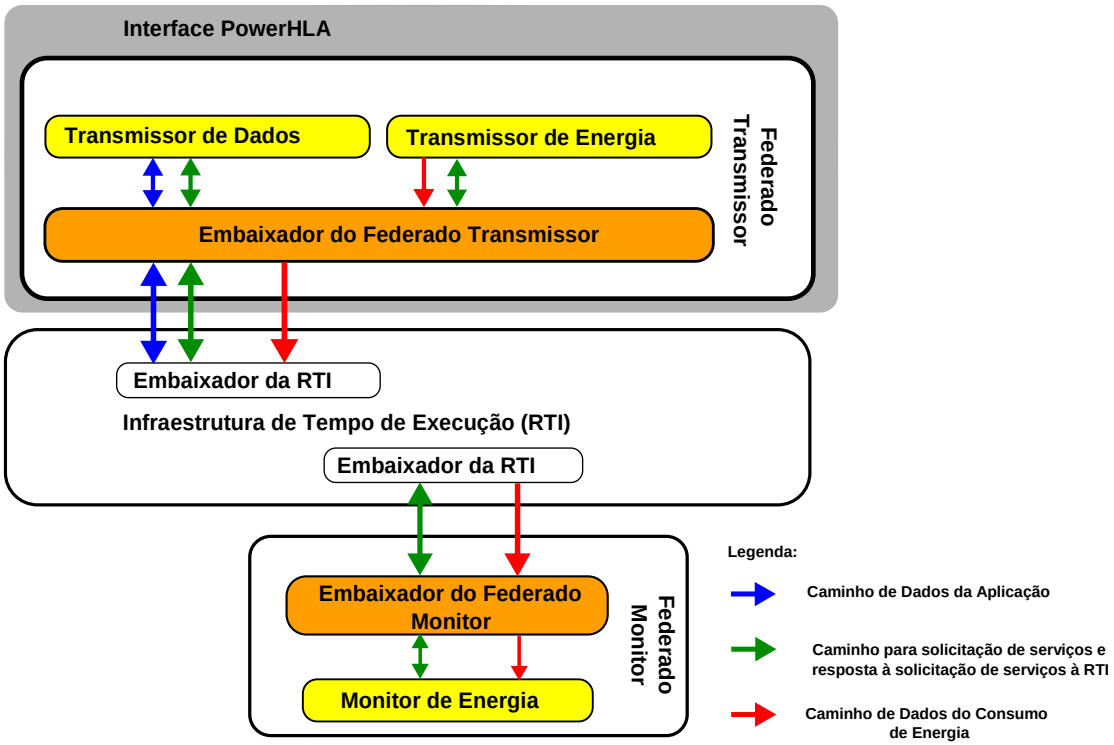
A abordagem foi validada a partir de experimentos realizados utilizando modelos criados pela plataforma MPSoCBench (DUENHA et al., 2014) assim como com o uso de um modelo RTL de um DPCM (*Differential Pulse-Code Modulation*) descrito em SystemVerilog/Verilog. A plataforma será explicada em detalhes na Seção 4.6.

A abordagem de estimação de consumo de energia utilizada durante os experimentos com a plataforma MPSoCBench será a, já discutida, PowerSC (KLEIN et al., 2007a).

4.2. Descrição da Abordagem Desenvolvida

Na Figura 6 é apresentado o diagrama em blocos da abordagem PowerHLA. A seguir, a descrição dos elementos que compõem a abordagem.

Figura 6 - Abordagem PowerHLA.



Fonte: O autor, 2015.

4.2.1. Transmissor de Dados

Este elemento é responsável por enviar e receber dados de aplicação (setas azuis, Figura 6) trocados pelo modelo simulado e também por realizar a sincronização do mesmo.

A maneira como é realizada a sincronização e a troca dos dados entre modelos simulados vai depender cada aplicação. Consequentemente, cabe ao usuário da abordagem PowerHLA escrever a parte do código relativa à sincronização e troca de dados entre os federados.

Na Seção 5.1.1.1 é apresentado um estudo de caso descrevendo a implementação do código para sincronização dos modelos simulados.

Os dados de aplicação enviados pelo modelo simulado deverão ser enviados para outros federados por meio do serviço HLA RTI *Update Attribute Values*, o qual deverá estar presente no do Transmissor de Dados. A recepção dos dados de aplicação de outros modelos ocorre via a chamada do serviço HLA RTI de retorno (*callback*) *Reflect Attribute Values*[†], a qual deverá estar presente no Embaixador do Federado Transmissor.

O conjunto composto pelo Transmissor de Dados, Transmissor de Energia e Embaixador do Federado Transmissor será denominado Federado Transmissor. Todo modelo simulado, que faça uso da abordagem PowerHLA, terá necessariamente um Federado Transmissor.

A federação pode ter muitos federados transmissores, visto que cada federado transmissor representa uma simulação distribuída (i.e., um federado). No decorrer do documento será usado o termo, no plural, "federados transmissores" visto que, geralmente, é utilizado mais de um deles durante a execução da federação.

4.2.2. Transmissor de Energia

O Transmissor de Energia é responsável por enviar dados de consumo de energia (setas vermelhas, Figura 6) acumulados por cada um dos componentes monitorados do modelo simulado. Um componente monitorado representa um trecho de código, geralmente um módulo funcional, o qual o projetista deseja monitorar. O Transmissor de Energia pode monitorar vários de componentes.

Os dados de consumo de energia são enviados para o Federado Monitor quando um intervalo predefinido pelo usuário é alcançado, por meio do serviço HLA RTI *Update Attribute Values*. Ao enviar os dados do consumo de energia de cada um dos componentes monitorados, o Transmissor de Energia define como zero todas as respectivas variáveis relativas à energia acumulada desses componentes.

As informações do consumo de energia enviadas pelo Federado Transmissor devem ser recebidas de maneira ordenada pelo Federado Monitor. Por este motivo, Federado Transmissor deverá ser obrigatoriamente regulador de tempo, o que possibilita o envio de mensagens TSO. Consequentemente, este federado deverá possuir um valor de *lookahead*. O

valor do *lookahead* deverá ser definido pelo usuário da abordagem de acordo com aspectos sincronização (ver Seção 4.2.8).

4.2.3. Embaixador do Federado Transmissor

A comunicação de um Federado Transmissor para a RTI é estabelecida via Embaixador do Federado Transmissor. Cada instância de um Federado Transmissor deverá possuir uma instância de um Embaixador do Federado Transmissor. O Embaixador do Federado Transmissor é responsável pela recepção dos dados de aplicação de outros modelos, assim como por receber respostas às solicitações de serviços feitos à RTI (setas verdes, Figura 6), via chamadas de retorno (*callback*).

Por exemplo, quando o modelo simulado recebe um dado de outro modelo, ele o recebe via a chamada de retorno *Reflect Attribute Values*†, contida no Embaixador do Federado Transmissor.

Em relação às solicitações de serviços, por exemplo, o Federado Transmissor pode solicitar um avanço de tempo à RTI. Quando a solicitação de avanço de tempo é concedida, a RTI envia uma resposta para o Federado Transmissor via seu Embaixador (o Embaixador do Federado Transmissor) por meio da chamada de retorno *Time Advance Grant*†. São exemplos de serviços iniciados pela RTI (RTI *Initiated Services*): *discover*†*, reflect*†*, time advance grant*†, etc.

O Embaixador do Federado Transmissor constituiu parte do código desenvolvido para a criação da abordagem PowerHLA.

4.2.4. Infraestrutura de Tempo de Execução (RTI)

Infraestrutura de Tempo de Execução (RTI) é o software que fornece uma interface de serviços comum durante uma execução de federação HLA para sincronização e troca de dados (IEEE COMPUTER SOCIETY, 2010b).

A RTI é a implementação dos serviços iniciados pelo federado (*Federate Initiated Services*). A RTI não foi implementada como parte da pesquisa. Nos experimentos realizados para validação da abordagem foi utilizada uma HLA RTI de código aberto denominada CERTI (NOULARD; ROUSSELOT; SIRON, 2009).

4.2.5. Embaixador da RTI

A comunicação de um federado para a RTI é estabelecida via Embaixador da RTI (*RTI Ambassador)*. Cada federado deve solicitar os serviços disponíveis na RTI via Embaixador da RTI. Serviços iniciados pelos federados na RTI são denominados (*Federate Initiated Services*). São exemplos de serviços iniciados pelo federado: *publish, subscribe, register, update*, etc.

O Embaixador da RTI é parte atrelada à estrutura funcional da RTI. Por este motivo, assim como a RTI, o Embaixador da RTI não foi implementado durante o desenvolvimento da abordagem PowerHLA.

4.2.6. Embaixador do Federado Monitor

A comunicação do Federado Monitor para a RTI é estabelecida via Embaixador do Federado Monitor. A federação possui uma única instância do Embaixador do Federado Monitor, visto que o Federado Monitor é único dentro da federação. O Embaixador do Federado Monitor é responsável pela recepção dos dados de aplicação de outros modelos, assim como por receber respostas às solicitações de serviços feitos à RTI, via chamadas de retorno (*callback*).

O Embaixador do Federado Monitor constituiu parte do código desenvolvido para a criação da abordagem PowerHLA.

4.2.7. Monitor de Energia

O Monitor de Energia é único na federação. O conjunto composto pelo Monitor de Energia e Embaixador do Federado Monitor será denominado Federado Monitor. O Federado Monitor recebe e organiza dados de estimação do consumo de energia enviados por todos os Federados de Transmissão que compõem a federação, via Embaixador do Federado Monitor, fornecendo uma visão unificada desses dados.

Além dos dados da estimação do consumo de energia, o Federado Monitor também recebe dados de aplicação enviados por todos os federados. Os dados de aplicação são recebidos pelo federado Monitor apenas para fins de exibição dos dados trocados entre os demais federados.

Visto que é um federado, o federado Monitor também realiza solicitações de serviços à RTI (e.g., *Create Federation Execution*, *Time*

Advance Request, etc). Ele é responsável pela criação da federação e pela espera do sinal para começar as simulações quando todos os Federados de Transmissão entram na federação.

Durante a simulação, arquivos CSV (*comma separated values*) são gerados pelo Federado Monitor. Arquivos CSV são as saídas geradas pela abordagem. Para cada Federado Transmissor, o Federado Monitor gera um arquivo CSV, o qual contém a energia acumulada de cada um dos componentes monitorados por esse federado durante cada intervalo de tempo, pré-definido pelo usuário da abordagem, na linha de tempo simulado. Gráficos podem ser facilmente criados por meio de ferramentas como o Microsoft Excel ou Openoffice Calc utilizando os arquivos CSV gerados.

4.2.8. Interface PowerHLA

Este elemento disponibiliza todos os métodos necessários para coletar dados de estimação de consumo de energia durante um intervalo de tempo no decorrer da simulação. Estes métodos são chamadas aos métodos do Federado Transmissor.

Como já mencionado, os métodos para troca de dados e sincronização entre os modelos simulados devem ser codificados pelo usuário da abordagem PowerHLA, visto que, esses métodos são dependentes de cada aplicação. Consequentemente, disponibilização dos mesmos também deve ser codificada na Interface PowerHLA pelo usuário da abordagem.

Além da disponibilização dos métodos para troca de dados de aplicação, envio de dados do consumo de energia e sincronização, a Interface PowerHLA também é responsável por criar seu respectivo Federado Transmissor e lidar com aspectos de inicialização da HLA.

A Interface PowerHLA foi implementada de acordo com o padrão de projeto *singleton*. O *singleton* garante que uma classe tenha apenas uma única instância, fornecendo um ponto de acesso global para a instância. Este padrão de projeto foi utilizado a fim de prover uma maneira fácil de chamar métodos da Interface PowerHLA de qualquer lugar do código, permitindo que as propriedades acessados sejam sempre as mesmas.

A fim de permitir à abordagem simular um conjunto de modelos SystemVerilog/Verilog, como também, simular modelos C++/SystemC e SystemVerilog/Verilog juntos, duas classes extras foram criadas, uma C++ e outra SystemVerilog. Essas classes habilitam a chamada de métodos da Interface PowerHLA pelos modelos SystemVerilog/Verilog via DPI (*Direct Programming Interface*).

A maneira como a Interface PowerHLA foi implementada facilita sua extensão para trabalhar com outras linguagens e simuladores. Para cada Federado Transmissor da federação existe uma respectiva Interface PowerHLA.

4.2.9. Outros Detalhes Relevantes

A abordagem desenvolvida possui um arquivo na parte do código do Transmissor e do Monitor, denominado *defines.h*, no qual o projetista deve definir um conjunto de constantes, a fim de configurar o ambiente de simulação.

O ambiente de simulação compreende uma parte para lidar com energia e outra parte para lidar com HLA. A constante do Federado Transmissor para lidar com energia é apresentada a seguir.

- NUMBER_OF_MONITORED_COMPONENTS: O número total de componentes monitorados (pode ser um módulo, um conjunto de módulos, um SoC, etc.). Deve ser um valor inteiro positivo.

Para lidar com HLA, os Federados Transmissores possuem três constantes, descritas a seguir.

- SENDER_FEDERATE_NAME: O nome do Federado Transmissor. Este nome deve ser único dentro da federação.
- SENDER_FEDERATE_LOOKAHEAD: O *lookahead* do federado. Deve ser um valor real positivo. O *lookahead* define um "contrato", no qual o Federado Transmissor garante que não irá enviar nenhuma mensagem TSO com *timestamp* menor do que o valor do seu tempo lógico somado ao valor do seu *lookahead*. O valor do *lookahead* é utilizado para restringir o quanto um federado de tempo restringido pode avançar no tempo. O valor do *lookahead* vai depender de como se deseja realizar a sincronização dos federados. Por exemplo, supondo dois federados transmissores F1 e F2, ambos sendo

reguladores de tempo e de tempo restringindo. Supondo que ambos assinam (recebem) e publicam (estão aptos para enviar) um atributo A1 e que ambos solicitam avanços de tempo de tamanho igual aos seus *lookahead*. Supondo que o federado F1 possui *lookahead* de valor 10 e o federado F2 possui *lookahead* de valor 1, seus *timesteps* serão respectivamente 10 e 1. Isto significa que, considerando ambos os federados no mesmo tempo lógico da federação, o federado F2 deverá avançar 10 vezes no tempo, para que o federado F1 consiga avançar uma única vez. Ver IEEE Computer Society (2010b) e Schollii (2008) para detalhes.

- SENDER_FEDERATE_TIME_STEP: O *timestep* do federado. O *timestep* define o tamanho de cada avanço de tempo solicitado pelo federado. Deve ser um valor real positivo. Assim como o valor do SENDER_FEDERATE_LOOKAHEAD, o valor do *timestep* vai depender de como se deseja realizar a sincronização dos federados. Ver IEEE Computer Society (2010b) e Schollii (2008) para detalhes.

O Federado Monitor possui apenas duas constantes para lidar com HLA: a SENDER_FEDERATE_NAME e a SENDER_FEDERATE_TIME_STEP. Como mencionado anteriormente, o Federado Monitor é apenas restrito ao tempo, ou seja, ele não possui um *lookahead*.

4.3. Sincronização dos Federados

Para a sincronização dos federados são utilizados os serviços de gerenciamento de tempo (*time management services*). O PowerHLA permite a liberdade de escolha entre os dois mecanismos de avanço no tempo para sincronizar as simulações: o avanço do tempo em sequência de *timesteps* de igual tamanho, como também o avanço do tempo do ponto onde ocorre um determinado evento para o tempo do próximo evento. A escolha do mecanismo de avanço no tempo vai depender de cada aplicação. Consequentemente, para alcançar a sincronização dos federados, o usuário da abordagem PowerHLA deverá codificar em cada um deles a maneira como o tempo avança.

Na implementação, o Transmissor deverá ser, no mínimo, definido como um federado regulador de tempo (*time-regulating federate*), ou seja, um federado afiliado que pode enviar mensagens TSO e que restringe o

avanço de tempo de outros federados afiliados (de tempo restringido) dentro da execução da federação. Esta imposição é necessária, pois os dados de estimação do consumo de energia cada um dos federados transmissores devem ser recebidos de maneira ordenada pelo Federado Monitor, o qual foi definido como um federado de tempo restringido. Apenas federados de tempo restringido podem receber mensagens ordenadas por *timestamp* (mensagens TSO).

Para se tornar regulador de tempo, o Federado Transmissor solicita o serviço *Enable Time Regulation* na RTI (Código 3, linha 6). A RTI por sua vez, ao processar essa solicitação, lhe envia uma resposta, por meio da chamada de retorno (*callback*) *Time Regulation Enabled*† (Código 4). Toda chamada de retorno do Federado Transmissor é implementada dentro do Embaixador do Federado Transmissor.

O Código 3 é responsável por definir a política de tempo do Federado Transmissor. Como já mencionado anteriormente, foi definido que o Federado Transmissor é, no mínimo, regulador de tempo. Para tornar um federado regulador de tempo, deve ser informado um tempo local (Código 3, linha 4), que por padrão foi definido como sendo zero, e um valor de *lookahead* (Código 3, linha 5), o qual é definido dentro da classe *defines.h* e por padrão tem valor 1.

Código 3 - Trecho de código do elemento Transmissor responsável para chamada do serviço *Enable Time Regulation*.

```
1    void SenderFederate::enableTimePolicy()
2    {
3      //Enabling time regulation
4      RTIfedTime federateTime = fedamb->federateTime;
5      RTIfedTime lookahead = fedamb->federateLookahead;
6      rtiamb->enableTimeRegulation( federateTime, lookahead );
7      //Ticks until the Sender Federate gets the callback
8      while( fedamb->isRegulating == false )
9      {
10       rtiamb->tick();
11     }
12   }
```

O laço da linha 8 do Código 3 faz com que o Federado Transmissor espere a RTI processar a solicitação do serviço por meio do método *tick()*. Quando o serviço solicitado for atendido, a RTI enviará uma resposta para o Federado Transmissor, via Embaixador do Federado Transmissor, informando que a partir daquele momento o Federado Transmissor é regulador de

tempo (Código 4, linha 7) e que seu tempo local é o tempo definido pela RTI (Código 4, linha 8), ou seja, o tempo HLA atual.

Código 4 - Trecho de código do elemento Embaixador do Federado Transmissor responsável para chamada do serviço *Enable Time Regulation*.

```
1    void FedAmb::timeRegulationEnabled( const RTI::FedTime&
2    theFederateTime )
3      throw( RTI::InvalidFederationTime,
4      RTI::EnableTimeRegulationWasNotPending,
5      RTI::FederateInternalError )
6    {
7      this->isRegulating = true;
8      this->federateTime = convertTime( theFederateTime );
9    }
```

O Federado Monitor foi implementado apenas como um federado de tempo restringido (*time-constrained federate*), ou seja, um federado afiliado que pode receber mensagens TSO, para o qual o avanço de tempo é restringido por outros federados afiliados (reguladores de tempo) dentro da execução da federação. No caso, o Federado Monitor terá seu tempo restringido pelos federados transmissores.

Para se tornar de tempo restringido, o Federado Monitor solicita o serviço *Enable Time Constrained* na RTI (Código 5, linha 4). A RTI por sua vez envia uma resposta a essa solicitação, a *callback* (chamada de retorno) *Time Constrained Enabled*^{t} (Código 6).

Código 5 – Trecho de código do elemento Federado Monitor responsável para chamada do serviço *Enable Time Constrained*.

```
1    void MonitorFederate::enableTimePolicy()
2    {
3      //Enabling time constrained
4      rtiamb->enableTimeConstrained();
5      //Ticks until the Monitor Federate gets the callback
6      while( fedamb->isConstrained == false )
7      {
8        rtiamb->tick();
9      }
13   }
```

De maneira análoga ao código de definição de política de tempo do Federado Transmissor, o laço da linha 6 do Código 5 faz com que o Federado Monitor espere a RTI processar a solicitação do serviço por meio do método *tick()*. Quando o serviço solicitado for atendido, a RTI enviará uma resposta para o Federado Monitor, via Embaixador do Federado Monitor, informando que a partir daquele momento o Federado Monitor é de tempo restringido

(Código 6, linha 7) e que seu tempo local é o tempo definido pela RTI (Código 6, linha 8).

Código 6 – Trecho de código do elemento Embaixador do Federado Monitor responsável pela chamada de retorno *Time Constrained Enabled*† do elemento Federado Monitor.

```
1    void MonitorFedAmb::timeConstrainedEnabled( const RTI::FedTime&
2    theFederateTime )
3    throw( RTI::InvalidFederationTime,
4    RTI::EnableTimeConstrainedWasNotPending,
5    RTI::FederateInternalError)
6    {
7      this->isConstrained = true;
8      this->federateTime = convertTime( theFederateTime );
9    }
```

Por ser apenas um federado de tempo restringido, o Federado Monitor não possui um *lookahead*. Porém, todos os federados (transmissor e monitor) possuem um *timestep*. O *timestep* define o tamanho de cada avanço de tempo solicitado pelo federado.

O tempo para o qual o federado deve avançar é definido como seu tempo atual somado ao valor do *timestep* (Código 7, linha 5). Conforme já mencionado, a maneira como cada um dos federados transmissores avançará no tempo será dependente de cada aplicação. O usuário da abordagem PowerHLA deverá definir o tamanho do *timestep* e o tamanho do *lookahead* de cada um desses federados, e o instante em que cada federado deverá avançar no tempo, visando sincronizá-los. Além disso, o usuário da abordagem também deverá definir os métodos necessários para o envio dos dados de aplicação, os quais serão enviados via chamada do serviço *Update Attribute Values* da RTI.

Por exemplo, caso o requisito de um projeto seja que, para um dado um conjunto de modelos, estes devam apenas enviar dados de estimação do consumo de energia, ou seja, não devam trocar dados entre si. Visto que os federados transmissores do cenário descrito não precisam estar sincronizados entre si, o tamanho do *timestep* de cada um deles poderia ser arbitrário, assim como o valor do seu *lookahead*. Os federados transmissores desses modelos poderiam, por exemplo, solicitar um avanço no tempo (de qualquer tamanho) toda vez que um dado de estimação fosse enviado ao Federado Monitor.

O Federado Monitor também poderia solicitar um avanço de tempo de qualquer tamanho, porém, com a ressalva que o avanço de tempo só seria concedido caso o avanço de tempo não ultrapasse um ponto no qual ele ainda poderia receber mensagens TSO, visto que o Federado Monitor é de tempo restringido.

Na abordagem PowerHLA, o Federado Monitor funciona como um consumidor de dados. A simulação termina quando não existem mais dados dos federados transmissores a serem recebidos.

Código 7 – Trecho de código para avanço de tempo (Transmissor e Monitor).

```
1    void SenderFederate::advanceTime( double timestep )
2    {
3       // requesting the advance
4       fedamb->isAdvancing = true;
5       RTIfedTime newTime = (fedamb->federateTime + timestep);
6       rtiamb->timeAdvanceRequest( newTime );
7       // wait for the time advance to be granted
8       while( fedamb->isAdvancing )
9       {
10         rtiamb->tick();
11      }
12   }
```

O valor do *timestep* do federado deve ser definido dentro do seu respectivo arquivo *defines.h*. O *timestep* dos federados (transmissores e monitor), assim como o *lookahead* dos federados transmissores podem ter diferentes valores entre si, porém vale salientar que essa diferença terá efeito no tempo de simulação e no uso do *buffer* da RTI. Esse *buffer* é utilizado para enfileirar mensagens a serem entregues aos federados, de acordo com aspectos de sincronização.

Na abordagem desenvolvida, o Federado Monitor deve ser o primeiro a ser executado, visto que ele é responsável pela criação da federação e pela espera de um sinal para começar as simulações quando todos os federados de transmissão entram na federação.

4.4. Publicação e Assinatura de Atributos

Na abordagem desenvolvida, uma diferença relevante entre federados transmissores e o Federado Monitor refere-se à publicação e assinatura de atributos. Os serviços utilizados para publicação e assinatura de atributos são denominados serviços de gerenciamento de declaração (*declaration management services*).

Federados transmissores devem informar a intenção de publicar os atributos da classe de objeto antes de registrá-lo. A intenção de publicar é feita pela chamada do serviço *Publish Object Class Attributes* na RTI (Código 8, linha 17). Os atributos necessariamente publicados pelos federados de transmissão são: o nome do federado, o nome do componente monitorado, o valor de energia atual acumulado por esse componente, e o tempo simulado atual do componente monitorado. Esses atributos se referem, respectivamente, às linhas 11, 12, 13 e 14 do Código 8. Estes atributos precisam ser definidos dentro do arquivo .fed (Código 9, linhas 15, 16, 17 e 18, respectivamente). Eles são definidos dentro da classe de objeto *PowerObject* (Código 9, linha 14).

Como já mencionado, o arquivo .fed contém informações derivadas do FOM (classes, atributos, parâmetros, nomes, etc.) utilizadas pelo RTI durante o tempo de execução.

Os dados trocados entre os federados transmissores, assim como os dados recebidos pelo Federado Monitor, também precisam ser definidos como atributos dentro do arquivo .fed. Esses atributos deverão ser definidos pelo usuário da abordagem PowerHLA, de acordo com cada aplicação. Os atributos poderão ser definidos dentro da classe PowerObject, ou poderão ser definidos em uma nova classe. Cada atributo representa um dado que pode ser recebido ou manipulado por algum dos federados.

Código 8 – Trecho de código do elemento Transmissor responsável pela chamada do serviço *Publish Object Class Attributes*.

```
1    void SenderFederate::publishAndSubscribe()
2    {
3      //Publishing all attributes from ObjectRoot.powerObject
4      //Before register some instance of the object class
5      ObjectRoot.powerObject and
6      //update their attributes values, its need to tell to the RTI
7      the intention to publish the information
8      //Adding the information into a handle set
9      RTI::AttributeHandleSet *attributes =
10     RTI::AttributeHandleSetFactory::create( 4 );
11     attributes->add( this->federateNameHandle );
12     attributes->add( this->componentNameHandle );
13     attributes->add( this->energyHandle );
14     attributes->add( this->currentSimulatedTimeHandle );
15     //Makes the current publication of the information added into
16     the handle set
17     rtiamb->publishObjectClass( this->powerObjectHandle,
18     *attributes );
19     delete attributes;
20   }
```

Código 9 – Código do arquivo .fed representando a informação trocada entre os federados.

```
1    (FED
2      (Federation Portico-Test)
3      (FEDversion v1.3)
4      (spaces
5      (space TestSpace
6       (dimension TestDimension))
7      (space OtherSpace
8       (dimension OtherDimension))
9      )
10     (objects
11       (class ObjectRoot
12       (attribute privilegeToDelete reliable timestamp)
13       (class RTIprivate)
14       (class PowerObject
15       (attribute federateName reliable timestamp TestSpace)
16       (attribute componentName reliable timestamp TestSpace)
17       (attribute energy reliable timestamp TestSpace)
18       (attribute currentSimulatedTime reliable timestamp
19       TestSpace)
20       )))
21   )
```

Quando o valor de algum atributo for atualizado por um transmissor (via serviço *Update Attribute Values*) (Código 10, linha 8), em algum momento do tempo o Federado Monitor receberá uma chamada de retorno relativa a essa atualização. Esta chamada de retorno, denominada *Reflect Attribute Values*[†](Código 11), é implementada no Embaixador do Federado Monitor e tem como objetivo lhe repassar os valores atualizados. Porém, para que o Federado Monitor possa receber os valores de atributos enviados pelos federados de transmissão ele deve assinar previamente tais atributos da classe de objeto (*PowerObject*).

A assinatura é feita por meio da chamada do serviço *Subscribe Object Class Attributes* (Código 12). Os atributos assinados (*subscribed*) pelo Federado Monitor são os mesmos publicados pelos federados transmissores: o nome do federado, o nome do componente monitorado, o valor de energia atual acumulado por esse componente, e o tempo simulado atual do componente monitorado (Código 12, linhas 6 a 9).

Quando valores de atributos de instância de objetos, que correspondem aos atributos de classe especificados, forem descobertos o Federado Monitor receberá esses valores via RTI, por meio do já mencionado método de retorno *Reflect Attribute Values*[†](Código 11). Esse método possui uma iteração sobre o conjunto de atributos para que possam

ser processados e, posteriormente, sejam geradas as saídas CSV (Código 11, linhas 18 a 24). Os atributos de instância de objetos são descobertos pela chamada de retorno *Discover Object Instance*[†](Código 13).

De maneira similar, caso o usuário da abordagem PowerHLA deseje imprimir no Federado Monitor os dados de aplicação trocados entre os federados transmissores, o método apresentado no Código 11 deverá ser adequado para este fim.

Código 10 – Código do método para envio de dados do federado transmissor.

```
1    void SenderFederate::sendData(char* federateName, char*
2    componentName, double energy, int currentSimulatedTime){
3      /* For each calling of sendData the Sender federate will update
4      the attribute values of the registered object: federateName,
5      componentName, energy and currentSimulatedTime
6      */
7      //Updating the attribute values
8      updateAttributeValues(myObjectHandle, federateName,
9      componentName, energy, currentSimulatedTime);
10     //Requesting a time advance and wait until get it
11     advanceTime(SENDER_FEDERATE_TIME_STEP);
12   }
```

Código 11 – Trecho de código do elemento Embaixador do Federado Monitor responsável pela chamada de retorno *Reflect Attribute Values*[†] do elemento Federado Monitor.

```
1    void MonitorFedAmb::reflectAttributeValues( RTI::ObjectHandle
2    theObject, const RTI::AttributeHandleValuePairSet& theAttributes,
3    const RTI::FedTime& theTime, const char *theTag,
4    RTI::EventRetractionHandle theHandle ) throw(
5    RTI::ObjectNotKnown, RTI::AttributeNotKnown,
6    RTI::FederateOwnsAttributes, RTI::InvalidFederationTime,
7    RTI::FederateInternalError )
8    {
9    cout << "Reflection Received:";
10   // print the handle
11   cout << " object=" << theObject;
12   // print the tag
13   cout << ", tag=" << theTag;
14   // print the time
15   cout << ", time=" << convertTime( theTime );
16   // print the attribute information
17   cout << ", attributeCount=" << theAttributes.size() << endl;
18     for( RTI::ULong i = 0; i < theAttributes.size(); i++ )
19     {
20       RTI::ULong length = theAttributes.getValueLength(i);
21       char *value = theAttributes.getValuePointer(i,length);
22       //Processing each attribute according with its index
23       managerPowerLogger->processAttribute(value, i);
24     }
25   }
```

Código 12 – Trecho de código do elemento Federado Monitor responsável pela chamada do serviço *Subscribe Object Class Attributes*.

```
1    void MonitorFederate::publishAndSubscribe()
2    {
3      // package the information into a handle set
4      RTI::AttributeHandleSet *attributes =
5      RTI::AttributeHandleSetFactory::create( 4 );
6      attributes->add( this->federateNameHandle );
7      attributes->add( this->componentNameHandle );
8      attributes->add( this->energyHandle );
9      attributes->add( this->currentSimulatedTimeHandle );
10
11     // subscribe to all attributes of ObjectRoot.PowerObject
12     /* subscribing the above attributes we are able to hear about
13     this information when it is created and/or altered in other
14     federates */
15
16     rtiamb->subscribeObjectClassAttributes( this-
17     >powerObjectHandle, *attributes );
18     // clean up
19     delete attributes;
20   }
```

Código 13 – Trecho de código do elemento Embaixador do Federado Monitor responsável pela chamada de retorno *Discover Object Instance*[†] do elemento Federado Monitor.

```
1    void MonitorFedAmb::discoverObjectInstance( RTI::ObjectHandle
2    theObject, RTI::ObjectClassHandle theObjectClass, const char*
3    theObjectName )
4    throw(RTI::CouldNotDiscover,RTI::ObjectClassNotKnown,
5    RTI::FederateInternalError )
6    {
7      cout << "Discoverd Object: handle=" << theObject << ",
8      classHandle=" << theObjectClass << ", name=" << theObjectName
9      << endl;
10   }
```

4.5. O Uso de DPI

O uso de DPI com HLA na abordagem desenvolvida permite que modelos C++, SystemC, SystemVerilog e Verilog possam enviar dados de aplicação e dados de consumo de energia, para o Federado Monitor em intervalos de tempo pré-definidos. O código relativo ao envio de dados de aplicação da aplicação deverá ser escrito pelo usuário da abordagem PowerHLA.

Como já mencionado, a implementação da RTI utilizada no decorrer da pesquisa foi a RTI C++ de código aberto, denominada CERTI 3.4.0 (NOULARD; ROUSSELOT; SIRON, 2009). Um dos desafios foi encontrar uma maneira a partir da qual simulações Verilog e SystemVerilog pudessem

enviar informações por meio do CERTI. Para tal, foi adotado o uso de DPI (*Direct Programming Interface*).

DPI pode ser definida como uma interface entre SystemVerilog e a linguagem C (DOULOS, 2015). Uma classe C++ (Código 14) e uma classe SystemVerilog (Código 15) foram criadas para que alguns métodos da Interface PowerHLA (Código 16) pudessem ser chamados dentro do código SystemVerilog/Verilog via DPI.

O método *initializePowerHLA* (Código 14, linha 8) é responsável por chamar o método *initializePowerHLA* da Interface PowerHLA (Código 16, linha 25). Este método tem como objetivo executar os aspectos de inicialização do Federado Transmissor. Similarmente, os métodos *accumulateEnergy* (Código 14, linha 12) e *sendData* (Código 14, linha 18) possuem métodos correspondentes na Interface PowerHLA (Código 16, linhas 84 e 132, respectivamente).

Todos os métodos declarados dentro do Código 14 possuem a diretiva "*extern C*". Essa diretiva permite que essas funções tenham sua visibilidade entendida para todo o código, ou seja, no contexto da pesquisa, ela permitirá que as funções DPI declaradas na classe SystemVerilog (Código 15) possam acessar as funções declaradas na classe C++ (Código 14), que por sua vez acessam funções da Interface PowerHLA (Código 16).

Com o objetivo de garantir que o *singleton* (classe Interface PowerHLA, Código 16) seja *thread-safe*, a partir do momento que uma *thread* T1 chama seu método *getInstance()* a primeira vez, ele deverá ser "travado" para que não exista a possibilidade de que outras *threads* possam tomar vir a chamar o método *getInstance()* antes que a *thread* T1 retorne deste método, ou seja, antes que o singleton seja de fato instanciado. Isso garantirá que o *singleton* tenha uma única instância dentro da aplicação.

O padrão ISO C++ 2011 (C++ STANDARDS COMMITTEE AND OTHERS, 2011) já contempla um meio simplificado para a solução do problema acima descrito. O padrão garante que variáveis estáticas sejam inicializadas de maneira *thread-safe* (Código 16, linha 18). As versões a partir da versão 4.7 do compilador g++/gcc do compilador são totalmente compatíveis com os aspectos deste padrão. A partir da versão 4.4 os

compiladores g++/gcc já garantem que variáveis estáticas sejam inicializadas de maneira *thread-safe*.

De maneira similar, para garantir que o método *accumulateEnergy* da classe Interface PowerHLA também seja *thread-safe,* é realizada uma "trava" (*lock*) dentro deste método, o qual é definido no Federado Transmissor. O "destravamento" (*unlock*) é realizado quando a execução do método é finalizada. Esta técnica pode ser utilizada para os demais métodos que, por ventura, possam ser chamados por mais de uma *thread*.

Qualquer um dos métodos da Interface PowerHLA podem ser facilmente chamados dentro do código SystemVerilog/Verilog utilizando a técnica descrita. Vale destacar, que durante a revisão bibliográfica não foi encontrada pesquisa que contemplasse chamadas DPI para viabilizar a troca de informações entre modelos SystemVerilog via implementação de uma HLA RTI, tal qual o CERTI.

Código 14 – Classe C++ que disponibiliza os métodos da Interface PowerHLA.

```
1    #include "defines.h"
2    #include "SenderFederate.h"
3    #include <iostream>
4    #include <pthread.h>
5    #include <stdlib.h>
6    #include "power_HLA_interface.h"
7
8    extern "C" void initializePowerHLA(){
9      power_HLA_interface::getInstance()->initializePowerHLA();
10   }
11
12   extern "C" void accumulateEnergy(double powerValue, double
13   currentTime, char* componentName){
14     power_HLA_interface::getInstance()-
15     >accumulateEnergy(powerValue, currentTime, componentName);
16   }
17
18   extern "C" void sendData(){
19     power_HLA_interface::getInstance()->sendDataSV();
20   }
```

Código 15 – Classe SystemVerilog que permite a chamada dos métodos da Interface PowerHLA via DPI.

```
1    import "DPI-C" function void accumulateEnergy(real powerValue,
2    real svTime, string componentName);
3
4    import "DPI-C" function void initializePowerHLA();
5
6    import "DPI-C" function void sendData();
```

Código 16 – Código da Interface PowerHLA. (continua)

```
1     #include<iostream>
2     #include "power_HLA_interface.h"
3     #include "defines.h"
4
5     power_HLA_interface::power_HLA_interface()
6     {
7     senderFederate = senderFederate = new
8     SenderFederate(SENDER_FEDERATE_NAME);
9     }
10
11    /* This function is called to create an instance of the class.
12     * Calling the constructor publicly is not allowed. The
13     * constructor
14     * is private and is only called by this Instance function.
15     */
16    power_HLA_interface& power_HLA_interface::getInstance()
17    {
18          static power_HLA_interface instance;
19          return instance;
20    }
21
22    /*
23     * Initialize the approach
24     */
25    void power_HLA_interface::initializePowerHLA(){
26
27          senderFederate->joinFederationExecution();//Should be
28          called after the Manager Federate creation
29          senderFederate-> prepareToSynchronize();//Preparing the
30          federate to synchronize
31          senderFederate->setPolicies();//Setting federate time
32          policy (constrained, regulating) and data interest policy
33          (publish, subscribe)
34
35          //Creates an array of MonitoredComponent objects from an
36          array of component monitored names defined into the
37          defines.h file
38          #ifdef MONITORED_COMPONENTS
39          char *componentNames[] = MONITORED_COMPONENTS;
40          addComponents(componentNames);
41          senderFederate->setInitialized();//After add all
42          components, sets a flag to say that the Sender Federate
43          was initialized
44          #endif
45    }
46
```

Código 16 – Código da Interface PowerHLA. (continua)

```
47    /*
48     * Creates an array of MonitoredComponent from an array of
49     * component monitored names
50     * @param componentName the array of component monitored names
51     */
52    void power_HLA_interface::addComponents(char *componentNames[]){
53          char* currComponentName;
54          for (int i = 0 ; i < NUMBER_OF_MONITORED_COMPONENTS; i++){
55                currComponentName = componentNames[i];
56                MonitoredComponent *component = new
57                MonitoredComponent(currComponentName);
58                senderFederate-> addComponent(component);
59          }
60    }
61
62    /*
63     * Accumulates a value in joules into the current energy from a
64     * specified component
65     * @param energyValue the value in joules to be accumulated
66     * @param componentName the component name
67     */
68    void power_HLA_interface::accumulateEnergy(double energyValue,
69    char *componentName){
70          senderFederate->accumulateEnergy(energyValue,
71          componentName);
72    }
73
74    /*
75     * Accumulates a value in joules into the current energy from a
76     * given power value and a given sc_time. The energy will be
77     * calculated making the difference betwen the current sc_time
78     * and the last sampled sc_time and muliplicand this result
79     * by the current power value.
80     * @param powerValue the current power value in watts
81     * @param scCurrentTime the current sc_time in seconds
82     * @param componentName the component name
83     */
84    void power_HLA_interface::accumulateEnergy(double powerValue,
85    double scCurrentTime, char *componentName){
86          senderFederate->accumulateEnergy(powerValue,
87          scCurrentTime, componentName);
88    }
89
90    /*
91     * Sets the current energy in joules for a specified
92     * component
93     * @param componentName the component name
94     * @param newValue the desired value in joules
95     */
```

Código 16 – Código da Interface PowerHLA. (conclusão)

```
96    void power_HLA_interface::setEnergy(double newValue, char
97    *componentName){
98          senderFederate->setEnergy(newValue, componentName);
99    }
100
101   /*
102    * Returns the current accumulated energy in joules from a
103    * specified component
104    * @param componentName the component name
105    * @return the current accumulated energy in joules from a
106    * specified component
107    */
108   double power_HLA_interface::getEnergy(char *componentName){
109         return senderFederate->getEnergy(componentName);
110   }
111
112   /*
113    * Returns the current power in watts
114    * @param componentName the component name
115    * @return the current power in watts
116    */
117   double power_HLA_interface::getCurrentPower(char *componentName)
118   {
119         return senderFederate->getCurrentPower(componentName);
120   }
121
122   /*
123    * Returns and returns the average power in watts
124    * @param componentName the component name
125    * @return the the average power in watts
126    */
127   double power_HLA_interface::getPowerAverage(char *componentName)
128   {
129         return senderFederate->getPowerAverage(componentName);
130   }
131
132   void power_HLA_interface::sendDataSV(){
133         senderFederate->sendDataSV();
134   }
135
136   /*
137    * Asks for time advance defined by the timestep variable.
138    * @param timestep the value of timestep
139    * @return the the average power in watts
140    */
141   void power_HLA_interface::advanceTime( double timestep ){
142         senderFederate->advanceTime( timestep );
}
```

4.6. A Plataforma MPSoCBench

MPSoCBench (DUENHA et al., 2014) é uma plataforma composta de um conjunto escalável de MPSoC útil para o desenvolvimento e avaliação de alto nível de novas ferramentas, metodologias, *software* e componentes de *hardware*. A plataforma fornece uma infraestrutura de simulação completa de código aberto incluindo componentes escaláveis de *hardware* e *software*, com fácil instrumentação e rápida simulação em diferentes níveis de abstração.

A ferramenta é um conjunto de MPSoC escalável incluindo quatro diferentes ISA (*Instruction Set Architecture*)[25] com até 64 núcleos, com diferentes dispositivos de interconexão que definem diferentes níveis de abstração de simulação. O usuário fornece parâmetros para criar simuladores MPSoC e um *script* gera o código apropriado para processadores, interconexões, *caches*, memórias, e IP. Estes componentes são compilados para criar um ou mais simuladores MPSoC que serão armazenados em diretórios específicos. O conjunto de aplicações é compilado com compiladores cruzados (*cross-complilers*)[26] apropriados e os arquivos executáveis são armazenados juntos com os simuladores. Após a execução de cada simulação, uma prova de corretude é realizada com avaliação dos arquivos de saída da aplicação.

A plataforma MPSoCBench foi utilizada para construção de modelos SystemC ESL de MPSoC, os quais serviram como parte dos estudos de caso para validação da abordagem proposta. A decisão de se utilizar essa plataforma se deu mediante os seguintes fatores:

- A documentação necessária para instalação e uso da plataforma é disponibilizada na internet (UNIVERSITY OF CAMPINAS, 2015a);

[25] O ISA representa o conjunto de operações que um processador, microprocessador, microcontrolador, CPU ou outros periféricos programáveis suporta, fornece ou disponibiliza para o programador. É a representação em mnemônicos do código de máquina, com a finalidade de facilitar o acesso ao componente (JAVVIN TECHNOLOGIES, 2007).

[26] Compilador cruzado é um compilador que é capaz de produzir código executável em uma plataforma diferente da qual o compilador está sendo executado (JAVVIN TECHNOLOGIES, 2007).

- A plataforma utiliza a abordagem de estimação de consumo de energia PowerSC (KLEIN et al., 2007a, 2007b), a qual havia sido eleita para validação da pesquisa ora apresentada.
- O tempo de resposta do contato feito com membros da equipe desenvolvedora da MPSoCBench se deu de forma rápida.
- A plataforma pode ser utilizada na etapa de exploração do espaço do projeto de MPSoC, visto que modelos MPSoC executáveis podem ser criados a partir de parâmetros fornecidos pelo usuário.

Os principais parâmetros que podem ser fornecidos para a plataforma MPSoCBench são apresentados a seguir (Figura 7).

Figura 7 – Parâmetros utilizados pela plataforma MPSoCBench.

```
-b or --build: para construir os simuladores
-r or --run: para executar os simuladores
-nb or --nobuild: para executar sem recompilar os modelos
-l or --clean: apaga arquivos de objetos e os simuladores previamente
criados.
-d or --distclean: apaga os arquivos do modelo de processador
previamente criado pelo ArchC.
-h or --help: ajuda
-p or --processor: para escolher o modelo de processadores
-pw or --power: para habilitar a estimativa de dissipação de potência
de processadores SPARC e MIPS
-p or --processor: para escolher o modelo de processadores
-n or --numcores: para escolher o número de núcleos (1,2,4,8,16,32,ou
64)
-s or --software: para escolher a aplicação
-i or --interconnection: para escolher o dispositivo de interconexão.
-c or --condor: para ativar execução no HTCondor
```

Os possíveis valores para os parâmetros são listados a seguir (Figura 8).

Figura 8 – Possíveis valores para os parâmetros da plataforma MPSoCBench.

```
processor: powerpc, mips, sparc, or arm
n_cores: 1,2,4,8,16,32, or 64
device: router.lt, noc.lt, noc.at
software: basicmath, dijkstra, fft, lu, sha, stringsearch,
susancorners, susanedges, susansmoothing, water, water-spatial, multi-
8, multi-16, multi-office-telecomm, multi-network-automotive, multi-
security.
```

Um tutorial completo para uso da plataforma MPSoCBench pode ser visto na página web do projeto (UNIVERSITY OF CAMPINAS, 2015a).

4.6.1. Modelos de Energia

A plataforma MPSoCBench inclui um modelo de dissipação de potência para processadores MIPS e SPARC utilizando PowerSC (KLEIN et al., 2007a, 2007b) e acPower (MA; AZEVEDO, 2009). O acPower é uma ferramenta de estimação de consumo de energia que gera relatórios de potência de *softwares* executados sobre o simulador ArchC[27] (UNIVERSITY OF CAMPINAS, 2015b).

O AcPower utiliza o método de caracterização descrito por Tiwari, Malik e Wolfe (1994) dentro do ambiente ArchC. O processo de extração das características de dissipação de potência dos processadores MIPS e SPARC utilizados na plataforma MPSoCBench é descrito no trabalho de Guedes et al. (2013).

Os modelos de consumo de energia dos processadores SPARC e MIPS foram construídos a partir do processo de caracterização utilizando modelos RTL de código aberto de processadores LEON3 e PLASMA, respectivamente. Os modelos de consumo de energia são baseados em diferentes tecnologias de FPGA e ASIC. Um dos principais usos é a avaliação de dissipação de potência durante a execução de uma aplicação.

O processo de caracterização é realizado uma única vez para cada processador e requer uma ferramenta de síntese e simulação RTL, tal qual Synopsys Design Compiler, Xilinx ISE ou Altera Quartus. Após a síntese, é realizada a simulação da versão *back-annotated*[28] do processador, coleta de atividades de chaveamento e o uso de um fluxo de estimação de consumo de energia, tal qual Xilinx XPower, Altera PowerPlay, ou Synopsys Power Compiler.

A média da potência dissipada por instrução é coletada e uma tabela de potências dissipadas é criada para cada configuração de *hardware* (tecnologia e frequência de operação). O usuário pode escolher entre diferentes tecnologias de caracterização e frequência para o MIPS (XILINX SPARTAN-3 1200e XC3S1200 100MHz, XILINX SPARTAN-6 XC6SLX75

[27] ArchC é uma linguagem de descrição de arquitetura de código aberto desenvolvida pela Universidade de Campinas.

[28] *Back-annotation*, para FPGAs, significa capturar os atrasos (às vezes os números dos pinos também) do layout (após a fase de *place & route*) e colocá-los na descrição (esquemático ou HDL), permitindo uma simulação mais realista (EDABOARD, 2009).

100MHz, Altera CycloneV 25MHz ou 100 MHz) e SPARC (XILINX SPARTAN-3 1000 40MHz ou XILINX SPARTAN - 6 XC6SLX75 40MHz).

A plataforma MPSoCBench integrada com PowerSC e o acPower faz a leitura das tabelas acima mencionadas e gera as características de dissipação de potência no modelo de plataforma virtual. Após a simulação, MPSoCBench gera um relatório com a dissipação de potência de cada núcleo (utilizando relatórios do acPower) e também armazena os detalhes da estimação em arquivos.

4.6.1.1. Tabelas de Caracterização para MIPS e SPARC

A plataforma MPSoCBench oferece um conjunto de tabelas no formato CSV, que representam o conjunto de instruções caracterizadas em termos de energia para processadores MIPS e SPARC. O usuário da plataforma pode escolher dentre diferentes tecnologias de caracterização e frequência para o MIPS (XILINX SPARTAN-3 1200e XC3S1200 100MHz, XILINX SPARTAN-6 XC6SLX75 100MHz, Altera CycloneV 25MHz or 100 MHz) e SPARC (XILINX SPARTAN-3 1000 40MHz or XILINX SPARTAN- 6 XC6SLX75 40MHz).

Os estudos de caso criados pela plataforma MPSoCBench, apresentados no Capítulo 5, foram executados utilizando as tabelas padrões definidas dentro do código fonte da plataforma. Para o processador MIPS, a tabela utilizada se refere à FPGA Altera CycloneV 100 MHz (Código 17).

Código 17 – Tabela para caracterização para processador MIPS referente à FPGA Altera CycloneV 100 MHz. (continua)

```
1    # Number of profiles
2    1
3    # ProfileID, Characterization Frequency,Frequency Scale,EPI
4    Scale,Characterization Name,Characterization Description
5    0,100,1E6,1E-9,"mips-plasma-xc6slx75","Plasma Characterization
6    xc6slx75 at 100MHz"
7    # Stall consumption
8    0
9    1,lb,5.08
10   2,lbu,3.95
11   3,lh,5.82
12   4,lhu,3.61
13   5,lw,5.1
14   6,lwl,5.1
15   7,lwr,5.1
16   8,sb,6.3
```

Código 17 – Tabela para caracterização para processador MIPS referente à FPGA Altera CycloneV 100 MHz. (conclusão)

```
17   9,sh,6.53
18   10,sw,6.64
19   11,swl,6.64
20   12,swr,6.64
21   13,addi,1.24
22   14,addiu,1.24
23   15,slti,0.78
24   16,sltiu,0.74
25   17,andi,0.66
26   18,ori,0.67
27   19,xori,0.93
28   20,lui,0.65
29   21,add,1.24
30   22,addu,1.2
31   23,sub,1.24
32   24,subu,1.2
33   25,slt,0.56
34   26,sltu,0.56
35   27,instr_and,0.56
36   28,instr_or,0.58
37   29,instr_xor,1.18
38   30,instr_nor,1.05
39   31,nop,0.44
40   32,sll,0.56
41   33,srl,0.56
42   34,sra,0.61
43   35,sllv,0.56
44   36,srlv,0.56
45   37,srav,0.6
46   38,mult,1.01
47   39,multu,1
48   40,div,2.52
49   41,divu,2.61
50   42,mfhi,0.48
51   43,mthi,0.73
52   44,mflo,0.48
53   45,mtlo,0.69
54   46,j,2.85
55   47,jal,2.99
56   48,jr,3.67
57   49,jalr,3.44
58   50,beq,2.32
59   51,bne,2.32
60   52,blez,2.32
61   53,bgtz,2.32
62   54,bltz,2.32
63   55,bgez,2.32
64   6,bltzal,2.32
65   57,bgezal,2.32
66   58,sys_call,0
67   59,instr_break,0
```

Para o processador SPARC foi utilizada a tabela XILINX SPARTAN-3 1000 40MHz (Código 18). Cada linha de código representa uma linha na tabela. Cada vírgula representa a separação de valores em colunas. A primeira coluna representa o índice da instrução, a segunda coluna o

opcode (código de operação) da instrução e a terceira coluna o valor de energia atribuído a cada *opcode*.

Código 18 – Tabela para caracterização para processador SPARC referente à FPGA XILINX SPARTAN-3 1000 40MHz. (continua)

```
1    # Number of profiles,,,,,
2    1,,,,,
3    # ProfileID, Characterization Frequency,Frequency Scale,EPI
4    Scale,Characterization Name,Characterization Description
5    0,40,1.00E+06,1.00E-12,Leon3 at 40MHz,"Using Xilinx tools,
6    Spartan3 at 40MHz"
7    # Stall consumption,,,,,
8    0,,,,,
9    1,add_imm,540.25,,,
10   2,add_reg,624.00,,,
11   3,addcc_imm,502.50,,,
12   4,addcc_reg,625.25,,,
13   5,addx_imm,505.25,,,
14   6,addx_reg,624.00,,,
15   7,addxcc_imm,503.50,,,
16   8,addxcc_reg,626.25,,,
17   9,and_imm,386.75,,,
18   10,and_reg,370.25,,,
19   11,andcc_imm,390.25,,,
20   12,andcc_reg,372.75,,,
21   13,andn_imm,419.75,,,
22   14,andn_reg,373.75,,,
23   15,andncc_imm,422.25,,,
24   16,andncc_reg,379.75,,,
25   17,ba,283.50,,,
26   18,bcc,288.00,,,
27   19,bcs,348.00,,,
28   20,be,283.25,,,
29   21,bg,348.00,,,
30   22,bge,288.00,,,
31   23,bgu,348.00,,,
32   24,bl,348.00,,,
33   25,ble,283.25,,,
34   26,bleu,283.00,,,
35   27,bn,345.75,,,
36   28,bne,346.25,,,
37   29,bneg,347.75,,,
38   30,bpos,288.25,,,
39   31,bvc,288.25,,,
40   32,bvs,348.00,,,
41   33,call,285.00,,,
42   34,jmpl_imm,283.50,,,
43   35,jmpl_reg,283.50,,,
44   36,ld_imm,295.50,,,
45   37,ld_reg,295.50,,,
46   38,ldd_imm,314.75,,,
47   39,ldd_reg,378.25,,,
48   40,ldsb_imm,304.50,,,
49   41,ldsb_reg,304.50,,,
50   42,ldsh_imm,303.00,,,
51   43,ldsh_reg,303.00,,,
52   44,ldstub_imm,343.25,,,
53   45,ldstub_reg,343.25,,,
```

Código 18 – Tabela para caracterização para processador SPARC referente à FPGA XILINX SPARTAN-3 1000 40MHz. (conclusão)

```
54   46,ldub_imm,295.25,,,
55   47,ldub_reg,295.25,,,
56   48,lduh_imm,295.75,,,
57   49,lduh_reg,295.75,,,
58   50,mulscc_imm,469.00,,,
59   51,mulscc_reg,605.50,,,
60   52,nop,345.75,,,
61   53,or_imm,429.75,,,
62   54,or_reg,386.75,,,
63   55,orcc_imm,433.25,,,
64   56,orcc_reg,388.25,,,
65   57,orn_imm,425.00,,,
66   58,orn_reg,395.00,,,
67   59,orncc_imm,426.00,,,
68   60,orncc_reg,395.00,,,
69   61,rdy,356.50,,,
70   62,restore_imm,379.75,,,
71   63,restore_reg,380.50,,,
72   64,save_imm,381.75,,,
73   65,save_reg,383.25,,,
74   66,sdiv_imm,355.75,,,
75   67,sdiv_reg,295.00,,,
76   68,sdivcc_imm,354.25,,,
77   69,sdivcc_reg,296.50,,,
78   70,sethi,450.25,,,
79   71,sll_imm,392.75,,,
80   72,sll_reg,383.50,,,
81   73,smul_imm,355.75,,,
82   74,smul_reg,295.00,,,
83   75,smulcc_imm,354.25,,,
84   76,smulcc_reg,296.50,,,
85   77,sra_imm,390.50,,,
86   78,sra_reg,372.50,,,
87   79,srl_imm,392.00,,,
88   80,srl_reg,372.75,,,
89   81,st_imm,433.25,,,
90   82,st_reg,433.25,,,
91   83,stb_imm,396.50,,,
92   84,stb_reg,386.50,,,
93   85,std_imm,412.75,,,
94   86,std_reg,412.75,,,
95   87,sth_imm,458.50,,,
96   88,sth_reg,458.50,,,
97   89,sub_imm,507.25,,,
98   90,sub_reg,631.50,,,
99   91,subcc_imm,513.75,,,
100  92,subcc_reg,634.25,,,
101  93,subx_imm,505.75,,,
102  94,subx_reg,629.00,,,
103  95,subxcc_imm,511.00,,,
104  96,subxcc_reg,631.75,,,
105  97,swap_imm,327.50,,,
106  98,swap_reg,327.50,,,
107  99,trap_imm,0.00,,,
108  100,trap_reg,0.00,,,
109  101,udiv_imm,325.25,,,
...
```

4.7. O Arcabouço Ptolemy

Ptolemy (BERKELEY EECS, 2015b) é um arcabouço de código aberto que suporta experimentos com projetos orientados a atores (*actor*). Atores são componentes de *software* que executam concorrentemente e se comunicam por meio de mensagens enviadas via portas (*port*) interconectadas.

No Ptolemy, um modelo é representado por uma interconexão de atores que podem interagir entre si. A Semântica de um modelo no Ptolemy não é determinado pelo arcabouço, mas por um componente de *software* denominado diretor (*director*), o qual implementa um modelo de computação (MoC).

Um modelo de computação (MoC) é uma definição formal do conjunto de operações permitidas usadas em uma computação e seus respectivos custos. Isso define o custo de um sistema em certo nível de abstração para refletir as características essenciais do mesmo (CHEN; DOEMER; CENTER, 2009).

Um modelo de computação governa a semântica da interação entre os atores, impondo uma disciplina de tempo de execução. Estes modelos de computação também são conhecidos como domínios.

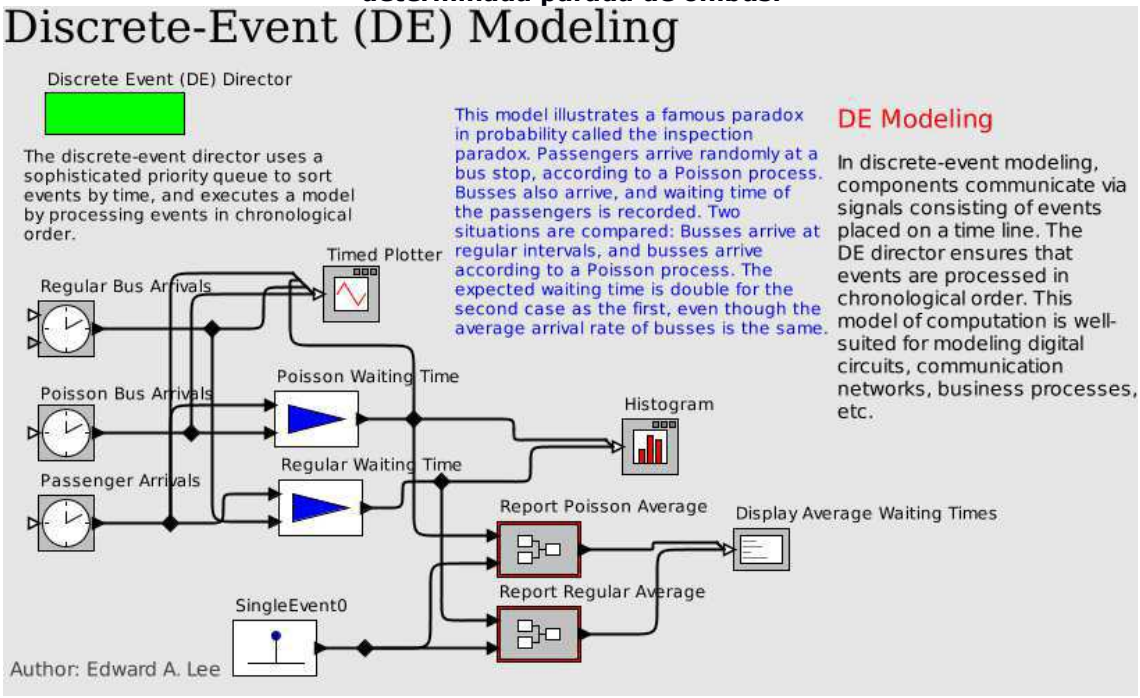
O Ptolemy possui diretores para diversos domínios: fluxo de dados, evento discreto, tempo contínuo, entre outros. O foco do trabalho de Costa (2015) está no âmbito de domínio discreto (*Discrete-Event domain* ou simplesmente *DE domain*).

O domínio discreto fornece um ambiente geral para simulações orientadas por tempo de sistemas tais como redes de comunicação, e sistemas de *hardware*. Neste domínio os atores se comunicam enviando fichas (tokens) através de suas conexões (BERKELEY EECS, 2015b). A ficha enviada e o tempo na qual foi enviada constituem um evento no domínio de evento discreto (MULIADI, 1999).

Ao receber um evento, o ator de destino é ativado e reage a este evento. Essa reação pode modificar o estado interno do ator e possivelmente gerar novos eventos, resultando em futuras reações. O escalonador do domínio DE garante que eventos sejam processados em ordem cronológica.

Muliadi (1999) cita como um exemplo de modelo Ptolemy, a chegada de ônibus e passageiros em uma determinada parada de ônibus como sendo eventos discretos. Ele afirma que construindo um modelo com os atores apropriados, é possível calcular diversas propriedades estatísticas, por exemplo, a distribuição do tempo de espera dos passageiros. O modelo descrito é apresentado na Figura 9. A explicação do seu funcionamento e de cada um dos seus elementos encontra-se na página web do projeto[29].

Figura 9 – Modelo Ptolemy para a chegada de ônibus e passageiros em uma determinada parada de ônibus.



Fonte: BERKELEY EECS, 2015b.

Costa (2015), desenvolveu um modelo de dados que permitiu o compartilhamento de informações de robôs na HLA. Para isso, foi necessário desenvolver componentes no Ptolemy que utilizassem um determinando modelo de dados proposto, o qual permitiu a integração de dispositivos de hardware ou robôs com a HLA. Além disso, sua pesquisa permitiu a co-simulação entre o Ptolemy e o simulador de robôs Stage (GERKEY; VAUGHAN; HOWARD, 2003).

Sua pesquisa foi baseada no trabalho de NEGREIROS e BRITO (2012), visto que já havia sido desenvolvida uma interface de comunicação entre o Ptolemy e a HLA por esses autores. Costa desenvolveu, como parte

[29] http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII8.1/ptII/ptolemy/domains/de/demo/Inspection/InspectionVergil.htm

de sua pesquisa, um ator que permitiu a troca de dados entre elementos do seu sistema e robôs ou simuladores de robôs, tal qual o Stage, durante a execução da federação (Figura 10).

Figura 10 –Ator Ptolemy desenvolvido por Costa (2015) que permite a comunicação de elementos do seu sistema com aplicações HLA.

Fonte: COSTA, 2015.

O ator desenvolvido por Costa (2015), e um diretor HLA disponibilizado no trabalho deste autor serão utilizados para a criação de um modelo que será utilizado como parte de um dos estudos de caso para a validação da abordagem.

A ferramenta Ptolemy foi escolhida como componenete para construção dos estudos de caso, para demostrar que, até mesmo com ferramentas completamente diferentes, como no caso do simulador SystemC e do arcabouço Ptolemy, a abordagem PowerHLA pode criar um ambiente de simulação heterogêneo, no qual os modelos e ferramentas que compõem esse ambiente podem trocar dados de maneira sincronizada e distribuída.

4.8. Considerações Finais do Capítulo

Neste capítulo, foi apresentada a abordagem desenvolvida, assim como aspectos essenciais de sua implementação. Foram abordados detalhes de implementação sobre a sincronização dos federados, publicação e assinatura de objetos, como também detalhes de configuração do ambiente de simulação. No capítulo foi apresentada plataforma MPSoCBench e suas tabelas de caracterização para processadores MIPS e SPARC, assim como o arcabouço Ptolemy.

Dois estudos de caso preliminares, criados a partir da plataforma MPSoCBench, utilizaram processadores MIPS e SPARC. Esses estudos de caso tiveram como objetivo demonstrar apenas a parte funcional de envio de dados de estimação do consumo de energia e a parte de coleta e geração dos arquivos CSV da abordagem PowerHLA. Os resultados obtidos durante os experimentos preliminares podem ser vistos no artigo "*Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations",* o qual se encontra anexo no final deste documento.

No Capítulo 5, serão apresentados os resultados obtidos durante o desenvolvimento e avaliação da abordagem proposta. Nele é apresentado um estudo de caso, composto por dois modelos: um modelo de um MPSoC e um modelo Ptolemy. O estudo de caso teve como objetivo demonstrar o funcionamento da abordagem na criação de um ambiente de simulação distribuído e heterogêneo.

Capítulo 5

Apresentação e Análise dos Resultados

Neste capítulo, serão apresentados os resultados obtidos durante o desenvolvimento e avaliação da abordagem proposta. A abordagem foi validada mediante um estudo de caso. O estudo de caso teve como objetivo demonstrar o funcionamento da abordagem na criação de um ambiente de simulação distribuído e heterogêneo. Neste estudo de caso foram utilizados dois modelos. O primeiro modelo consistiu em um MPSoC SystemC ESL de um núcleo MIPS, criado por meio da plataforma MPSoCBench. O segundo modelo consistiu de um modelo Ptolemy para geração de mensagens (cadeias de *bytes)*, nas quais seus *bytes* serão consumidos, de maneira sincronizada, pelo modelo MPSoC.

Três estudos de caso preliminares tiveram como objetivo demonstrar a parte funcional de envio de dados de estimação do consumo de energia e a parte de coleta e geração dos arquivos CSV da abordagem PowerHLA. Um deles caracterizado por um modelo RTL de um DPCM (*Differential Pulse-Code Modulation*) descrito em SystemVerilog/Verilog, foi utilizado para validar a coleta de dados de estimação de consumo de energia de modelos descritos em SystemVerilog/Verilog por meio da abordagem proposta. Os resultados desses estudos preliminares podem ser vistos no artigo "*Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations",* o qual se encontra anexo no final deste documento.

5.1. Resultados Obtidos para a Abordagem Proposta

Nesta seção, serão apresentados os resultados experimentais utilizados para validação da abordagem proposta. O uso da HLA permitiu que os

modelos, do presente estudo de caso, pudessem ser simulados por diferentes ferramentas (simuladores) de maneira sincronizada e distribuída.

Foi demonstrado que a abordagem desenvolvida proporciona a coleta e o agrupamento de dados de estimação do consumo de energia de modo centralizado, mesmo quando cada modelo utiliza uma abordagem diferente para a medição de consumo.

Na pesquisa, mediante o presente estudo de caso, também foi avaliado o impacto no desempenho da simulação diante do uso da abordagem PowerHLA.

Os resultados demonstram que a interface PowerHLA não interfere na consistência da simulação e que a abordagem cria suas saídas CSV com valores de dissipação de potência consistentes com os relatórios de dissipação de potência da plataforma MPSoCBench.

5.1.1. Estudo de Caso

O presente estudo de caso teve como objetivo demonstrar o funcionamento da abordagem PowerHLA na criação de um ambiente de simulação distribuído e heterogêneo.

Neste estudo de caso foram utilizados dois modelos. O primeiro modelo consiste em um MPSoC SystemC ESL de um núcleo MIPS, criado por meio da plataforma MPSoCBench. O segundo modelo consiste de um modelo Ptolemy para geração de mensagens (cadeias de *bytes)*, nas quais seus *bytes* serão consumidos, de maneira sincronizada, pelo modelo MPSoC. Este segundo modelo foi desenvolvido a partir do trabalho de Costa (2015), visto que a porção de código que torna o simulador Ptolemy compatível com a arquitetura HLA já estava implementada em seu trabalho e este disponibilizava os elementos necessários para a realização de uma simulação sincronizada com o modelo do MPSoC.

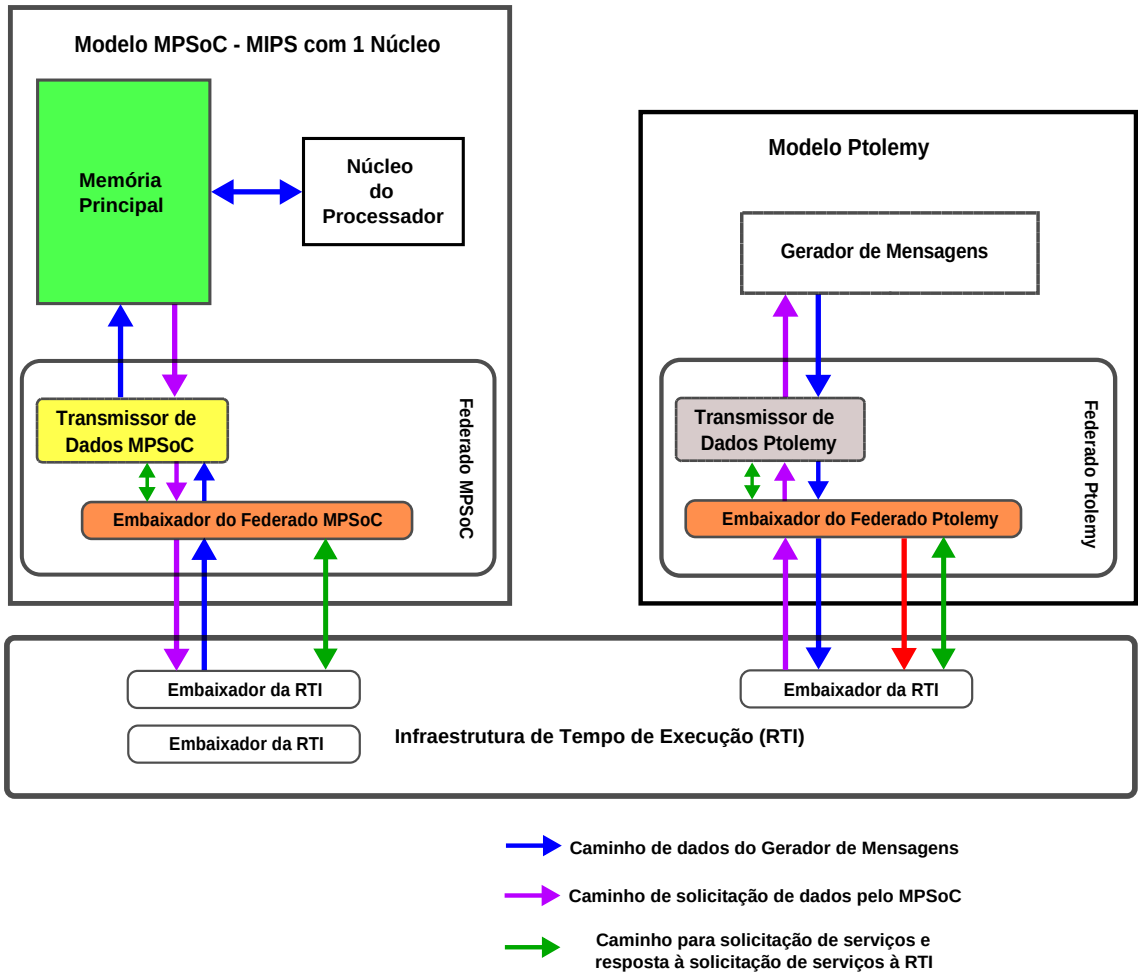
Os modelos foram distribuídos em duas diferentes máquinas com a mesma configuração: processador Intel i5 860 2.9 GHz, com 4 GB RAM, sistema operacional Linux 64 bits, gcc 4.8.1 e CERTI 3.4.3.

O modelo do MPSoC executa uma aplicação que define uma função de dispersão criptográfica (função *hash* criptográfica) denominada algoritmo de dispersão seguro (*Secure Hash Algorithm* - SHA). A partir de um dado de entrada (e.g., cadeia de caracteres), conhecido como mensagem

(*message*), o algoritmo SHA produz um valor de dispersão conhecido como resumo da mensagem (*message digest*). Funções *hash* criptográficas possuem aplicações no âmbito de segurança da informação e verificação de integridade de dados.

O modelo Ptolemy do Gerador de Mensagens gera os dados (*bytes*) que serão consumidos pela aplicação SHA que executa no modelo do MPSoC. A geração de dados do modelo Ptolemy está sincronizada com a aplicação SHA do MPSoC. Quando a aplicação SHA solicita um dado, uma função do HLA é chamada, fazendo com que a aplicação aguarde a chegada do dado do modelo Ptolemy. O Ptolemy, por sua vez, só envia um novo dado quando recebe do MPSoC uma confirmação de que o dado anterior foi recebido. Essas esperas são bloqueantes, o que faz com que o tempo simulado seja interrompido. Detalhes de sincronização serão apresentados na seção 5.1.1.1. Na Figura 11 é apresentado o ambiente de execução para a aplicação SHA.

Figura 11 - Ambiente de execução para a aplicação SHA.



Fonte: O autor, 2015.

Em uma visão global, os dados gerados pelo Gerador de Mensagens são enviados por meio do Federado Ptolemy, via RTI, para o Federado MPSoC. O Federado MPSoC, por sua vez, recebe esses dados e os disponibilizam para a escrita na memória principal do MPSoC, para que possam ser utilizados durante a execução da aplicação SHA. Este caminho de dados de aplicação é representado pelas setas de cor azul da Figura 11.

A cada dado recebido pelo Federado MPSoC, a memória principal deverá solicitar um novo dado ao Gerador de Mensagem do modelo Ptolemy, à medida que a aplicação SHA necessita de um novo dado para a composição da mensagem. A cada solicitação de dado realizada, uma chamada HLA é realizada, via Federado MPSoC, a qual bloqueia a execução da aplicação SHA e faz com que ela aguarde até que o dado solicitado ao Gerador de Mensagens seja recebido. Após receber o dado, a Memória Principal solicita um novo dado, confirmando o recebimento do dado anterior. Os dados são solicitados até que seja atingido o tamanho da mensagem.

Após recebidos todos os dados que compõem a mensagem, a aplicação SHA é executada e, ao seu término, é gerado um relatório indicando o fim de sua execução. Como saída, a aplicação é gera o resumo da mensagem. Este caminho é representado pelas setas de cor rosa da Figura 11.

Todos os federados solicitam serviços à RTI e também recebem respostas a essas solicitações. Este caminho é representado pelas setas de cor verde.

Em uma visão mais detalhada, o Federado MPSoC é constituído por dois principais elementos: o Transmissor de Dados MPSoC e o Embaixador do Federado MPSoC. O Transmissor de Dados MPSoC, é o elemento responsável pela disponibilização dos dados de aplicação oriundos do Gerador de Mensagens Ptolemy para a escrita memória principal do MPSoC. Esses dados de aplicação são recebidos por meio do Embaixador do Federado MPSoC.

De maneira análoga, o Federado Ptolemy também é constituído por dois elementos: um Transmissor de Dados Ptolemy e o Embaixador do Federado Ptolemy. O Transmissor de Dados Ptolemy é o elemento responsável pela transmissão dos *bytes* gerados pelo Gerador de Mensagens Ptolemy para o Federado MPSoC.

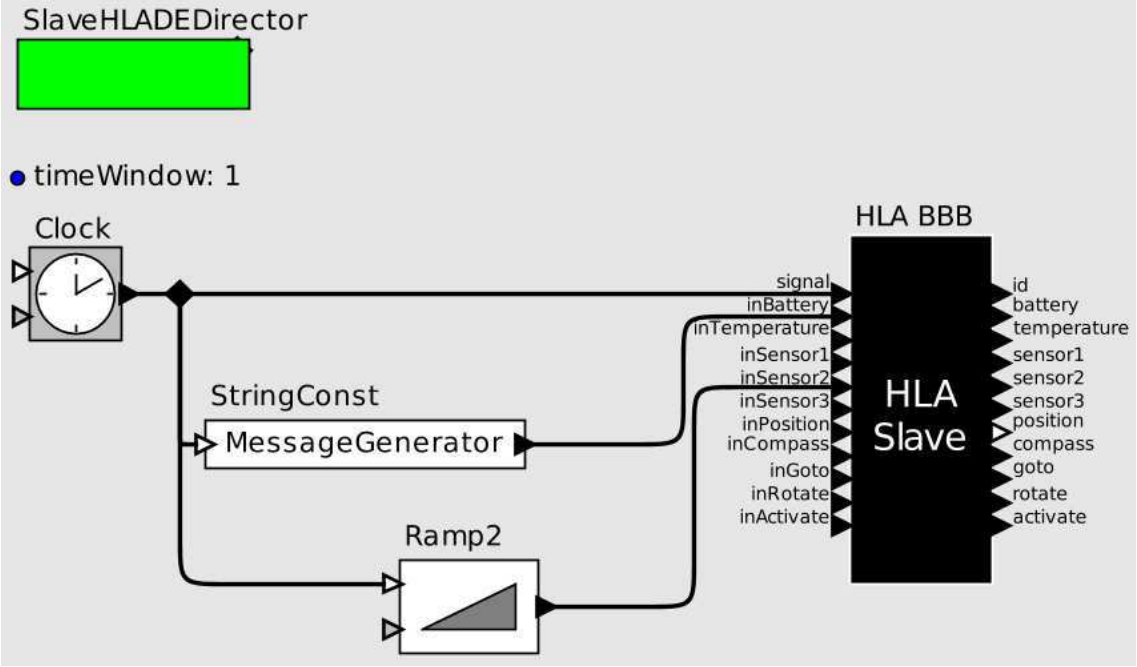
Na Figura 12 são apresentados os elementos que compõem o modelo Ptolemy para geração de mensagens. O primeiro elemento, o ator "MessageGenerator", é responsável por definir o nome do componente monitorado pelo Federado Ptolemy, aqui chamado de "PtolemyFederate".

O segundo ator, o "Ramp2", refere-se ao Gerador de Mensagens propriamente dito. Este ator gera valores sequenciais e crescentes a cada ciclo de relógio do ator "clock". O ator "clock" também dita quando o ator "MessageGenerator" será ativado. O "SlaveHLADEDirector" é o elemento que dita a sincronização e envio de dados do modelo Ptolemy compatível com a arquitetura HLA. Este diretor é uma variação do diretor para domínio de eventos discretos. Ele funciona como um diretor para o domínio de eventos discretos, porém pode utilizar atores como o "HLA BBB" para trocar dados com outros simuladores durante a execução da federação.

O ator "HLA BBB" representa a porção de código HLA que viabiliza a troca de dados entre o modelo Ptolemy e qualquer outro modelo compatível com a arquitetura HLA, contanto que este outro modelo assine ou publique atributos definidos no seu arquivo FED. O "HLA BBB" mapeia os componentes intrínsecos do arcabouço Ptolemy para atributos definidos no arquivo FED, os quais serão utilizados durante a simulação HLA.

Ambos os federados utilizam a Infraestrutura de Tempo de Execução (RTI) como meio de transmissão para os dados de aplicação. A implementação RTI utilizada no estudo de caso foi a CERTI (NOULARD; ROUSSELOT; SIRON, 2009).

Figura 12 - Elementos que compõem o modelo Ptolemy Gerador de Mensagens.



Fonte: O autor, 2015.

O primeiro experimento deste estudo de caso compreendeu no envio de oito *bytes* do modelo Ptolemy para o modelo SystemC. Cada byte enviado, representa um caractere na tabela ASCII (*American Standard Code for Information Interchange*). Foram enviados os números decimais 65, 66, 67, 68, 69, 70, 71 e 72, representando a cadeia de caracteres "ABCDEFGH".

À medida que os *bytes* são recebidos, os mesmos são escritos na Memória Principal do modelo MPSoC. Ao receber todos os 8 *bytes*, completando a mensagem, a aplicação SHA que é executada pelo modelo do MPSoC, lê a cadeia de caracteres e calcula o resumo da mensagem. O resumo da mensagem, para a cadeia de caracteres `ABCDEFGH`, foi `9ef0dce5 1ca340e5 fa08d6b2 c589fb0b 34521dab`.

Vale salientar que, a versão original da plataforma MPSoCBench não permitia que, nos modelos criados, dados de entrada fossem escritos diretamente em sua memória. Foram realizadas modificações no código fonte da plataforma MPSoCBench, as quais viabilizaram a escrita de valores disponibilizados pelo Transmissor de Dados MPSoC diretamente na memória

do modelo do MPSoC durante a execução da aplicação SHA, simulando assim um mecanismo DMA (*Direct Memory Access*).

5.1.1.1. Sincronização dos Modelos

Os federados de ambos os modelos foram definidos como sendo reguladores de tempo (*time-regulating federate*) e de tempo restringido (*time-constrained federate*). Além disso, o Federado MPSoC assina atributos da classe de objetos publicados pelo Federado Ptolemy e vice-versa. Estas características garantem que o Federado Ptolemy não avance no tempo a ponto de ultrapassar um ponto no qual mensagens TSO poderiam ainda ser enviadas pelo Federado MPSoC. De maneira análoga, o Federado MPSoC não poderá avançar no tempo e ultrapassar um ponto no qual mensagens TSO poderiam ainda ser enviadas pelo Federado Ptolemy.

Para sincronizar os modelos foi necessário verificar o tempo de ciclo de leitura dos *bytes* (caracteres) da memória do modelo MPSoC durante a execução da aplicação SHA. Apesar de não ser um modelo RTL, regido por um relógio, a partir da leitura do primeiro *byte*, cada ciclo de leitura do próximo *byte* da memória ocorre aproximadamente a cada 13 µs. A leitura do primeiro *byte* pelo SHA da memória do modelo MPSoC ocorre no tempo simulado SystemC de aproximadamente 182 µs. Até este momento, o processador foi iniciado e a aplicação foi transferida para a memória. Somente após este instante a aplicação ficará pronta e iniciará sua execução.

Diante da periodicidade para a leitura dos *bytes* da memória do modelo do MPSoC, decidiu-se que o tempo dos modelos avançaria de maneira discreta e constante. Na realização da simulação de tempo discreto o tempo de simulação é dividido em uma sequência de *timesteps* de igual tamanho. Assim, ambos os federados avançam de um *timestep* para o próximo. Em outros modelos onde não fosse detectada esta periodicidade, os modelos poderiam ser sincronizados por eventos, ou seja, o avanço do tempo poderia ser realizado à medida que todos os eventos para um determinado tempo fossem executados.

Por exemplo, se simulação estiver no tempo 1, ela só avançaria para o tempo 2 quando todos eventos ocorridos no tempo fossem executados. Este evento poderia ser, por exemplo, o momento em que o modelo MPSoC precisasse ler cada *byte* da memória principal. É importante ressaltar que a HLA permite a implementação de ambos os modos de sincronização: por tempo discreto e por eventos.

Assim, cada avanço de tempo realizado pelo Federado MPSoC ocorre a cada 13 μs de tempo simulado SystemC. Também foi definido que cada ciclo de leitura de um *byte* da memória do modelo MPSoC equivaleria a um ciclo de envio de dados do modelo Ptolemy. Para o modelo Ptolemy, um *byte* é enviado a cada 1 unidade do tempo de seu simulador, seguido da solicitação de avanço de tempo. Isto significa que um ciclo de 13 μs do tempo simulado do simulador SystemC que executa o modelo MPSoC equivale a 1 unidade do tempo simulado Ptolemy.

Dado que os federados de ambos os modelos são reguladores de tempo e de tempo restringido, e que o Federado MPSoC assina atributos da classe de objetos publicados pelo Federado Ptolemy e vice-versa, quando o modelo Ptolemy enviar um dado e solicitar o avanço no tempo, ele deverá esperar que o modelo do MPSoC solicite um avanço de tempo, antes de enviar um novo *byte*.

Os federados de ambos os modelos calculam o valor do próximo avanço no tempo por meio da soma do seu respectivo *lookahead* com o tempo atual da federação. Como o avanço de tempo é feito de forma discreta, o valor do *lookahead* de ambos os federados foi definido como sendo 1, ou seja, cada avanço de tempo solicitado por qualquer um destes, quando concedido, implicará na soma de uma unidade de tempo no tempo atual da federação.

Também foi definido que o modelo Ptolemy só começaria a gerar *bytes* a partir do momento da necessidade de leitura dos *bytes* por parte do modelo MPSoC. Desta forma, o modelo Ptolemy foi configurado para gerar dados a partir do tempo HLA de valor 13. O tempo HLA de valor 13 representa 13 ciclos de 13 μs no tempo simulado SystemC, ou seja, 169 μs. Isto significa que no momento 182 μs do tempo SystemC, ou seja, no

tempo HLA de valor 14, o valor do primeiro *byte* (valor decimal 65) estará disponível na memória principal do modelo MPSoC.

Quando o modelo do MPSoC recebe o *byte* do modelo Ptolemy, por meio da chamada de retorno *Reflect Attribute Values*[†], implementada no Embaixador do Federado MPSoC, o valor é lido e escrito na memória principal do MPSoC. O Código 19 representa o trecho de código responsável pela escrita do valor recebido pelo Transmissor de Dados MPSoC na memória principal do MPSoC. Como já mencionado, os *bytes* começam a ser lidos da memória principal pela aplicação SHA a partir do tempo simulado SystemC 182 μs.

Código 19 – Trecho de código responsável pela escrita do dado na memória do modelo do MPSoC.

```
1     if(sc_time_stamp().to_seconds() >= (nextTimeToAdvance)){
2           if((sc_time_stamp().to_seconds() >= next_time_to_write) &&
3           (addr_inc < TDATA_SIZE - 1)){
4                 unsigned char byte =
5                 power_HLA_interface::getInstance().read_byte();
6                 current_tdata_addr = addrBase + addr_inc;
7                 write_byte(current_tdata_addr, &byte);
8                 next_time_to_write = next_time_to_write + 0.000013;
9                 addr_inc++;
10                }
11          power_HLA_interface::getInstance().advanceTime(
12          SENDER_FEDERATE_LOOKAHEAD );
13    nextTimeToAdvance = nextTimeToAdvance + 0.000013;
14    }
```

A linha 1 do Código 19 representa o momento no qual o modelo MPSoC deve avançar no tempo. A variável `nextTimeToAdvance` representa o tempo, em segundos do tempo simulado SystemC, no qual o modelo MPSoC deverá avançar no tempo. Conforme foi definido, a cada avanço de tempo, esta variável incrementa seu valor a cada 13 us.

A variável `next_time_to_write` (linha 2, Código 19) define o tempo, em segundos do tempo simulado SystemC, no qual o modelo do MPSoC deverá iniciar a leitura dos *bytes* oriundos do modelo Ptolemy. O modelo Ptolemy foi sincronizado para começar a enviar os dados a partir do tempo HLA 13. O valor inicial da variável `next_time_to_write` foi definido como sendo 0.000182 (182 us). Assim, partir do momento 182 us do tempo simulado SystemC, ou seja, a partir do tempo HLA 14, o modelo MPSoC

iniciará a leitura dos *bytes* que estão sendo enviados pelo modelo Ptolemy. A cada escrita, o valor da variável `next_time_to_write` será somado de 0.000013, ou seja, 13 us (Código 19, linha 8).

Assim, no tempo 182 us SystemC, ou seja, no tempo HLA 14, o modelo do MPSoC lerá o primeiro *byte* que havia sido enviado, no tempo HLA 13, pelo modelo Ptolemy e escreverá este *byte* no primeiro endereço de memória utilizado pela aplicação SHA para armazenar o conteúdo da mensagem.

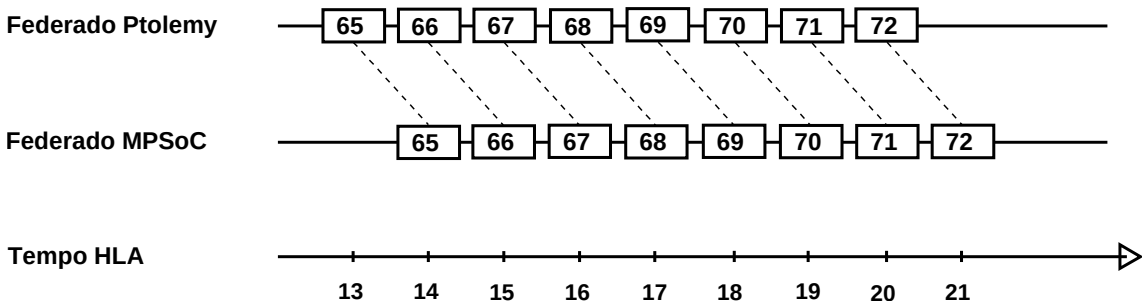
A aplicação SHA continuará solicitando a leitura de *bytes* do modelo Ptolemy até que a quantidade de *bytes*, definida como sendo o tamanho da mensagem, seja atingida. Essa quantidade de *bytes* é definida como sendo o valor da variável `TDATA_SIZE` decrescido de 1 unidade (Código 19, linha 3).

A aplicação SHA lê um conjunto de *bytes* escritos em endereços de memória consecutivos. O primeiro endereço de memória, o qual deve armazenar o primeiro *byte* lido pela aplicação, foi descoberto por meio de experimentos. Este endereço é representado pelo número decimal 59824, o qual é denominado `addrBase` no Código 19 (Linha 6).

A cada recebimento de *byte*, o valor do endereço de escrita do *byte* na memória principal do modelo MPSoC é incrementado de 1 unidade. Esse incremento, representado pela variável `addr_inc` (Código 19, linha 6), garante que os *bytes* recebidos sejam escritos nos endereços de memória consecutivos utilizados na execução da aplicação SHA.

O registro da execução da simulação demonstrando a sincronização de ambos os federados é apresentado no Registro 1 (Ver Anexo I). A partir desse registro foi gerada a Figura 13, representando a sincronização entre o Federado MPSoC e o Federado Ptolemy.

Figura 13 - Sincronização dos dados entre o Federado MPSoC e o Federado Ptolemy.



Fonte: O autor, 2015.

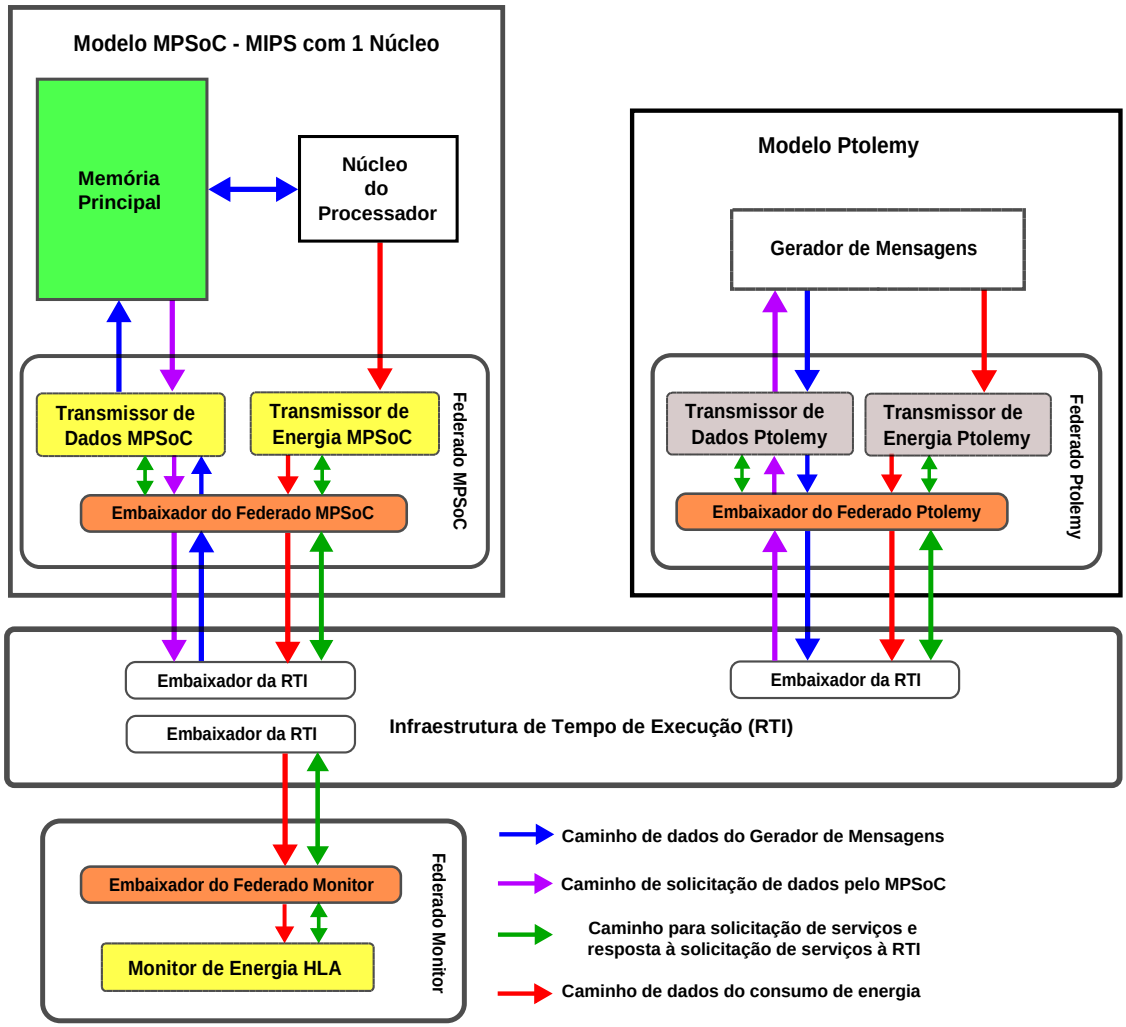
5.1.1.2. A Geração de Dados do consumo de Energia

Após a criação do ambiente de simulação distribuído e heterogêneo, constituído do modelo Ptolemy e do modelo do MPSC, o conjunto de métodos que define a parte de transmissão de dados do consumo energia da abordagem PowerHLA foi adicionada ao modelo do MPSoC. Esse conjunto de métodos é representado pelo Transmissor de Energia MPSoC. Para o modelo Ptolemy, também foi adicionado um elemento similar, o Transmissor de Energia Ptolemy (Figura 14).

Para a coleta de informações de energia dos modelos simulados, foi realizada a adição de mais um federado: O Federado Monitor. Este federado é composto por dois elementos, o Embaixador do Federado Monitor e o Monitor de Energia HLA. O Federado Monitor organiza e agrupa os dados de energia recebidos, gerando como saída arquivos no formato CSV (*Comma-separated values*). O caminho de dados do consumo de energia é representado pelas setas de cor vermelha da Figura 14. O Federado Monitor e a RTI foram executados na mesma máquina onde foi executado o modelo Ptolemy.

Os dados do consumo de energia gerados pelo Núcleo do Processador assim como pelo Gerador de Mensagens são enviados para o Federado Monitor, via RTI, pelo Transmissor de Energia MPSoC e Transmissor de Energia Ptolemy, respectivamente.

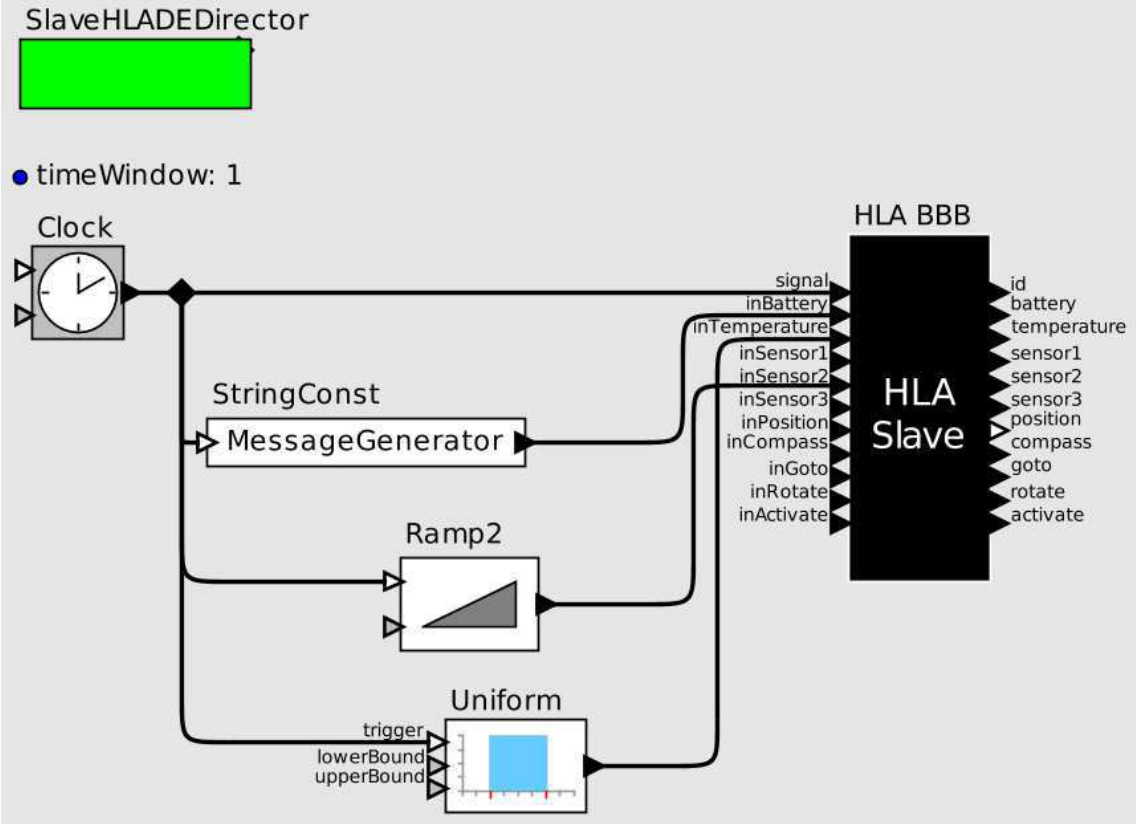
Figura 14 - Ambiente de execução para a aplicação SHA com coleta de dados do consumo de energia.



Fonte: O autor, 2015.

A energia consumida pelo núcleo do modelo do MPSoC foi estimada durante sua simulação, por meio da abordagem de estimação de consumo PowerSC (KLEIN et al., 2007a, 2007b). Para o Gerador de Mensagens do modelo Ptolemy, valores de energia arbitrários na faixa de 0,09 e 0,11 Joules foram gerados a cada envio de *byte*. Para geração desses valores por parte do modelo Ptolemy foi necessária a adição de mais um ator: O ator "Uniform" (Figura 15).

Figura 15 - Adição do ator *Uniform* no modelo Ptolemy Gerador de Mensagens.

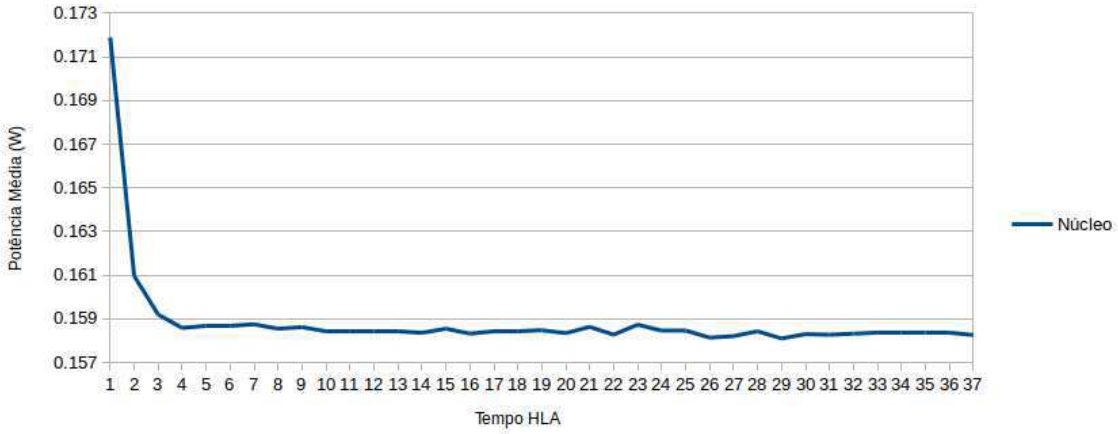


Fonte: O autor, 2015.

O núcleo do modelo do MPSoC acumula valores de energia em Joules durante a execução de cada instrução. A energia é acumulada durante um intervalo pré-definido pelo usuário que, no caso, foi definido como sendo o mesmo intervalo para cada avanço de tempo realizado pelo Federado MPSoC (ou seja, 13 µs). Assim, a cada 13 µs, o Federado MPSoC envia os dados de energia acumulados para o Federado Monitor e define como zero todas as respectivas variáveis relativas à energia do componente, no caso, o núcleo do processador.

Neste sentido, a potência média em Watts, a cada 13 µs, pode ser obtida dividindo o valor de energia acumulada dentro deste intervalo pelo valor 13 µs. Durante a simulação, foram obtidas 37 amostras de 13 µs. Graficamente, os dados de dissipação de potência obtidos do núcleo do modelo do MPSoC são apresentados na Figura 16. Os dados apresentados na Figura 16 podem ser verificados a partir do Registro 2 (Ver Anexo I).

Figura 16 - Potência média em função do tempo HLA para o processador MIPS com 1 núcleo.



Fonte: O autor, 2015.

Dados de energia são gerados pelo núcleo do modelo do MPSoC antes que a Memória Principal realize a leitura do primeiro *byte* para o cálculo do resumo da mensagem (tempo HLA 1 ao tempo HLA 13). Esse consumo de energia é consequência do carregamento da aplicação SHA na Memória Principal, assim como da inicialização das variáveis necessárias para execução da aplicação.

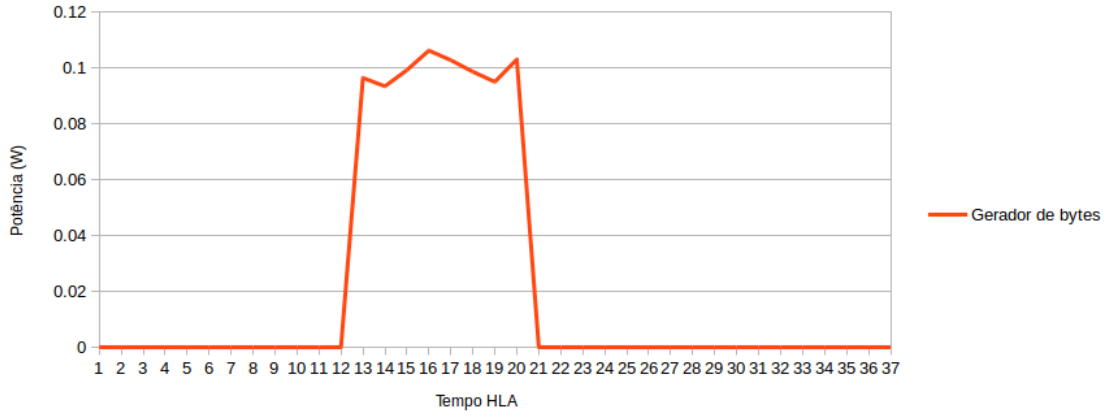
Também são gerados dados do consumo de energia após a leitura dos 8 *bytes* na Memória Principal. Este tempo após a leitura dos *bytes* contempla a fase de execução da aplicação para o cálculo do resumo da mensagem (tempo HLA 22 ao tempo HLA 37). Analisando a Figura 16, é possível perceber um curto pico de dissipação de potência exatamente no instante 23 do eixo do tempo HLA.

O Gerador de Mensagens do modelo Ptolemy envia dados do consumo de energia a partir do momento criação do primeiro *byte*, ou seja, no tempo HLA 13. A cada geração de *bytes*, ou seja, a cada 1 unidade de tempo, o modelo Ptolemy envia dados do consumo de energia. *Bytes* são criados pelo gerador do modelo Ptolemy até o tempo HLA 20, assim, o intervalo de envio de dados de energia deste componente é compreendido entre o tempo HLA 13 e o tempo HLA 20.

Graficamente, os dados de dissipação de potência do Gerador de Mensagens do modelo Ptolemy, obtidos a partir de seus dados de consumo

de energia, são apresentados na Figura 17. Os dados apresentados na Figura 4 podem ser verificados a partir do Registro 2 (Ver Anexo I).

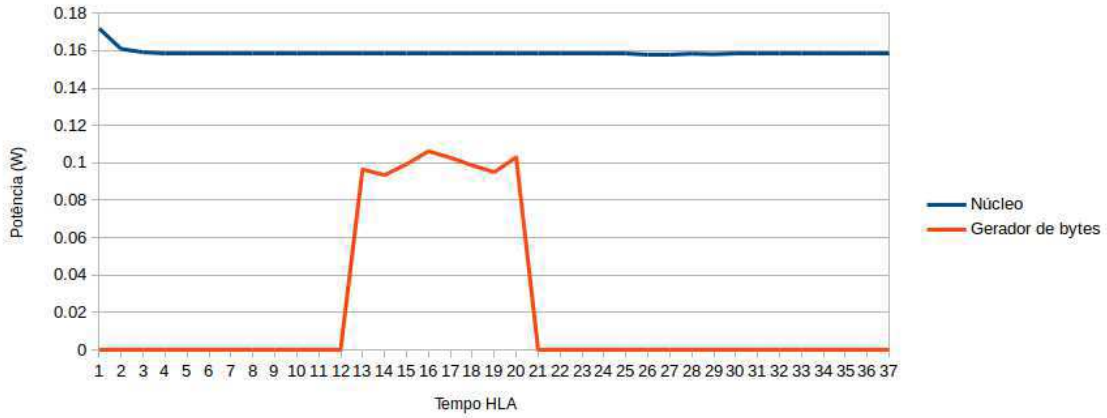
Figura 17 - Potência em função do tempo HLA para o Gerador de Mensagens.



Fonte: O autor, 2015.

A combinação dos gráficos apresentados nas Figuras 16 e 17 é representada na Figura 18.

Figura 18 - Potência em função do tempo HLA para processador MIPS com 1 núcleo e Gerador de Mensagens.



Fonte: O autor, 2015.

5.1.1.3. Publicação e Assinatura de atributos

Para que o Federado MPSoC possa receber valores de atributos enviados pelo Federado Ptolemy, o Federado MPSoC deve assinar previamente tais atributos. De maneira análoga, para que o Federado Ptolemy possa receber valores de atributos enviados pelo Federado MPSoC, o Federado Ptolemy também deve assinar atributos publicados pelo Federado MPSoC.

Costa (2015) desenvolveu um ambiente de simulação para integrar o simulador Ptolemy e o simulador Stage (GERKEY; VAUGHAN; HOWARD, 2003) utilizando a arquitetura HLA. Como já mencionado, o trabalho de Costa (2015) foi utilizado como base para a criação do modelo Ptolemy para geração de *bytes*, visto que a porção de código que torna o simulador Ptolemy compatível com a arquitetura HLA já havia sido implementada em seu trabalho.

Visando uma menor quantidade de modificações no código fonte do trabalho desenvolvido por Costa (2015), seu arquivo FED foi adotado para definição dos dados trocados entre o Federado MPSoC e o Federado Ptolemy. O arquivo FED utilizado é apresentado no Código 20.

Código 20 – Arquivo FED (Costa, 2015).

```
1     (FED
2     (Federation TestFed)
3     (FEDversion v1.3)
4     (spaces
5     )
6     (objects
7     (class ObjectRoot
8     (attribute privilegeToDelete reliable timestamp)
9     (class RTIprivate)
10    (class robot
11          (attribute id reliable timestamp)
12          (attribute battery reliable timestamp)
13          (attribute temperature reliable timestamp)
14          (attribute sensor1 reliable timestamp)
15          (attribute sensor2 reliable timestamp)
16          (attribute sensor3 reliable timestamp)
17          (attribute gps reliable timestamp)
18          (attribute compass reliable timestamp)
19          (attribute goto reliable timestamp)
20          (attribute rotate reliable timestamp)
21          (attribute activate reliable timestamp)
22    )
23    )
24    )
25    )
```

Tanto o modelo Ptolemy quanto o modelo do MPSoC utilizam os atributos da classe *robot* (Código 20, linha 10) para a troca de informações. Para a criação do ambiente de execução para a aplicação SHA com coleta de

dados do consumo de energia (Figura 14, Seção 5.1.1.2), os atributos `id`, `battery`, `temperature`, `sensor1` e `sensor2` foram utilizados para representar as informações de nome do federado, nome do componente monitorado, energia, tempo simulado e valor do `byte`, respectivamente. O restante dos atributos não é utilizado durante a execução da federação.

A cada geração de *byte*, o Federado Ptolemy envia os seguintes atributos:

- Nome do federado (`id`): Definido como sendo "PtolemyFederate";
- Nome do componente monitorado (`battery`): Definido como sendo "MessageGenerator";
- Energia (`temperature`): Definido como sendo um valor arbitrário entre 0,09 e 0,11 Joules;
- Valor do byte (`sensor2`): Definido como sendo um valor decimal incremental de 65 a 72.

Para enviar estes atributos, o Federado Ptolemy deve, previamente, informar a RTI a intenção de publicá-los[30] como também registrar uma instância da classe de objeto que contém estes atributos, no caso, registrar uma instância do objeto[31] da classe robot. A partir deste ponto, o Federado Ptolemy estará apto para realizar atualizações desses atributos[32], o que implica no envio dos dados representados pelos mesmos. No Federado Ptolemy, cada atualização de atributo é seguida por uma solicitação de avanço de tempo[33].

Para receber estes atributos, o Federado MPSoC deve assiná-los, realizando a chamada do serviço *Subscribe Object Class Attributes*. O recebimento destes dados se traduz no recebimento de um *byte* do modelo Ptolemy.

[30] Por meio da chamada do serviço *Publish Object Class Attributes*.

[31] Por meio da chamada do serviço *Register Object Instance*.

[32] Por meio da chamada do serviço *Update Attribute Values*.

[33] Por meio da chamada do serviço *Time Advance Request*.

Para confirmar o recebimento do *byte* ao Federado Ptolemy, o Federado MPSoC deve enviar alguma informação de recebimento. Por sua vez, o Federado Ptolemy deverá assinar os atributos publicados pelo Federado MPSoC. As informações enviadas pelo Federado MPSoc são:

- Nome do federado (`id`): Definido como sendo "MPSoCFederate";
- Nome do componente monitorado (`battery`): Definido como sendo "TLMmemory" ou "ProcessorCore". O nome do componente será definido como "TLMmemory", quando um dado é recebido pelo Transmissor de Dados MPSoC, este envia uma confirmação de recebimento do dado, informando que o componente "TLMmemory" (no caso, a memória principal) recebeu o dado. O nome do componente será "ProcessorCore" (no caso, o núcleo do processador) quando a atualização dos atributos for referente aos dados do consumo de energia do núcleo do processador;
- Energia (`temperature`): A energia consumida pelo componente. Utilizado apenas nas atualizações referentes ao componente "ProcessorCore" do modelo MPSoC;
- Tempo simulado (`sensor1`): O tempo simulado atual do simulador SystemC. Utilizado apenas nas atualizações referentes ao componente "ProcessorCore" do modelo MPSoC;
- Valor do *byte* (`sensor2`): O valor decimal do *byte* recebido pelo modelo Ptolemy. É utilizado apenas na confirmação do recebimento do *byte* pelo componente "TLMmemory" do modelo MPSoC.

Similarmente ao Federado Ptolemy, antes de enviar estes atributos, o Federado MPSoC deve informar a RTI a intenção de publicá-los como também registrar uma instância da classe de objeto que contém estes atributos, no caso, também registrar uma instância do objeto da classe *robot*. A partir deste ponto, o Federado MPSoC também estará apto para realizar atualizações desses atributos, o que implica no envio dos mesmos.

As atualizações dos atributos no Federado MPSoC são seguidas de uma solicitação de avanço de tempo. Dado que ambos os federados são reguladores de tempo e de tempo restringido, e que o Federado MPSoC assina atributos da classe de objetos publicados pelo Federado Ptolemy e

vice-versa, quando o modelo Ptolemy enviar um dado e solicitar o avanço no tempo, ele deverá esperar que o modelo MPSoC solicite um avanço de tempo, antes de enviar um novo *byte*.

O Federado Monitor foi definido apenas como federado de tempo restringido. Sua função consiste em receber os objetos publicados pelo Federado MPSoC e pelo Federado Ptolemy, imprimir os dados dos objetos recebidos e agrupar os dados de energia de cada federado, de maneira ordenada, gerando como saída arquivos no formato CSV (Comma-separated values). O Federado Monitor assina os atributos `id`, `battery`, `temperature`, `sensor1` e `sensor2`. Assim, todos os objetos publicados pelo Federado MPSoC e pelo Federado Ptolemy serão recebidos pelo Federado Monitor (Anexo I, Registro 2).

5.1.1.3.1. Medição do Tempo de Simulação

A fim de avaliar o impacto no desempenho da simulação diante do uso da abordagem PowerHLA, foram realizados dois experimentos. O primeiro experimento consistiu em verificar o tempo de simulação do modelo MPSoC para a execução de 1000 rodadas da aplicação SHA, utilizando mensagens de 8 *bytes.* Neste experimento o modelo do MPSoC troca informações com modelo Ptolemy de maneira sincronizada e distribuída via HLA, sem utilizar o conjunto de métodos e classes que define a parte de transmissão de dados de consumo de energia da abordagem PowerHLA (Figura 11, Seção 5.1.1).

O segundo experimento foi semelhante ao primeiro, porém, introduzindo o conjunto de métodos e classes que define a parte de transmissão de dados de consumo de energia da abordagem PowerHLA, ou seja, introduzindo o Transmissor de Energia MPSoC (Figura 14, Seção 5.1.1.2). Na Tabela 1 é apresentado o tempo de simulação de cada um desses experimentos.

No segundo experimento, em relação ao tempo de simulação do primeiro experimento, houve um acréscimo de 17,39 %. O fato de se habilitar a coleta do consumo de dados de energia implica na adição de outro federado durante a execução da federação, o Federado Monitor. Essa

adição de mais um federado, tem como consequência um aumento no fluxo de dados que transitam pela federação, assim como um aumento das chamadas de serviços de sincronização na RTI, justificando tais acréscimos.

Tabela 1 – Tempo de simulação dos experimentos.

Experimento	Tempo de simulação (seg.)
1 (PowerHLA sem coleta de energia)	46
2 (PowerHLA com coleta de energia)	54

Para ambos os experimentos, o tempo simulado foi de 0,157754 segundos. De maneira análoga, a potência total dissipada pelo processador em todos os experimentos foi de 0,157781 W, o que demonstra que a abordagem não interfere na consistência da simulação. A potência total dissipada é apresentada em forma de relatório pela plataforma MPSoCBench ao final da execução dos experimentos. Ambos os experimentos apresentaram relatórios de dissipação de potência idênticos (Registro 3).

Registro 3 - Relatório de informação de potência dissipada.

```
---------------------------------------------- POWER REPORT
---------------------------------------
  L  -      Cell  Name      -   Cell Type   -  Leakage Power  -
Internal Power   -  Aggregate Power
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
 RT  - proc0                - Processor     - 0.000000e+00    -
0.000000e+00      - 1.577810e-01
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
Summary:
  Switching power :    0.000000e+00W
  Internal power  :    0.000000e+00W
  Leakage power   :    0.000000e+00W
  Aggregate power :    1.577810e-01W
                     ----------------
  TOTALS          :    1.577810e-01W
```

5.2. Considerações Finais do Capítulo

Neste capítulo, foram apresentados os resultados obtidos durante o desenvolvimento e validação da abordagem. O estudo de caso apresentado no capítulo teve como objetivo demonstrar o funcionamento da abordagem na criação de um ambiente de simulação distribuído e heterogêneo.

Nesse estudo de caso foram utilizados dois modelos: um modelo de um MPSoC SystemC ESL de um núcleo MIPS, criado por meio da plataforma MPSoCBench, e um modelo Ptolemy. Os resultados demonstram a eficácia da abordagem PowerHLA na criação de um ambiente de simulação distribuído e heterogêneo.

No Capítulo 6, serão apresentadas as considerações finais sobre a pesquisa, assim como suas contribuições, limitações e sugestões para pesquisas futuras.

Capítulo 6

Considerações Finais

Neste capítulo, serão apresentadas as considerações finais sobre a pesquisa, assim como suas contribuições, limitações e sugestões para pesquisas futuras.

Dispositivos eletrônicos estão cada vez mais complexos e com mais restrições sobre o consumo de energia. A pesquisa ora descrita teve como objetivo prover um meio para distribuir e simular modelos descritos em diversas linguagens e/ou níveis de abstração, utilizando diferentes ferramentas, de maneira combinada, para a coleta de dados da estimação do consumo de energia de forma sincronizada. Os resultados experimentais demonstraram a flexibilidade e a eficácia da abordagem proposta.

A pesquisa desenvolvida gerou a publicação de um artigo para uma conferência, o SBCCI 2015[34] (*Symposium on Integrated Circuits and Systems Design*). O artigo (*Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations*) se encontra anexo no final deste documento.

6.1. Contribuições da Pesquisa

A principal contribuição da pesquisa foi o desenvolvimento de uma abordagem, utilizando *High Level Architecture* (HLA), que permite a criação de um ambiente de simulação integrado e heterogêneo, composto por diferentes ferramentas e modelos. Esses modelos foram descritos em diferentes linguagens e utilizaram diferentes abordagens de estimação do consumo de energia. O uso da HLA permitiu que os modelos pudessem ser simulados por diferentes simuladores de maneira sincronizada e distribuída. Também foi demonstrado que a abordagem desenvolvida proporciona a coleta e o agrupamento de dados de estimação do consumo de energia de

[34] http://www.chipinbahia.eng.ufba.br/sbcci-2015/home

modo centralizado, mesmo quando cada modelo utiliza uma abordagem diferente para a medição de consumo.

Os modelos podem ser descritos em C++, SystemC, SystemVerilog, Verilog e Java (no caso do Ptolemy) e utilizar diferentes abordagens para estimação de consumo de energia.

O uso da HLA, como parte da abordagem desenvolvida permite que outras ferramentas (e.g., arcabouços, simuladores) possam atuar de maneira combinada e sincronizada com os modelos simulados, dando flexibilidade à abordagem. Destaca-se, também, que HLA é um padrão IEEE já difundido, o que facilita sua adoção por desenvolvedores de outras ferramentas.

Outro aspecto relevante a destacar da PowerHLA reside no fato de que sua interface (Interface PowerHLA) pode ser estendida para se adequar a outras linguagens de programação.

6.2. Limitações e Sugestões para Pesquisas Futuras

A abordagem desenvolvida foi validada apenas por meio de um estudo de caso. Dentre as abordagens para estimação de consumo estudadas, apenas uma abordagem, a denominada PowerSC, foi utilizada para os modelos dos MPSoC. Porém, outras abordagens de estimação de consumo poderiam ter sido utilizadas.

Essas abordagens de estimação de consumo de energia podem ser integradas nos modelos para avaliar, por exemplo, a precisão e o tempo de simulação de cada uma das delas.

Simulações de modelos a partir de outras ferramentas podem ser executadas, a fim de avaliar melhor o impacto da aplicação da abordagem no tempo de simulação dos experimentos.

A abordagem precisa de melhorias em termos de codificação para o devido tratamento de alguns tipos de exceções e erros que podem ser causados por falhas do usuário no momento de configuração do ambiente, durante a edição do arquivo “defines.h”.

Por exemplo, definir a constante número de componentes monitorados com um valor diferente do número de componentes

monitorados, que é instanciado dentro do código, causará um erro no início da execução da federação.

Outro exemplo: definir federados com nomes iguais não acarreta em erro de execução, porém faz com que apenas um deles consiga afiliar-se na federação. Quando o segundo federado solicitar se afiliar, não conseguirá, e uma mensagem exceção será lançada. Mensagens mais intuitivas podem ser apresentadas ao usuário quando ocorrer esse tipo de exceção.

Apesar de a abordagem ter sido validada, outras ferramentas poderiam ser utilizadas para demonstrar, de forma mais efetiva, a sua flexibilidade na construção de ambientes de simulação heterogêneos.

Modelos mais complexos podem ser utilizados para melhor avaliar o impacto no desempenho da simulação diante do uso da abordagem PowerHLA. Além disso, modelos que fazem uso de técnicas de redução de energia (e.g.,*power gating*, *clock gating*) podem ser utilizados a fim de verificar a eficácia da abordagem PowerHLA em simulá-los de maneira sincronizada e distribuída.

Referências Bibliográficas

ABRIL, A. et al. Energy estimation and optimization in architectural descriptions of complex embedded systems. **VLSI Circuits and Systems II**, v. 33, n. 0, 2005.

ACCELERA. **Accelera Systems Initiative**. Disponível em: <http://accellera.org/>. Acesso em: 3 maio. 2015.

AHUJA, S. et al. Power estimation methodology for a high-level synthesis framework. **Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design**, p. 541 – 546, 2009.

AHUJA, S. **High Level Power Estimation and Reduction Techniques for Power Aware Hardware Design**. Tese (Doutorado) - Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2010.

ANDERSON, J. H.; MEMBER, S.; NAJM, F. N. Power Estimation Techniques for FPGAs. **Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,** v. 12, n. 10, p. 1015–1027, 2004.

ARM LTD. **AMBA Specifications**. Disponível em: <http://www.arm.com/products/system-ip/amba/amba-open-specifications.php>. Acesso em: 29 jun. 2015.

ASCIA, G. et al. A Multiobjective Genetic Fuzzy Approach for Intelligent System-level Exploration in Parameterized VLIW Processor Design. **2006 IEEE International Conference on Evolutionary Computation**, 2006.

AYNSLEY, J. OSCI TLM-2.0 language reference manual. **Open SystemC Initiative (OSCI), Tech. Rep**, n. July, 2009.

BAILEY, B.; MARTIN, G.; PIZIALI, A. **ESL Design and Verification**. Burlington: Morgan Kaufmann/Elsevier, 2007.

BANSAL, N. et al. **Power monitors: a framework for system-level power estimation using heterogeneous power models**, 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design. **Anais**...IEEE Computer Soc, 2005

BELTRAME, G.; SCIUTO, D.; SILVANO, C. Multi-Accuracy Power and Performance Transaction-Level Modeling. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, 2007.

BENINI, L.; HODGSON, R.; SIEGEL, P. System-level power estimation and optimization. **Power Electronics and Design**, p. 173–178, 1998.

BERKELEY EECS. **The Spice Page**. Disponível em: <http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/>. Acesso em: 4 jun. 2015a.

BERKELEY EECS. **The Ptolemy Project**. Disponível em: <http://ptolemy.eecs.berkeley.edu/>. Acesso em: 17 jul. 2015b.

BOUHADIBA, T.; MOY, M.; MARANINCHI, F. System-Level Modeling of Energy in TLM for Early Validation of Power and Thermal Management. **Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013**, p. 1609–1614, 2013.

BRAND, D.; VISWESWARIAH, C. **Inaccuracies in Gate-Level Power Estimation**, Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on. **Anais**...1996

C++ STANDARDS COMMITTEE AND OTHERS. **ISO/IEC 14882: 2011, Standard for Programming Language C++Office**. New York: ISO, 2011.

CADENCE. **Cadence Design Systems**. Disponível em: <www.cadence.com>. Acesso em: 3 jun. 2015.

CADENCE DESIGN SYSTEMS. **Open Verification Methodology**. Disponível em: <http://www.cadence.com/>. Acesso em: 12 maio. 2015.

CALDARI, M. et al. Instruction based power consumption estimation methodology. **Electronics, Circuits and Systems, 2002. 9th International Conference on**, v. vol.2, p. 721 – 724, 2002.

CALDARI, M. et al. System-level power analysis methodology applied to the AMBA AHB bus. **Proceedings of the conference on Design, Automation and Test in Europe: Designers**, 2003.

CALYPTO. **Calypto Design Systems**. Disponível em: <http://calypto.com/en/products/catapult/overview>. Acesso em: 1 jul. 2015.

ÇAYIRCI, E.; MARINCIC, D. **Computer Assisted Exercises and Training: A Reference Guide**. New Jersey: Wiley, 2009.

CHEN, W.; DOEMER, R.; CENTER. ConcurrenC: A new approach towards effective abstraction of C-based SLDLs. In: **Analysis, Architectures and Modelling of Embedded Systems**. Langenargen: Springer, 2009. p. 57–65.

CONTI, M. et al. **Solutions on Embedded Systems**. New York: Springer, 2011.

COSTA, L. F. S.; *Aspetos de sincronização em co-simulação de equipas de robôs*. 2015. 43 f. Dissertação (Mestrado em Engenharia Eletrotécnica e de

Computadores). Coordenação de Pós-Graduação de Engenharia Eletrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2015.

CSU. **California State University, Chico**. Disponível em: <http://www.ecst.csuchico.edu/~hla/courses.html>. Acesso em: 15 jul. 2015.

DAAD. **Deutscher Akademischer Austausch Dienst**. Disponível em: <http://www.daad.org.br/pt/>. Acesso em: 26 maio. 2015.

DAMAŠEVIČIUS, R. Estimation of design characteristics at rtl modeling level using SystemC. **Information Technology and Control**, v. 35, n. 2, p. 117–123, 2006.

DING, C.-S.; TSUI, C.-Y.; PEDRAM, M. Gate-level Power Estimation Using Tagged Probabilistic Simulation 1 Introduction. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, p. 1099 – 1107, 1998.

DOD. **U.S. Department of Defense High-Level Architecture Object Model Template Specification**. Washington: DoD, 1998. v. 1998

DOULOS. **Doulos - Developing and Delivering KnowHow**. Disponível em: <https://www.doulos.com/knowhow/sysverilog/tutorial/dpi/>. Acesso em: 14 jul. 2015.

DUENHA, L. et al. **MPSoCBench: A toolset for MPSoC system level** evaluation, 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV). **Anais**...IEEE, jul. 2014

DUMOND, R.; LITTLE, R. **A Federation Object Model (FOM) Flexible Federate Framework**. Pittsburgh: Carnegie Mellon University, 2003.

EDABOARD. **EDABOARD**. Disponível em: <http://www.edaboard.com/thread165351.html>. Acesso em: 2 jul. 2015.

EISNER, C.; NAHIR, A.; YORAV, K. Functional verification of power gated designs by compositional reasoning. **Formal Methods in System Design**, v. 35, n. 1, p. 40–55, 2009.

FUJIMOTO, R. M. **Time management in the High Level Architecture**. Simulation. v.71, n.6, 1998.

FUJIMOTO, R. M. **Parallel and Distributed Simulation Systems**. New York: John Wiley & Sons, Inc, 2000.

GERKEY, B. P.; VAUGHAN, R. T.; HOWARD, A. The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems. **Proceedings of the**

International Conference on Advanced Robotics (ICAR 2003), n. Icar, p. 317–323, 2003.

GIAMMARINI, M.; CONTI, M.; ORCIONI, S. **System-level energy estimation with powersim**2011 18th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2011. **Anais**...2011

GIAMMARINI, M.; ORCIONI, S. **Powersim**. Disponível em: <http://sourceforge.net/projects/powersim>. Acesso em: 5 ago. 2015.

GIRARD, P.; NICOLICI, N.; WEN, X., **Power-Aware Testing and Test Strategies for Low Power Devices**, Boston, MA: Springer Verlag, 2009.

GIVARGIS, T.; VAHID, F. Instruction-based system-level power evaluation of system-on-a-chip peripheral cores. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 10, n. 6, p. 856–863, 2002.

GOERING, R. **SystemC director quits, blasts design initiative**. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1215992>. Acesso em: 3 maio. 2015.

GOMBINER, J. Carbon Footprinting the Internet. **Consilience: The Journal of Sustainable Development**, v. 5, n. 1, p. 119–124, 2011.

GRAHAM, P. **ANSI Common Lisp**. Saddle River: Prentice Hall, 1995.

GREAVES, D.; YASIN, M. **TLM POWER3: Power estimation methodology for SystemC TLM** 2.0, Lecture Notes in Electrical Engineering. Springer Verlag, 2014

GRÖTKER, T. et al. **System Design with SystemC TM**. 1st. ed. Hingham: Springer US, 2002.

GUEDES, M. et al. An automatic energy consumption characterization of processors using ArchC. **Journal of Systems Architecture**, v. 59, n. 8, p. 603–614, 2013.

GUPTA, P. **Resource-Constraint And Scalable Data Distribution Management For High Level Architecture**. Orlando: University of Central Florida, 2007.

HENZLER, S. **Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies**. New York: Springer, 2006.

HSIEH, W. T. et al. **System power analysis with DVFS on ESL virtual platform**International System on Chip Conference. **Anais**...2011

HSIEH, W.-T.; YEH, J.-C.; HUANG, S.-Y. PAC Duo System Power Estimation at ESL. **Proceedings of the 2010 Asia and South Pacific Design Automation Conference**, p. 815–820, 2010.

HUANG, W. et al. **Interaction of scaling trends in processor architecture and cooling**, Annual IEEE Semiconductor Thermal Measurement and Management Symposium. **Anais**...2010

IEEE COMPUTER SOCIETY. **IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)**. 1516.3. ed. New York: IEEE, 2003.

IEEE COMPUTER SOCIETY. **IEEE 1801-2009: IEEE Standard for Design and Verification of Low-Power Integrated Circuits**. 1801.–2013. ed. New York: IEEE, 2009.

IEEE COMPUTER SOCIETY. **IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules**. 1516. ed. New York: IEEE, 2010a.

IEEE COMPUTER SOCIETY. **IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification**. 1516.1. ed. New York: IEEE, 2010b.

IEEE COMPUTER SOCIETY. **IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model template (OMT) Specification**. 1516.2. ed. New York: IEEE, 2010c.

IEEE COMPUTER SOCIETY. **IEEE Standard for Standard SystemC Language Reference Manual**. New York: IEEE, 2012.

IEEE COMPUTER SOCIETY. **IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language**. New York: IEEE, 2013.

IKHWAN, L. et al. **PowerViP: SoC Power Estimation Framework at Transaction Level**ASP-DAC '06 Proceedings of the 2006 Asia and South Pacific Design Automation Conference. **Anais**...2006

JADCHERLA, S. et al. **Verification Methodology Manual for Low Power**. New York: Synopsys, 2009.

JAVVIN TECHNOLOGIES. **Network Dictionary**. 1st. ed. Saratoga: Javvin Press, 2007.

JIANG, C. et al. **Dynamic voltage/frequency scaling for power reduction in data centers: Enough or not?**, 2009 Second ISECS International Colloquium on Computing, Communication, Control, and Management, CCCM 2009. **Anais**...2009

KAWA, J. Low Power and Power Management for CMOS—An EDA Perspective. **IEEE Transactions on Electron Devices**, v. 55, n. 1, p. 186–196, 2008.

KEATING, M. et al. **Low power methodology manual: For system-on-chip design**. New York: Springer US, 2007.

KIM, B.; CHOI, C.; KIM, T. **Multifaceted modeling and simulation framework for system of systems using HLA/RTI**, CNS '13 Proceedings of the 16th Communications & Networking Symposium. **Anais**...2013

KIM, N. S. et al. Frequency and yield optimization using power gates in power-constrained designs. **Islped 2009.**, p. 121–126, 2009.

KIM, N. S. et al. Leakage Current: Moore's Law Meets Static Power. **Computer**, v. 36, n. 12, p. 68–75, 2003.

KLEIN, F. et al. PowerSC: A SystemC-based framework for power estimation. p. 19, 2007a.

KLEIN, F. et al. An efficient framework for high-level power exploration. **2007 50th Midwest Symposium on Circuits and Systems**, p. 1046–1049, ago. 2007b.

KUEHNLE, M.; WAGNER, A.; BECKER, J. **A statistical power estimation methodology embedded in a SystemC code translator**Proceedings of the 24th symposium on Integrated circuits and systems design - SBCCI '11. **Anais**...New York, New York, USA: ACM Press, 2011.

KUHL, F.; WEATHERLY, R.; DAHMANN, J. **Creating Computer Simulation Systems: An Introduction to the High Level Architecture**. 1st. ed. Saddle River: Prentice Hall, 1999.

LANDMAN, P. **High-level power estimation**Proceedings of 1996 International Symposium on Low Power Electronics and Design. **Anais**...IEEE, 1996.

LIN, T.-J. et al. Overview of ITRI PAC project - from VLIW DSP processor to multicore computing platform. **VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on**, p. 188 – 191, 2008.

LIU, F. et al. AOP-based high-level power estimation in SystemC. **Proceedings of the 20th symposium on Great lakes symposium on VLSI - GLSVLSI '10**, p. 353, 2010.

LIU, X.; PAPAEFTHYMIOU, M. C. A Markov chain sequence generator for power macromodeling. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 23, n. 7, p. 1048–1062, 2004.

MA, J. T. H.; AZEVEDO, R. **Estimativa de consumo de energia em nivel de instrucao para processadores modelados em ArchC, in: Workshop de Sistemas Computacionais**WSCAD-SSC. **Anais**...São Paulo: SBC, 2009

MACII, E. High-Level Power Modeling, Estimation, and Optimization. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**. v. 17, n. 11, p. 1061–1079, 1998.

MARIANI, G. et al. Multi-processor system-on-chip Design Space Exploration based on multi-level modeling techniques. **2009 International Symposium on Systems, Architectures, Modeling, and Simulation**, 2009.

MATTOS, J.; JR, L. R.; PILLA, M. **Desafios e Avanços em Computação - o estado da arte**. Pelotas: Editora e Gráfica Universitária, 2009.

MAZDAK AND ALBORZ DESIGN AUTOMATION. **V2SC**. Disponível em: <http://www.mazdak-alborz.com/v2sc.html>. Acesso em: 1 jul. 2015.

MENTOR. **Mentor Graphics**. Disponível em: <http://www.mentor.com/>. Acesso em: 19 ago. 2015a.

MENTOR. **UVM connect**. Disponível em: <http://www.mentor.com/products/fv/multimedia/uvm-connect>. Acesso em: 19 ago. 2015b.

MCCULLOUGH, J. C. et al. **Evaluating the effectiveness of model-based power characterization**USENIXATC'11 Proceedings of the 2011 USENIX conference on USENIX annual technical conference. **Anais**...2011

MOHANTY, S. P. et al. **Low-Power High-Level Synthesis for Nanoscale CMOS Circuits**. Boston: Springer US, 2008.

MOY, M. **Mini power-aware TLM-platform**. Disponível em: <http://www-verimag.imag.fr/~moy/?Mini-Power-Aware-TLM-Platform>. Acesso em: 1 jul. 2015.

MUHAMMAD, S.; IQBAL, Z.; LIANG, Y. ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems. **Computer Architecture Letters**. v. 9, n. 2, p. 45–48, 2010.

MULIADI, L. **Discrete Event Modeling In Ptolemy II**. Berkeley: Department of Electrical Engineering and Computer Science University of California, 1999.

NEGREIROS, A. L. V.; BRITO, A. V. **The development of a methodology with a tool support to the distributed simulation of heterogeneous and complexes embedded systems**. In Computing System Engineering (SBESC), 2012 Brazilian Symposium on, p. 37–42, Nov 2012. doi:10.1109/SBESC.2012.16.

NOULARD, E.; ROUSSELOT, J.-Y.; SIRON, P. CERTI, an Open Source RTI, why and how. **Fall Simulation**, 2009.

ORACLE. **Oracle Corporation**. Disponível em: <http://docs.oracle.com/cd/E19253-01/806-1379-10/mmapbus.html>. Acesso em: 3 jun. 2015.

OST, L. et al. **A high abstraction, high accuracy power estimation model for networks-on-chip**Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design Chip on the Dunes - SBCCI '09. **Anais**...New York, New York, USA: ACM Press, 2009

PARK, Y.-H. et al. **System level power estimation methodology with H.264 decoder prediction IP case study**, 2007 25th International Conference on Computer Design. **Anais**...IEEE, out. 2007.

PATERSON, D. J.; HOUGLAND, E. S.; SANMIGUEL, J. J. A Gateway / Middleware HLA implementation and the extra Services that can be provided to the Simulation. **Fall Simulation Interoperability Workshop**, p. 9, 2000.

PEDRAM, M. Power minimization in IC design: principles and applications. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, v. 1, n. 1, p. 3 – 56, 1996.

PEDRAM, M. Advanced Power Estimation Techniques. In: **Low Power Design in Deep Submicron Electronics**. Los Angeles: University of Southern California, 1997. p. 23.

PIMENTEL, A. D.; ERBAS, C.; POLSTRA, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. **IEEE Transactions on Computers**, v. 55, n. 2, p. 99–111, 2006.

RETHINAGIRI, S. K. et al. **PETS: Power and Energy Estimation Tool at System-Level**, 15th Int'l Symposium on Quality Electronic Design. Anais...2014

RETHINAGIRI, S. K. et al. **VPPET: Virtual platform power and energy estimation tool for heterogeneous MPSoC based FPGA platforms**2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS). **Anais**...Mallorca: IEEE, 2014b

ROTH, C. et al. **HLA-based simulation environment for distributed SystemC simulation**, Conference on Simulation. **Anais**...2011a.

ROTH, C. et al. **Modular Framework for Multi-level Multi-device MPSoC Simulation**, 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. **Anais**...IEEE, maio 2011b.

ROTH, C.; SANDER, O.; BECKER, J. **Flexible and efficient co-simulation of networked embedded devices**, Proceedings of the 24th symposium on Integrated circuits and systems design - SBCCI '11. **Anais**...New York, New York, USA: ACM Press, 2011

SAMSUNG. **Samsung's VIP Design Methodology Reduces SoC Design Time Up to 40 Percent**. Disponível em: <http://www.samsung.com/global/business/semiconductor/news-events/press-releases/detail?newsId=4299>. Acesso em: 29 jun. 2015.

SCHMIDT, D. C. Model-Driven Engineering. **IEEE Computer**, p. 25–31, 2006.

SCHOLLII. **Concept of lookahead in HLA**. Disponível em: <https://schollii2.wordpress.com/2008/01/16/concept-of-lookahead-in-hla/>. Acesso em: 20 ago. 2015.

SCHÜRMANS, S. et al. **Creation of ESL power models for communication architectures using automatic calibration**, Proceedings of the 50th Annual Design Automation Conference on - DAC '13. **Anais**...New York, New York, USA: ACM Press, 2013

SILVEIRA, G. S. **Uma Abordagem para Verificação Funcional no Nível de Sistema de Circuitos Digitais que empregam a técnica de Power Gating**. Tese (Doutorado) - Universidade Federal de Campina Grande, Campina Grande, Paraíba, 2011.

SINGH, A. K.; DAS, A.; KUMAR, A. **RAPIDITAS: RAPId Design-Space-Exploration Incorporating Trace-Based Analysis and Simulation**2013 Euromicro Conference on Digital System Design. **Anais**...Los Alamitos: IEEE, set. 2013

SPEAR, C. **SystemVerilog for Verification - A guide to Learing the Testbech Language Reatures**. Second ed.New York: Springer Science, 2008.

SYNOPSYS. **OpenVera**. Disponível em: <http://www.open-vera.com/>. Acesso em: 3 jun. 2015.

TALARICO, C. et al. A New Framework for Power Estimation of Embedded Systems. **IEEE Computer**, v. 38, p. 71–78, 2005.

TIWARI, V.; MALIK, S.; WOLFE, A. Power analysis of embedded software: A first step towards software power minimization. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 2, n. 4, p. 437–445, 1994.

TRABELSI, C. et al. A Model-Driven Approach for Hybrid Power Estimation in Embedded Systems Design. **EURASIP Journal on Embedded Systems**, v. 2011, n. 1, p. 569031, 2011.

TRČKA, N. et al. **Integrated model-driven design-space exploration for embedded systems**, Proceedings - 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2011. **Anais**...2011

UNIVERSITY OF CAMPINAS. **MPSoCBench - Benchmark Suite**. Disponível em: <http://www.archc.org/benchs/mpsocbench/howtouse.html>. Acesso em: 3 jul. 2015a.

UNIVERSITY OF CAMPINAS. **Architecture Description Language - ArchC**. Disponível em: <http://www.archc.org/>. Acesso em: 2 jul. 2015b.

VARMA, S.; SOBEY, K.; CAMPBELL, C. E.; KUO, S.-M. **Low power architecture and design techniques for mobile handset LSI MedityTM M2.** [S.l.]: IEEE, 2008. v. 48p. 748-753

VECE, G. B.; CONTI, M. **Power estimation in embedded systems within a SystemC-based design context: The PKtool environment**, 2009 Seventh Workshop on Intelligent solutions in Embedded Systems. Anais...2009.

VISWANATH, V.; VASUDEVAN, S.; ABRAHAM, J. A. Dedicated Rewriting: Automatic Verification of Low Power Transformations in RTL. **2009 22nd International Conference on VLSI Design**, p. 77-82, jan 2009.

WEST TEAM. **West Team - Gaspard**. Disponível em: <http://www.lifl.fr/west/gaspard/>. Acesso em: 1 jul. 2015.

XANTHOS, S.; CHATZIGEORGIOU, A.; STEPHANIDES, G. Energy estimation with systemC: a programmer's perspective. **7th WSEAS International**, 2003.

YEAP, G. Probabilistic Power Analysis. In: **Practical Low Power Digital VLSI Design**. New York: Springer US, 1998. p. 59–83.

ZHANG, P. **Advanced Industrial Control Technology**. 1st. ed. Singapore: Elsevier Science, 2010.

Anexo I

Aqui serão apresentados dois registros de execução da simulação. O Registro 1 se refere ao registro da execução da simulação demonstrando a sincronização entre o Federado MPSoC e o Federado Ptolemy. Este registro contempla a execução da simulação para a aplicação SHA com mensagem de 8 *bytes* sem utilizar o conjunto de métodos e classes que define a parte de transmissão de dados de consumo de energia da abordagem PowerHLA.

O Registro 2 contempla a execução da simulação para a aplicação SHA com mensagem de 8 *bytes*, utilizando, adicionalmente, o conjunto de métodos e classes que define a parte de transmissão de dados de consumo de energia da abordagem PowerHLA.

Registro 1 - Registro da execução da simulação para a aplicação SHA com mensagem de 8 *bytes* sem coleta de energia.

```
                  SystemC 2.3.0-ASI --- Sep  3 2014 20:35:55
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Discoverd Object: handle=1, classHandle=3, name=HLAObject_1
Discoverd Object: handle=2, classHandle=3, name=HLAObject_2

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=13, attributeCount=5
, HLA Time =13
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =65

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=14, attributeCount=5
, HLA Time =14
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =65

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=14, attributeCount=5
, HLA Time =14
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =66

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=15, attributeCount=5
```

```
, HLA Time =15
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =66

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=15, attributeCount=5
, HLA Time =15
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =67

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=16, attributeCount=5
, HLA Time =16
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =67

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=16, attributeCount=5
, HLA Time =16
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =68

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=17, attributeCount=5
, HLA Time =17
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =68

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=17, attributeCount=5
, HLA Time =17
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =69

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=18, attributeCount=5
, HLA Time =18
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =69

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=18, attributeCount=5
, HLA Time =18
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =70
```

```
Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=19, attributeCount=5
, HLA Time =19
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =70

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=19, attributeCount=5
, HLA Time =19
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =71

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=20, attributeCount=5
, HLA Time =20
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =71

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=20, attributeCount=5
, HLA Time =20
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Byte Sent Value =72

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=21, attributeCount=5
, HLA Time =21
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =72
```

Registro 2- Registro da execução da simulação para a aplicação SHA com mensagem de 8 *bytes* com coleta de energia.

```
                  SystemC 2.3.0-ASI --- Sep  3 2014 20:35:55
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Discoverd Object: handle=1, classHandle=3, name=HLAObject_1
Discoverd Object: handle=2, classHandle=3, name=HLAObject_2

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=1, attributeCount=5
, HLA Time =1
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
```

```
, Energy Value =0.000002234507212 J
, Simulated Time = 0.000013 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=2, attributeCount=5
, HLA Time =2
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002092470191 J
, Simulated Time = 0.000026 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=3, attributeCount=5
, HLA Time =3
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002069550490 J
, Simulated Time = 0.000039 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=4, attributeCount=5
, HLA Time =4
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002061571183 J
, Simulated Time = 0.000052 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=5, attributeCount=5
, HLA Time =5
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002062960915 J
, Simulated Time = 0.000065 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=6, attributeCount=5
, HLA Time =6
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002062406917 J
, Simulated Time = 0.000078 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=7, attributeCount=5
, HLA Time =7
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002063699306 J
, Simulated Time = 0.000091 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=8, attributeCount=5
```

```
, HLA Time =8
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002061090432 J
, Simulated Time = 0.000104 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=9, attributeCount=5
, HLA Time =9
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002062008457 J
, Simulated Time = 0.000117 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=10, attributeCount=5
, HLA Time =10
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059274788 J
, Simulated Time = 0.000130 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=11, attributeCount=5
, HLA Time =11
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059389178 J
, Simulated Time = 0.000143 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=12, attributeCount=5
, HLA Time =12
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059847784 J
, Simulated Time = 0.000156 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=13, attributeCount=5
, HLA Time =13
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.0964816129694
, Byte Sent Value =65

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=13, attributeCount=5
, HLA Time =13
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059320777 J
, Simulated Time = 0.000169 s
```

```
Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=14, attributeCount=5
, HLA Time =14
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =65

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=14, attributeCount=5
, HLA Time =14
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058655092 J
, Simulated Time = 0.000182 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=14, attributeCount=5
, HLA Time =14
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.0934579102148
, Byte Sent Value =66

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=15, attributeCount=5
, HLA Time =15
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =66

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=15, attributeCount=5
, HLA Time =15
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002061119454 J
, Simulated Time = 0.000195 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=15, attributeCount=5
, HLA Time =15
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.0992625216226
, Byte Sent Value =67

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=16, attributeCount=5
, HLA Time =16
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =67
```

```
Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=16, attributeCount=5
, HLA Time =16
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058152633 J
, Simulated Time = 0.000208 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=16, attributeCount=5
, HLA Time =16
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.1062713765743
, Byte Sent Value =68

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=17, attributeCount=5
, HLA Time =17
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.1027926158215
, Byte Sent Value =69

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=17, attributeCount=5
, HLA Time =17
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =68

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=17, attributeCount=5
, HLA Time =17
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059914220 J
, Simulated Time = 0.000221 s

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=18, attributeCount=5
, HLA Time =18
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =69

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=18, attributeCount=5
, HLA Time =18
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059913816 J
```

```
, Simulated Time = 0.000234 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=18, attributeCount=5
, HLA Time =18
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.0986519729679
, Byte Sent Value =70

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=19, attributeCount=5
, HLA Time =19
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =70

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=19, attributeCount=5
, HLA Time =19
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002060212092 J
, Simulated Time = 0.000247 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=19, attributeCount=5
, HLA Time =19
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
, Energy Value =0.0950457738697
, Byte Sent Value =71

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=20, attributeCount=5
, HLA Time =20
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =71

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=20, attributeCount=5
, HLA Time =20
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058437171 J
, Simulated Time = 0.000260 s

Reflection Received: object=2, tag=PtolemyDataObjectAttributes,
time=20, attributeCount=5
, HLA Time =20
, Federate Name =PtolemyFederate
, Component Name =MessageGenerator
```

```
, Energy Value =0.1030694145579
, Byte Sent Value =72

Reflection Received: object=1, tag=MPSoCDataObjectAttributes,
time=21, attributeCount=5
, HLA Time =21
, Federate Name =MPSoCFederate
, Component Name =TLMmemory
, Byte Received Value =72

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=21, attributeCount=5
, HLA Time =21
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002062194081 J
, Simulated Time = 0.000273 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=22, attributeCount=5
, HLA Time =22
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002057544200 J
, Simulated Time = 0.000286 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=23, attributeCount=5
, HLA Time =23
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002063499569 J
, Simulated Time = 0.000299 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=24, attributeCount=5
, HLA Time =24
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059951717 J
, Simulated Time = 0.000312 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=25, attributeCount=5
, HLA Time =25
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002060056662 J
, Simulated Time = 0.000325 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=26, attributeCount=5
, HLA Time =26
```

```
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002055791512 J
, Simulated Time = 0.000338 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=27, attributeCount=5
, HLA Time =27
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002056691444 J
, Simulated Time = 0.000351 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=28, attributeCount=5
, HLA Time =28
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002059562001 J
, Simulated Time = 0.000364 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=29, attributeCount=5
, HLA Time =29
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002055305724 J
, Simulated Time = 0.000377 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=30, attributeCount=5
, HLA Time =30
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002057844164 J
, Simulated Time = 0.000390 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=31, attributeCount=5
, HLA Time =31
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002057522584 J
, Simulated Time = 0.000403 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=32, attributeCount=5
, HLA Time =32
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058140797 J
, Simulated Time = 0.000416 s
```

```
Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=33, attributeCount=5
, HLA Time =33
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058433867 J
, Simulated Time = 0.000429 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=34, attributeCount=5
, HLA Time =34
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058718526 J
, Simulated Time = 0.000442 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=35, attributeCount=5
, HLA Time =35
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058521309 J
, Simulated Time = 0.000455 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=36, attributeCount=5
, HLA Time =36
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002058795782 J
, Simulated Time = 0.000468 s

Reflection Received: object=1, tag=MPSoCPowerObjectAttributes,
time=37, attributeCount=5
, HLA Time =37
, Federate Name =MPSoCFederate
, Component Name =ProcessorCore
, Energy Value =0.000002057294432 J
, Simulated Time = 0.000481 s
```

Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations

Helder F. A. Oliveira
Fed. Univ. of Campina Grande (UFCG)
Campina Grande, PB, Brazil
helder@copin.ufcg.edu.br

Alisson V. Brito
Fed. Univ. of Paraíba (UFPB)
João Pessoa, PB, Brazil
alisson@ci.ufpb.br

Elmar U. K. Melcher
Fed. Univ. of Campina Grande (UFCG)
Campina Grande, PB, Brazil
elmar@dsc.ufcg.edu.br

Harald Bucher
Karlsruhe Inst. of Technology (KIT)
Karlsruhe, Germany
bucher@kit.edu

Joseana M. F. R. Araújo
Fed. Univ. of Campina Grande (UFCG)
Campina Grande, PB, Brazil
joseana@dsc.ufcg.edu.br

Liana Duenha
Fed. Univ. of Mato Grosso do Sul (UFMS)
Campo Grande, MS, Brazil
lianaduenha@facom.ufms.br

ABSTRACT

Power consumption is a big challenge in chip design. Decisions taken in early design phases have large impact on the power consumption. Generally, simulation-based Design Space Exploration (DSE) is computationally costly for large problems due the size of design space. Simulate the possible scenarios in a distributed fashion can decrease the time to find efficient solutions. In this paper we describe an approach using HLA (High level Architecture) that allows to distribute and simulate different scenarios of Electronic System Level (ESL) models and/or Register Transfer Level (RTL) models for collecting and grouping power estimation data in a centralized manner. These models can be described in C++, SystemC, SystemVerilog or Verilog. As case study, we use a benchmark composed of a scalable set of MPSoCs described in C++/SystemC. We also use a small project in SystemVerilog/Verilog to validate the power estimation data collecting from models described in these languages through the approach. The experimental results show that the proposed method can distribute and simulate different scenarios of Electronic System Level (ESL) models as well as Register Transfer Level (RTL) models and provide a unified view of power estimation data.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Simulation*; I.6.8 [**Simulation and modeling**]: Types of Simulation—*Distributed, Combined*

General Terms

Design, Experimentation, Measurement

SBCCI '15, August 31 - September 04, 2015, Salvador, Brazil
 ISBN 978-1-4503-3763-2/15/08 $15.00
DOI: http://dx.doi.org/10.1145/2800986.2801023.

Keywords

Distributed, ESL, HLA, Power Estimation, RTL, Simulation

1. INTRODUCTION

With the growing complexity of chip design, power consumption becomes a challenge. This complexity and the time-to-market pressure require power estimation at higher level abstraction, such as ESL (Electronic System Level), for a system. The power issue should be considered by the designers as soon as possible in the design phases. Much of the work about power estimation at ESL focus the trade-off between accuracy and computing cost of the approaches [13][11][3][7][17]. There are several approaches for power estimation, at many abstraction levels.

In order to evaluate the power consumption, usually the designer has to evaluate other associated aspects, e.g., performance. This leaves a broad range of design options, among which, he needs to choose some that better fits with the project requirements. In a parametrized system-on-a-chip (SoC), e.g., a MPSoC, the configuration parameters (e.g., number of cores, processor architecture, interconnection devices, etc.) must be appropriately chosen to find the best trade-off in terms of the selected aspects (e.g., energy and performance) [2][14]. This process is known as Design Space Exploration (DSE). Two main approaches to DSE are distinguished by [10]: analytical and simulation-based DSE. The analytical basically prunes the design space (all the possible design solutions) according some algorithm based in some heuristic. The simulation-based DSE is usually executed after this design space reduction in order to find the best solutions from this subset. Simulate efficiently the pruned design space can decrease the simulation time, consequently reducing the time to find efficient solutions. In this way, the developed approach focus on distributing and simulating different scenarios for a given design, collecting and grouping power estimation data in a centralized manner using a generic simulation backbone based on the High Level Architecture (HLA) [1]. The scenarios can vary in terms of abstraction level (ESL or RTL), parameters, or even in terms of power estimation approaches. The HLA IEEE standard usage promotes the flexibility to connect other ar-

bitrary simulators on the approach. The models simulated can be described in C++, SystemC, SystemVerilog or Verilog, and can be extended to other languages.

As case study, we use a benchmark composed of a scalable set of MPSoCs described in C++/SystemC to build some scenarios. This benchmark is called MPSoCBench [4]. We also use a small project in SystemVerilog/Verilog (a Differential Pulse-Code Modulation) to validate the power estimation data collecting from models described in these languages through the approach. Based on experimental results we show the operation and the ease of use of the approach.

The remainder of this paper is organized as follows: In the Section II some fundamentals are provided. In Section III the related work is discussed. Section IV describes the proposed approach. Section V presents the experimental results. Finally, Section VI concludes the paper.

2. FUNDAMENTALS

2.1 High level Architecture

The HLA is an IEEE standard [1] since 2000. It was originally defined by the Defense Modeling and Simulation Office (DMSO) for the U.S. Department of Defense. Its original field of application are military training simulations. The HLA is a generic software architecture combining all the components necessary for *Parallel Discrete Event Simulation* (PDES) [19]. In HLA terminology the logical representation of an interconnection of different simulators is called a *federation* and includes multiple simulators called *federates*. Federates connect via *ambassadors* to a *runtime infrastructure* (RTI). The RTI implements services defined by the HLA standard like time management or data distribution management. A RTI can possibly run several independent logical federations in parallel. Also part of the HLA standard is the so called *Object Model Template* (OMT) which defines the format and syntax of HLA object models including object/interaction classes attributes, parameters and datatypes but not their content. The OMT allows to define Federation Object Models (FOM) and Simulation Object Models (SOM). The FOM contains properties of a whole federation. The SOM contains properties of a single federate.

2.2 MPSoCBench

MPSoCBench [4] is a simulation toolset composed of a scalable set of MPSoCs useful for the development and high level evaluation of new tools, methodologies, software, and hardware components. This tool provides a complete open source simulation infrastructure including scalable hardware and software components, with easy instrumentation and fast simulation at different abstraction levels.

The tool set supports four ISAs in many configurable and scalable MPSoC platforms with up to 64 cores, with different interconnections that define different simulation abstraction levels. The user provides parameters to create MPSoC simulators and the script generates appropriate code for processors, interconnections, caches, memories, and IPs. These components are compiled to create one or more MPSoC simulators that will be stored in specific folders. The set of applications are compiled with the appropriate cross-compilers, and the executable files are stored together with the simulators. After executing each simulation, a proof of correctness is performed evaluating the application output files. Figure 1 illustrates a MPSoC simulation process using MPSoCBench.

2.3 Power Models

The MPSoCBench also includes a power consumption model for SPARC and MIPS processors using PowerSC [11] and acPower [5] implemented based on the open source LEON3 and PLASMA RTL models. The power models are based on different FPGAs and ASICs technologies. Some key usages are power consumption evaluation during program execution, detecting power bottlenecks, and comparison among different architectures.

The characterization process is based on the Tiwari [21] method, which shows that the average energy consumption of a characterization program with a loop of several instructions with the same *opcode* but randomized operands can be used as a power per instruction approximation. In this way, a consistent measure of the average power per instruction can be obtained. By repeating this process to all different instructions, the energy characterization per instruction is obtained, enabling to estimate power using processor frequency and instruction per cycles (IPC) information. This method provides an instruction-based energy consumption analysis and can be used in order to characterize the entire processor instruction set.

The characterization step is executed only once for each processor and requires an RTL synthesis and simulation tool like Synopsys Design Compiler, Xilinx ISE or Altera Quartus. After synthesis, it is simulated the back-annotated version, collect switching activity and use a Power Estimation flow like Xilinx XPower, Altera PowerPlay, or Synopsys Power Compiler. The average power consumption per instruction is collected and one power table is created for each hardware configuration (technology and operation frequency).

The MPSoCBench integrated with PowerSC and acPower read the aforementioned power tables and generate power consumption characteristics of the virtual platform model. After simulation, MPSoCBench generates a report with power consumption by each core and tables the stores the dynamic energy consumption. The user can choose among different characterization technologies and frequency instrumentation for Xilinx (Spartan and Virtex), Altera (Cyclone and Stratix) FPGAs, and OpenPDK 45nm ASIC.

3. RELATED WORK

DSE has become a high and active research field [6]. Many techniques propose hybrid approaches for DSE. In [6] is proposed design flow integrating analytical and simulation-based design space exploration in order to find efficient design solutions without sacrificing timing guarantees. In [16] is presented an approach to interleave analytical and simulation-based DSE to reduce the number of simulation required during the DSE phase. The results are compared with pure simulation-based DSE, showing solutions with similar quality however consuming only a fraction of the execution time. The work presented in [12] developed a high level performance model allowing a parameter sweep of the large design space quickly. The sweep provides a reduced number of promising designs which can be simulated to arrive at the most energy-efficient design. The authors in [20] propose a methodology that executes analytical estimations to compute the throughput of different possible

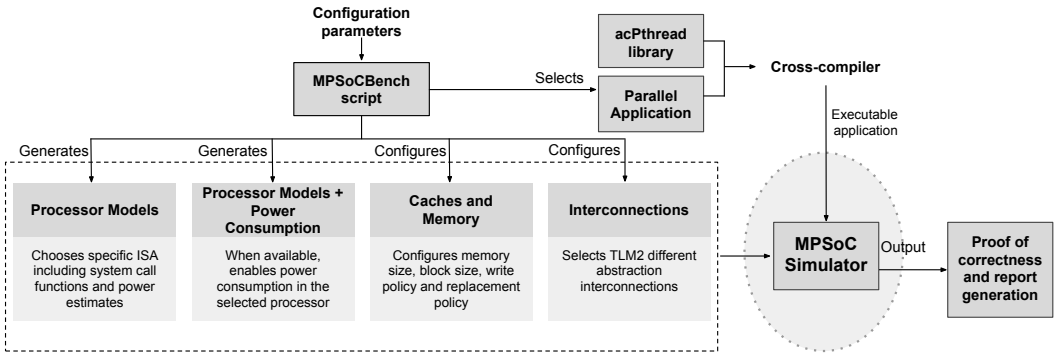


Figure 1: Development flow of a MPSoC simulator using MPSoCBench

mappings, then simulation is performed only for the best mapping, limiting the number of simulations. In [9] is presented a system level DSE infrastructure implemented in C++ to provide a single, common, and modular framework for system level DSE experiments. It allows to incorporate different (existing) system level simulation tools as well as different combinations of search strategies by means of a plug-in mechanism.

However, to the best of our knowledge, none of the aforementioned works use HLA as interconnection backbone to distribute and simulate the design space or part of it. The HLA is one of promising standardized ways to achieve concurrent simulation. Our approach uses HLA as base to execute simulations concurrently, collecting and organizing all power estimation data step-by-step in a predefined time interval.

In [19], the HLA is successfully used to execute distributed simulation of processing nodes from a SystemC MPSoC model. The processing nodes are described in a loosely-timed coding style using the OSCI TLM2.0 blocking transport interface. The communication channels (FIFOs) between the processing nodes are cut in two parts to work over network using HLA. In the work, each node/sub-module into the experiments is simulated by a separate SystemC kernel. In [18] the authors continued the former work, presenting a simulation framework which aims the simulation of multiple MPSoC systems on different abstraction levels while using several simulation platforms concurrently.

4. DEVELOPED APPROACH

The developed approach shown in Figure 2 can be divided in seven main elements, described in next subsections. All the elements were developed using C++. One of them makes use of SystemC.

4.1 Monitored Component

A Monitored Component represents a piece of code, usually a functional module, that the designer wants to monitor. The Sender can have various Monitored Component instances.

4.2 Sender

The Sender was developed using C++ and SystemC. It has a SC_THREAD process responsible to send the estimated energy data accumulated by each Monitored

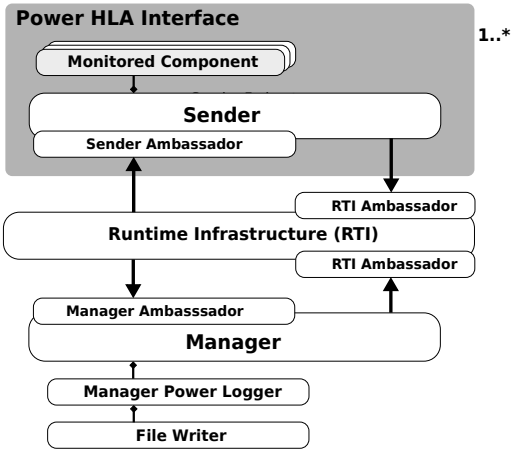


Figure 2: The developed approach

Component instance according with a predefined time interval. When the predefined interval is reached, the Sender takes the simulation control, sends all the accumulated data from each Monitored Component instance to the Manager Federate, sets to zero all the respective energy related variables and gives back the control to the main simulation process by calling the statement *wait(POWER_COLLECTING_INTERVAL, INTERVAL_UNIT)*. The POWER_COLLECTING_INTERVAL is the predefined time interval which the Sender must send the power collected data to the Manager Federate and the INTERVAL_UNIT is the time interval unit (e.g., microseconds, milliseconds, seconds, etc.). Any federate which contains the Sender will be called Sender Federate. The federation can have many Sender Federates, since each Sender Federate represents a distributed simulation (i.e., a federate).

4.3 Runtime Infrastructure

The RTI (Runtime Infrastructure) is the software that provides common interface services during a High Level Architecture (HLA) federation execution for synchronization and data exchange [1]. We didn't implement the RTI. The RTI used in our experiments is an Open Source HLA RTI called CERTI [15].

4.4 Ambassadors

Communication from the RTI to a federate is established via the RTI Ambassador. The opposite is made by the Federate Ambassadors: Sender Ambassador and Manager Ambassador. Each Sender instance will have a respective Sender Ambassador instance. In contrast, the Manager Ambassador will be unique into the federation because the Manager Federate is unique.

4.5 Manager

The Manager is unique inside the federation. The federate which contains the Manager will be called Manager Federate. The Manager Federate receives and organizes the power estimation data from all Sender Federates that compose the federation, providing a unified view of this data. It is also responsible for creating the federation and wait the signal to start the simulations when all Sender Federates join into the federation.

4.6 Manager Power Logger

The Manager Power Logger organizes the power estimation data received by the Manager Ambassador. The received power estimation data comes from all Sender Federates.

4.7 File Writer

The CSV (comma separated values) files are generated by the File Writer class. The CSV files are the outputs generated by the approach. For each Sender Federate, the File Writer class generates a CSV file which contains the accumulated energy from its respective monitored components during each POWER_COLLECTING_INTERVAL over the simulated time line. Graphics can be easily generated using the CSV files by applications such as OpenOffice Calc and Microsoft Excel.

4.8 Power HLA Interface

This element provides all the necessary methods to collect power/energy estimation data from the simulation. It is also responsible for creating its respective Sender Federate and deal with HLA initialization aspects. The Power HLA Interface was implemented according with the singleton design pattern. The singleton ensures a class has only one instance, and provides a global point of accessing it. This design pattern was used in order to provide an easy way to call the Power HLA Interface methods from any place of the code, without the need to instantiate the object every time it is necessary to call any of its methods. In order to allow the approach simulates a set of SystemVerilog/Verilog models, as well as, simulates C++/SystemC and SystemVerilog/Verilog models together, two extra classes were created, one C++ and other SystemVerilog. These classes enable the calling of methods from the Power HLA Interface by SystemVerilog/Verilog models via DPI (Direct Programming Interface). The manner which the Power HLA Interface was implemented facilitates its extension to work with other languages and simulators. In our federation we have a Power HLA Interface instance for each Sender Federate.

4.9 Other Relevant Details

The developed approach has a file at Sender and Manager code side, called defines.h, which the designer must define a set of constants in order to configure the simulation environment. The environment comprises of a side to deal with power and a side to deal with HLA. Some of most important constants from Sender Federate to deal with power are shown below:

- POWER_COLLECTING_INTERVAL: The time interval which the Sender Federate must send the power collected data to the Manager Federate. The unit is defined by the INTERVAL_UNIT constant.
- INTERVAL_UNIT: The time interval unit. It can be, for example, SC_US to define the time interval unit as microsecond.
- NUMBER_OF_MONITORED_COMPONENTS: The total number of monitored components (it can be a module, a set of modules, a SoC, etc.).

We also have constants to deal with the HLA:

- SENDER_FEDERATE_NAME: The Sender Federate name. This name should be unique.
- SENDER_FEDERATE_LOOKAHEAD: The federate Lookahead. See [1] for details.
- SENDER_FEDERATE_TIME_STEP: The federate Time Step. See [1] for details.

The Manager Federate has only the constants to deal with the HLA. These constants are similar with the Sender Federate constants.

5. EXPERIMENTAL RESULTS

In this section, we present a set of experimental results using the proposed approach through three case studies. The first and second case studies use scenarios built by the MPSoCBench platform. The third case study uses a DPCM (Differential Pulse-Code Modulation) RTL model described in SystemVerilog/Verilog. All the case studies were simulated in a distributed fashion using the approach.

For the first case study we defined six scenarios, which include a processor, an interconnection device, and a software to be executed. The built scenarios vary in terms of processor type and number of cores. The software selected to be executed for all its scenarios was the Susan Edges [8]. The Susan Edges algorithm identifies points in a digital image at which the image brightness changes sharply, i.e., has discontinuities.

We also had to decide between three interconnection devices: Router, NoC-LT and NoC-AT. As our focus was to measure the power consumption from the processors cores, the best choice would be the fastest interconnection device. The Router component was chosen because it is the fastest and simplest interconnection device if compared with the Noc-AT and NoC-LT. The Router component is a set of TLM channels connecting processors to the memory and lock devices, followed by communication interfaces with each instantiated processor [4]. The NoC-LT is a mesh based NoC in a TLM loosely timed (LT) abstraction level. The NoC-AT is a mesh based NoC in a TLM approximately timed (AT) that uses sockets, forward and backward transport interfaces. The only processors power characterized between the options given by MPSoCBench are the MIPS and SPARC. Consequently, both were chosen to evaluate our approach.

For the first case study, the approach will simulate, in a distributed way, scenarios with 2, 4 and 8 cores for MIPS and SPARC processors, running the Susan Edges software, using the Router as interconnection device. The Susan Edges software was chosen because it is between the most time consuming provided by the MPSoCBench platform (see Table III in [4]) which fits with our scenarios configuration.

We instrumented the code of MIPS and SPARC processors in order to monitor the cores during the scenarios simulation. The instrumentation is minimally invasive, since the approach allows to monitor each core by calling a single line of code inside them. The code line calls a method to accumulate energy for a given current power average value and a current time in seconds. The current time value is subtracted of the last sampled time value and so the energy consumed during this interval can be calculated. The aforementioned line is represented in the algorithm shown at Algorithm 1 (line 4). This algorithm represents the instrumentation code inserted into the MIPS and SPARC cores code. For each 1000 instructions executed (Algorithm 1, line 2) we call the method to accumulate energy. In order to call this method we must get the unique instance of the Power HLA Interface (Algorithm 1, line 3). Note that we must create the Power HLA Interface object before call its static methods ("**Require**" shown at Algorithm 1).

After the mentioned code instrumentation, all the six scenarios could be easily simulated in a distributed fashion by the approach. Each scenario is contained in its respective Sender Federate. The Manager Federate must be the first federate to be initiated. After initiating the Manager Federate, it will wait for all Sender Federates join the federation before starting the simulation. As we distributed the Sender Federates in pairs in three different machines with same configuration, we could reduce the total simulation time roughly to the time of slowest simulated scenario, which was 335 s. The slowest scenario is composed of a SPARC processor with 8 cores. Simulating all these MPSoCBench scenarios in a single machine, without apply our approach, give us a total simulation time of 386 s. Note that the simulation time using our approach was only 13.2% less if compared with the former. However, using more complex scenarios we will probably obtain better results. Furthermore, pure MPSoCBench scenarios do not allow to distribute the simulations in different machines collecting the estimated energy consumption in a centralized manner, step by step, during the simulation. It is important do not confuse "simulation time" with "simulated time". The simulation time is the wall clock time elapsed for a machine finish the simulation. This time can vary according with the machine configuration (e.g., processor frequency, memory size, memory frequency, etc.). The simulated time is the real clock time spent by the CPU to process the simulation. We performed the experiments on machines Intel i5 860 2.9 GHz, with 4 GB RAM, running Ubuntu Linux 13.10 64 bits operating system, gcc 4.8.1 and CERTI 3.4.3.

Algorithm 1 Instrumentation algorithm for MIPS and SPARC processors

Require: The Power HLA Interface object has to be created before calling its static methods
1: For each executed instruction, increment the instruction counter by 1
2: **if** instruction counter is 1000 **then**
3: Get Power HLA Interface unique instance
4: The Power HLA Interface unique instance accumulates energy for the given current power average value and current time
5: Set instruction counter to zero
6: **end if**

For all Sender Federates, the power collecting interval was defined as 1 ms. The fist scenario considered a MIPS with 2 cores. The results are shown in TABLE 1, line 1. The total simulated time spent by this scenario was 238 ms. The total energy consumed by the processor during the simulation was 71.67 mJ, where 35.50 mJ was from Core 1 and 36.17 mJ was from Core 2.

The second scenario (TABLE 1, line 2) was defined as a SPARC with 2 cores. The total simulated time spent by this scenario was 555 ms. The total energy consumed by the processor during the simulation was 15.97 mJ, where 8.05 mJ was from Core 1 and 7.92 mJ was from Core 2. Compared with the first scenario, the second scenario consumes only 22.28% from the total consumed by the former. However, the fist scenario consumes less CPU time to process the same algorithm, around 42.88% from the total consumed by the second scenario.

Analyzing these two simulation scenarios give us a direction that, if power consumption is the main requirement, probably we will choose the SPARC processor. Otherwise, if it aims to decrease the total CPU time spent to process the algorithm, the MIPS architecture will be a better choice. However, just these two simulations are not enough to help us to decide which one is better. Thus, we also simulated scenarios with 4 and 8 cores, which are shown next.

The scenario 3 (TABLE 1, line 3) consisted of a MIPS processor with 4 cores. The total simulated time spent by this scenario was 137 ms. We had a reduction of 42.43% of consumed CPU time when compared with scenario 1 just doubling the cores. However, the total energy consumption was 82.31 mJ, what means an increase of 14.84%. Notice that this scenario executes faster than second scenario, around 4x, but the energy consumption is around 5x higher.

The SPARC with 4 cores scenario (TABLE 1, line 4) used 324 ms of CPU time and consumed a total of 18.86 mJ. If compared with scenario 2, doubling the core numbers give us a reduction of 41.62% of consumed CPU time and an increase of 18.1% in terms of energy consumption.

The scenario 5 comprises a MIPS processor with 8 cores (TABLE 1, line 5). The total simulated time spent by this scenario was 80 ms and the total energy consumption was measured as 96.11 mJ. The scenario 6 (TABLE 1, line 6), a SPARC processor with 8 cores, spent 263 ms of CPU time and 31.14 mJ. A summary is shown at Figure 3. The results obtained in these four last simulations corroborate the previous analysis for the architecture choice.

As second case study, the same experiment was executed by changing the software for FFT (Fast Fourier Transform). The results were similar (Figure 4).

The third case study uses the approach during a functional verification of a DPCM (Differential Pulse-Code Modulation) RTL model described in SystemVerilog/Verilog. A DPCM converts a sequence of digital samples from a signal to a sequence of differences between consecutive samples. The differences are then sutured to limit their representation. This case study was used to validate the collecting of energy estimation data from models described in SystemVerilog/Verilog through the approach. We instru-

Table 1: Simulation Summary

Scenario	Proc. Arch.	N. of Cores	Energy (mJ)	Simulated Time (ms)
1	MIPS	2	71.67	238
2	SPARC	2	15.97	555
3	MIPS	4	82.31	137
4	SPARC	4	18.86	324
5	MIPS	8	96.11	80
6	SPARC	8	31.14	263

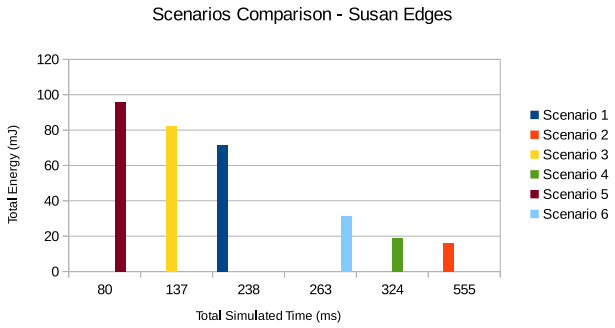


Figure 3: Scenarios Comparison for Susan Edges algorithm

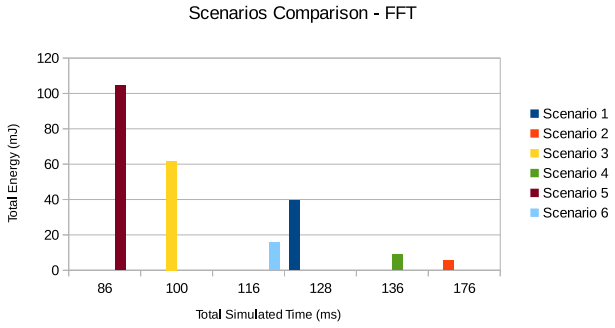


Figure 4: Scenarios Comparison for FFT algorithm

mented the DPCM code to accumulate an arbitrary value of energy each rising edge clock, by means of DPI call. The DPI call used into the DPCM code refers to a method from the Power HLA Interface. This method accumulates an energy value for a given monitored component. The arbitrary value assigned for each rising edge clock was 1 pJ. The *POWER_COLLECTING_INTERVAL* was defined as 1 ms. The total simulated time was 255 ms. The total energy consumption was 0,025 mJ. The experiment was executed using two machines, one executing the Manager Federate and the other executing the Sender Federate. The configuration machines were the same used at case study 1 and 2. To execute the functional verification we used the Open Verification Methodology (OVM) and a simulation toolset from Cadence.

As we can see, although the approach was developed using C++/SystemC, it also allows distributed simulation and power estimation data collecting from models described in SystemVerilog/Verilog, using the Power HLA Interface and its related DPI classes. It is important to note that we could mix distributed simulations of models described in ESL (C++/SystemC) and RTL (SystemVerilog/Verilog). This would be useful if the designer would like to compare, for example, the accuracy of power estimation in different abstraction levels. At the ESL, the designer could, for example, compare different power estimation approaches by their simulation time and also their accuracy. Another possible usage could be the use of RTL simulation power estimation results to create power models for ESL models.

6. CONCLUSION

In this paper we have presented and validated our approach for distribute and simulate different scenarios of Electronic System Level (ESL) models and/or Register Transfer Level (RTL) models allowing collect and group power/energy estimation data in a centralized manner through a single simulation environment based on HLA. Further research includes the evaluation of the approach using more complex scenarios. The approach simplicity and the minimally invasive instrumentation code characteristic makes it easy to use.

7. ACKNOWLEDGMENT

This work is supported by “Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)” by a postgraduate research scholarship.

8. ADDITIONAL AUTHORS

Additional author: Rodolfo Azevedo (Univ. of Campinas (Unicamp), Campinas, SP, Brazil, email: `rodolfo@ic.unicamp.br`).

9. REFERENCES

[1] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). *IEEE Std 1516.x-2010*, aug. 2010.

[2] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. A multiobjective genetic fuzzy approach for intelligent system-level exploration in parameterized vliw processor design. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1736–1743. IEEE, 2006.

[3] G. Beltrame, D. Sciuto, and C. Silvano. Multi-accuracy power and performance

transaction-level modeling. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(10):1830–1842, 2007.

[4] L. Duenha, M. Guedes, H. Almeida, M. Boy, and R. Azevedo. Mpsocbench: A toolset for mpsoc system level evaluation. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 164–171. IEEE, 2014.

[5] M. Guedes, R. Auler, L. Duenha, E. Borin, and R. Azevedo. An automatic energy consumption characterization of processors using archc. *Journal of Systems Architecture*, 59(8):603 – 614, 2013.

[6] F. Herrera and I. Sander. Combining analytical and simulation-based design space exploration for time-critical systems. In *Specification & Design Languages (FDL), 2013 Forum on*, pages 1–8. IEEE, 2013.

[7] W.-T. Hsieh, J.-C. Yeh, and S.-Y. Huang. Pac duo system power estimation at esl. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 815–820. IEEE, 2010.

[8] S. M. Z. Iqbal, Y. Liang, and H. Grahn. Parmibench-an open-source benchmark for embedded multiprocessor systems. *Computer Architecture Letters*, 9(2):45–48, 2010.

[9] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Núñez. Nasa: A generic infrastructure for system-level mp-soc design space exploration. In *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, pages 41–50. IEEE, 2010.

[10] T. Kempf, G. Ascheid, and R. Leupers. *Multiprocessor Systems on Chip: Design Space Exploration*. Springer Science & Business Media, 2011.

[11] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. Santos. An Efficient Framework for High-Level Power Exploration. In *Proceedings of the 50th Midwest Symposium on Circuits and Systems - MWSCAS 2007*, pages 1046–1049. August 2007.

[12] S. R. Kuppannagari, Y. Hu, and V. K. Prasanna. High level performance model based design space exploration for energy-efficient designs on fpgas. In *Green Computing Conference (IGCC), 2014 International*, pages 1–6. IEEE, 2014.

[13] I. Lee, H. Kim, P. Yang, S. Yoo, E.-Y. Chung, K.-M. Choi, J.-T. Kong, and S.-K. Eo. Powervip: Soc power estimation framework at transaction level. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 551–558. IEEE Press, 2006.

[14] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. Multi-processor system-on-chip design space exploration based on multi-level modeling techniques. In *Systems, Architectures, Modeling, and Simulation, 2009. SAMOS'09. International Symposium on*, pages 118–124. IEEE, 2009.

[15] E. Noulard, J.-Y. Rousselot, and P. Siron. Certi, an open source rti, why and how. In *Spring Simulation Interoperability Workshop*, pages 23–27, 2009.

[16] R. Piscitelli and A. D. Pimentel. Interleaving methods for hybrid system-level mpsoc design space exploration. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 7–14. IEEE, 2012.

[17] S. K. Rethinagiri, O. Palomar, J. Arias Moreno, O. Unsal, and A. Cristal. Vppet: Virtual platform power and energy estimation tool for heterogeneous mpsoc based fpga platforms. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*, pages 1–8. IEEE, 2014.

[18] C. Roth, G. M. Almeida, O. Sander, L. Ost, N. Hebert, G. Sassatelli, P. Benoit, L. Torres, and J. Becker. Modular framework for multi-level multi-device mpsoc simulation. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 136–142. IEEE, 2011.

[19] C. Roth, O. Sander, M. Kühnle, and J. Becker. Hla-based simulation environment for distributed systemc simulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 108–114. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[20] A. K. Singh, A. Das, and A. Kumar. Rapiditas: Rapid design-space-exploration incorporating trace-based analysis and simulation. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 836–843. IEEE, 2013.

[21] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In *IEEE Transactions on VLSI Systems, vol. 2*, page 437âĂŞ445. 1994.