FERNANDO FERREIRA PASSE

FERRAMENTAS DE ENSINO COM GRAFOS DE FLUXO DE DADOS EM TRÊS NÍVEIS DE ABSTRAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

Orientador: Ricardo dos Santos Ferreira

VIÇOSA - MINAS GERAIS
2020

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
P287f
2020

Passe, Fernando Ferreira, 1995-
Ferramentas de ensino com grafos de fluxo de dados em três níveis de abstração / Fernando Ferreira Passe. – Viçosa, MG, 2020.
67 f. : il. (algumas color.) ; 29 cm.

Orientador: Ricardo dos Santos Ferreira.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 59-67.

1. Fluxo de dados (Computadores). 2. Matrizes de portas programáveis em campo. 3. Unidade de processamento gráfico. 4. Internet das coisas. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22. ed. 004.35

FERNANDO FERREIRA PASSE

FERRAMENTAS DE ENSINO COM GRAFOS DE FLUXO DE DADOS EM TRÊS NÍVEIS DE ABSTRAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 23 de julho de 2020.

Assentimento:

Fernando Ferreira Passe
Autor

Ricardo dos Santos Ferreira
Orientador

Dedico este estudo à minha família, em especial, aos meus pais Joaquim e Ilda.

AGRADECIMENTOS

Agradeço à FAPEMIG com o apoio através do projeto "Aceleradores para Redes Reguladores de Genes" - Modalidade: "Edital 001/2018 - Demanda Universal", Processo N. : APQ-01203-18.

Agradeço que o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Aos meus pais, Joaquim e Ilda, que nunca mediram esforços para me ensinar o caminho do bem, e sempre me apoiaram em todas as etapas da minha vida. Sem vocês, eu não chegaria até aqui. Muito obrigado por tudo! O amor que sinto por vocês é incondicional.

Agradeço a minha irmã Fernanda e meu cunhado Pedro, pelo apoio e por todos os esforços feitos para que eu chegasse até aqui. Vocês foram fundamentais!

Ao meu orientador, Professor Ricardo dos Santos Ferreira, pela oportunidade de realizar este trabalho. Obrigado pela confiança e por me atender com paciência todas as vezes que bati em sua porta. Agradeço por todos os ensinamentos compartilhados de forma admirável, e por me guiar nos primeiros passos da pós-graduação. Muito obrigado por tudo!

À professora e amiga Gláucia Aparecida Soares Miranda, que me ajudou em todas as etapas da graduação e pós-graduação. Solicitei a sua ajuda inúmeras vezes, e em todas fui atendido com paciência e tranquilidade. Serei eternamente grato por toda ajuda durante a realização deste trabalho, você foi fundamental!

À minha namorada Victória Garcia Sperandio, pelo apoio, companheirismo e por estar comigo em todos os momentos. Te amo!

À Brenda de Araújo Dantas Jorge, amiga que fiz durante a graduação e embarcou comigo no sonho da pós-graduação. Obrigado por todo apoio ao longo desta caminhada. Dividimos a casa, os sonhos e as dificuldades juntos. Ter você por perto foi essencial para que eu conseguisse seguir em frente. Muito obrigado por tudo!

Aos meus amigos Vinícius Garcia Sperandio, Felippe Cathoud de Queiroz, Laura Pérez Vera, Davi Hagap Emanuel da Silva, Hector Pérez Baranda e Juan Esteban Niño Rodriguez pelo companheirismo diário no laboratório e na vida. Vocês tornaram os dias de trabalho muito mais leves e divertidos. Obrigado por me ajudarem em todos os momentos que precisei. Sou muito grato à vocês!

Tente mover o mundo - o primeiro passo será mover a si mesmo.
Platão

RESUMO

PASSE, Fernando Ferreira, M.Sc., Universidade Federal de Viçosa, julho de 2020. **Ferramentas de Ensino com Grafos de Fluxo de Dados em Três Níveis de Abstração**. Orientador: Ricardo dos Santos Ferreira.

O processamento de aplicações com grande volume de dados e alto consumo de recursos computacionais é uma realidade. Aplicações de stream de áudio, vídeo e jogos vêm exigindo uma mudança de paradigma, onde estruturas não bloqueantes com processamento paralelo são essenciais. Neste contexto, o uso de grafo de fluxos de dados na modelagem dos algoritmos vem como uma forma de tornar as aplicações escaláveis, já que é possível especificar o paralelismo de forma explícita. Além disso, os dados são processados à medida em que chegam e combinados para aproveitar o máximo de recursos do hardware. O uso de softwares como Node-RED e Digital JS auxiliam no aprendizado e na modelagem com grafos de fluxos de dados por meio interfaces gráficas amigáveis. Ademais, o uso de aceleradores de hardware por meio de FPGAs permite o desenvolvimento de implementações em hardware com estruturas paralelas. Esta dissertação tem contribuições em três eixos. Primeiro, no ensino de Internet das Coisas através da modelagem com grafos de fluxo de dados usando a ferramenta Node-Red. Segundo, no ensino de projeto de processadores com a visualização do grafo estrutural da especificação com a ferramenta Digital-JS e a linguagem de descrição de hardware Verilog, que resultou em uma nova ferramenta denominada RISCVerilog. Terceiro, na modelagem de aplicações com grafo de fluxo de dados para implementação em FPGA na nuvem com a ferramenta READY, resultando na ferramenta PLAIN.

Palavras-chave: IoT. Internet das coisas. FPGA. Grafo de fluxo de dados. CGRA. RiscV

ABSTRACT

PASSE, Fernando Ferreira, M.Sc., Universidade Federal de Viçosa, July, 2020. **Teaching Tools with Data Flow Graphs at Three Levels of Abstraction**. Advisor: Ricardo dos Santos Ferreira.

Processing applications with high computational resources usage and with high data volume is common nowadays. Audio, video and games streaming platforms demand a paradigm change where non blocking structures with parallel processing are essential. In this context, to use data flow graphs while modelling algorithms is a way to make applications scalable because it explicit allows parallelism. In addition, data is processed on the fly and they are combined to take maximum advantage of hardware resources. Applications such as Node-RED and DigitalJS help to understand data flow and its modelling via friendly interfaces. Furthermore, using hardware accelerators via FPGAs allows to develop hardware with parallel structures. This work contributes in three lines. First, it contributes with the teaching of Internet of Things combined with modelling data flow using the Node-Red tool. Second, it contributes with teaching processor design with the visualization of the structural graph of the specification with the Digital-JS tool and the hardware description language Verilog, which resulted in a new tool called RISC Verilog. Third, it contributes with modelling data flow applications to be used with FPGAs in the cloud with the READY tool, which resulted in the PLAIN tool.

Keywords: IoT. Internet of things. FPGA. Dataflow. CGRA. RiscV

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

Introdução

A utilização de grafos de fluxo de dados ou *Dataflows* permite a descrição de estruturas de software/hardware em que a comunicação, a interação e o paralelismo estão descritos de forma explícita (Johnston et al., 2004; Culler, 1986). As interligações do grafo representam a sequência de operações e/ou comunicações dos vértices que implementam uma computação.

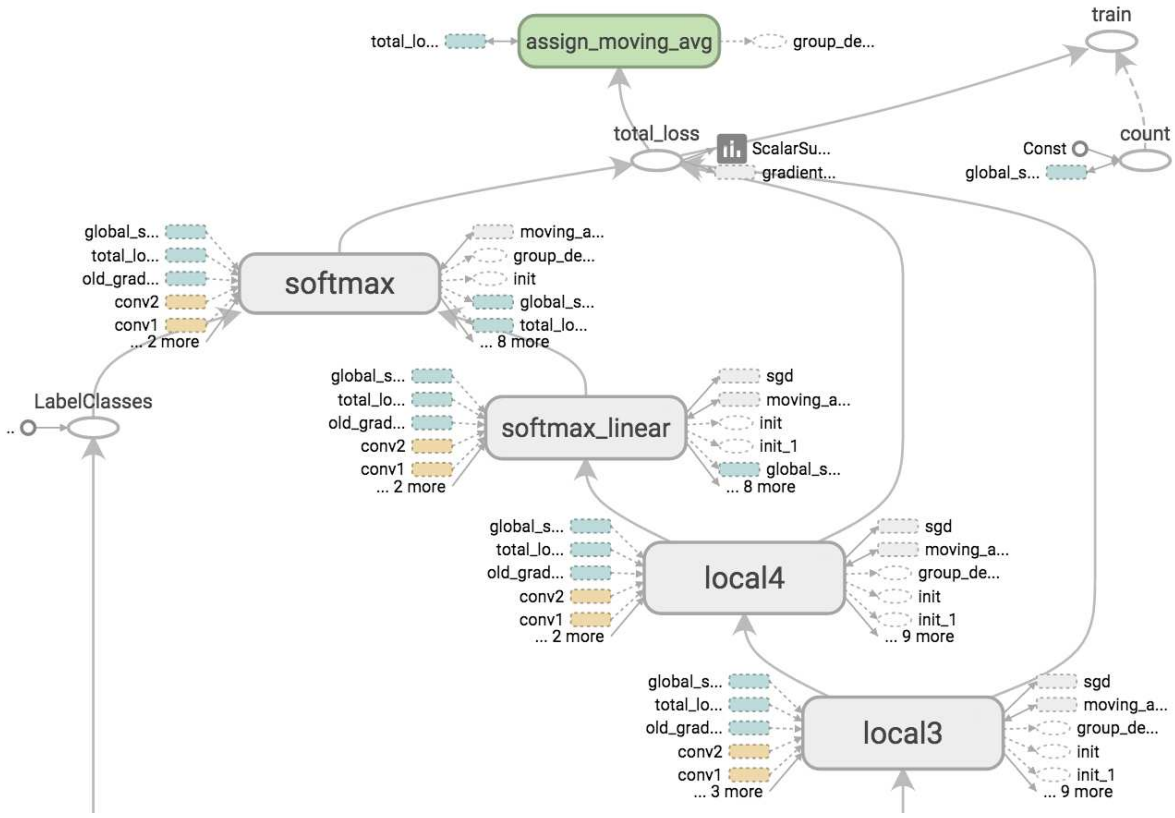


Figura 1.1: Exemplo de Grafo de Fluxo, visualização do TensorBoard (Wongsuphasawat et al., 2017), em que os nodos representam os tensores e as arestas as entradas de dados

Os grafos de fluxo são utilizados em diversos níveis de abstração. Atualmente temos vários modelos de computação utilizando grafos para visualização. Podemos citar os modelos de redes neurais com a ferramenta TensorFlow, como por exemplo o

grafo ilustrado na figura 1.1. As redes neurais com múltiplas camadas também fazem uso de grafos para uma descrição estrutural do modelo (Sharma et al., 2012).

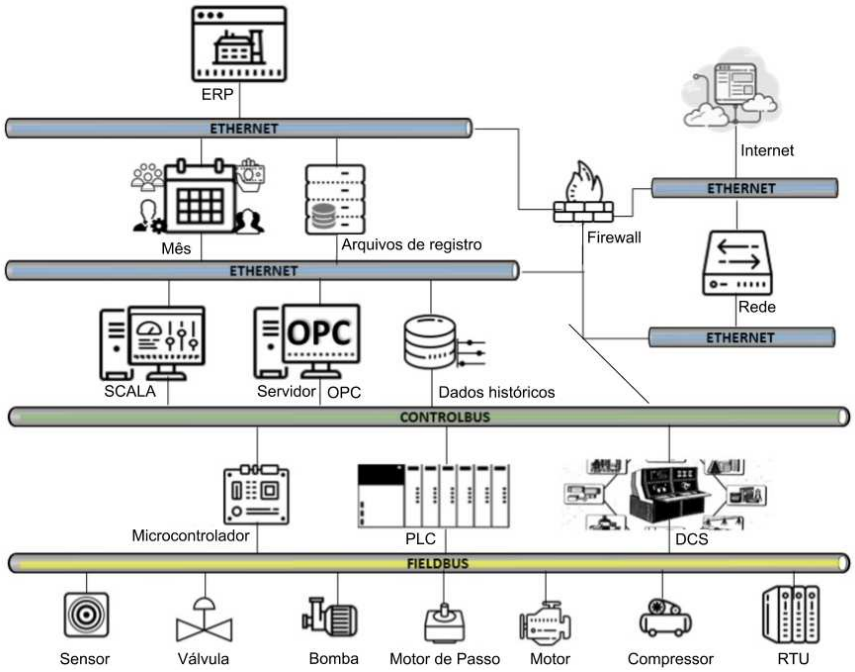


Figura 1.2: Fluxo de dados em uma indústria

Outro caso de uso são os grafos para descrição de sistemas com internet das coisas. A figura 1.2 (Veneri and Capasso, 2018) ilustra um grafo com a representação em alto nível de um sistema automatização de uma fábrica. Posteriormente, o projeto pode ser detalhado e desenvolvido em ambientes e/ou ferramentas que usam grafos como entrada em vários níveis de abstração.

Outro exemplo de uso dos grafos é a representação intermediária do código gerado por compilador. Mesmo que o código seja escrito em uma linguagem sequencial, o compilador gera o grafo de fluxo de dados e controle que permite visualizar as instruções que podem ser executadas em paralelo.

Os circuitos também são representados por grafos. Os primeiros circuitos eram projetos esquemáticos e tinham poucos componentes. Com o aumento exponencial do número de transistores, a complexidade dos circuitos inviabilizou uma descrição gráfica e surgiram linguagens de descrição de hardware como Verilog (IEEE, 1996) e VHDL (IEEE, 1988).

Mesmo com vários níveis de abstração - transistores, portas lógicas, nível de transferência de registros (RTL Register Transfer Level, etc.), as linguagens são indispensáveis para a especificação do projeto. Independente da representação textual, a visualização estrutural e hierárquica da especificação do circuito na forma de um grafo é um recurso importante para o desenvolvimento dos projetos.

1.1 Motivação

Os grafos de fluxo de dados começaram a ter grande uso na década de 70, motivados pelo livro *Structured Design* de Ed Yourdon e Larry Constantine (Yourdon et al., 1979). Essa notação foi proposta com base na teoria de grafo. Os grafos de fluxo de dados permitem uma melhor visualização das etapas e dos dados envolvidos nos processos de software. Além disso, os grafos podem ser aplicados em projetos para modelagem de processos de negócios e outras áreas.

Os grafos fazem parte da história da computação. Em muitos casos, grafos são um modo natural de visualizar os modelos e especificá-los. Esta dissertação explora o uso dos grafos e de ferramentas de projeto baseadas em grafos em três níveis diferentes de abstração: internet das coisas (IoT), nível de instruções e nível de registros para descrição de processadores. Nas próximas seções iremos detalhar cada um destes três tópicos.

1.1.1 Internet da Coisas

A internet das coisas (IoT) é uma grande revolução na computação (Gubbi et al., 2013) que permite a obtenção de dados de diferentes formas. Um conjunto de sensores e controladores pode ser modelado como um grafo com vários níveis de operadores. Um operador ou vértice pode ser um simples sensor, um microcontrolador, um servidor, um banco de dados, etc. A descrição de um projeto de IoT com um grafo possibilita uma visão geral de diversos dispositivos que trafegam por diferentes tipos de protocolos e meios (Belsa et al., 2018).

Segundo (Ali, 2015), a internet das coisas (IoT) é construída com base em três componentes: o hardware (sensores, atuadores e dispositivos capazes de conectar à rede), o midleware (pequeno software capaz de decodificar as entradas e saídas dos dispositivos, realizar a comunicação assíncrona entre os diversos sensores e atuadores e também armazenar os dados até que um dispositivo de manipulação de dados possa recuperá-los) e a apresentação (para ser possível visualizar, analisar e interpretar os dados para diversas aplicações). O conhecimento em IoT deve estar cada vez mais presente nos currículos de computação. Um dos desafios é como introduzir o assunto e motivar os estudantes no desenvolvimento de projetos em vários níveis.

1.1.2 Arquitetura de Processadores

O uso de aplicações que utilizam o processamento de dados em tempo real (stream) vem crescendo em diversas áreas como vídeo, áudio, mecanismos de busca, bioinformática, entre outras. Os processadores superescalares são uma solução amplamente

utilizada como plataforma de hardware para prover o processamento necessário, seja em computadores desktop, clusters e em estruturas de redes locais de computadores. O aumento de desempenho e a redução nos custos de produção fizeram esta solução se popularizar.

Desde a década de 70 ocorrem melhorias exponencias no número de transistores dos processadores superescalares, tornando-os cada vez mais rápidos. Contudo, devido a crescente complexidade com o aumento no número de transistores por unidade de área, alguns problemas intrínsecos dessa abordagem surgiram (Etiemble, 2018).

Os principais problemas encontrados são: alto consumo, falta de escalabilidade, alta complexidade nas conexões, tempo de verificação e projeto. Dentre estes, o consumo de energia é muito importante na atualidade, pois com dispositivos cada vez mais compactos é necessário o uso de baterias menores, entretanto a tecnologia de baterias não evolui na mesma velocidade. Outro fator que gera grandes problemas é o aquecimento gerado devido a alta densidade e as atividades síncronas, isso faz com que sejam necessários o uso de encapsulamentos mais caros, que sejam capazes de dissipar o calor, além do uso de sistemas de refrigeração em sua grande maioria mecânicos, que aumentam o custo final dos produtos, assim como a redução a sua vida útil dos circuitos ao trabalhar em altas temperaturas. Portanto, novos projetos devem considerar o consumo e a estimação de energia (Ferreira et al., 2000) e buscar novas alternativas com arquiteturas reconfiguráveis síncronas ou assíncronas (Ferreira et al., 2004, 2005).

1.1.3 Ensino de Fluxos de Dados

O ensino de linguagens de descrição de hardware como Verilog e VHDL é um desafio pela mudança de paradigma, principalmente para alunos de graduação em Ciência da Computação ou Sistemas de Informação, em que o foco maior é no nível de software. Outra barreira são os ambientes de simulação e projeto para FPGA, como o Vivado da Xilinx ou Quartus da Altera, que são complexos para a introdução ao assunto e requerem muito espaço em disco para instalação (Penha et al., 2016), mesmo a versão acadêmica para estudantes.

Recentemente, vários compiladores e simuladores estão disponíveis para execução em navegadores. Dentre eles podemos citar as ferramentas: Compiler Explorer (godbolt.org, 2020), Brisc-V (Agrawal et al., 2019) e EDA Tools (playground, 2020). Basicamente os ambientes são simples, com uma janela para entrada do código e outra janela para impressão. Algumas ferramentas, como EDA Tools, têm a visualização das formas de onda e podem ter uma janela separada para entrada dos *Testbenchs* (código para estimular as entradas para simulação).

1.2 Objetivos

Esta dissertação tem como objetivo trabalhar em três modelos de abstração de grafos com fluxos de dados: (a) no nível de internet das coisas, onde nosso objetivo é avaliar as ferramentas de projeto, selecionamos a ferramenta Node-Red para propor uma metodologia de ensino de IoT; (b) no nível de instruções, nosso objetivo é complementar a ferramenta READY, para desenvolvimento de aceleradores em hardware reconfigurável com uma camada com interface para edição do grafo de fluxo de dados, além de automatizar o projeto para diminuir o tempo de desenvolvimento de soluções em FPGA, minimizando os esforços que os programadores teriam no desenvolvimento de soluções para arquiteturas heterogêneas; (c) no nível de projeto de processadores, nosso objetivo é o desenvolvimento de ferramentas para simplificar a especificação de processadores RISC em Verilog, com auxílio de ferramentas de visualização do diagrama de blocos do projeto.

1.3 Estrutura da Dissertação

Esta dissertação está organizada no modelo de artigos. Na seção 1.3.1 descrevemos o primeiro artigo selecionado, intitulado "Perspectivas para o uso do *Node-Red* no Ensino de IoT", que foi publicado no *International Journal of Computer Architecture Education* e aborda o uso da ferramenta Node-Red (Passe et al., 2017) no ensino de internet das coisas ou IoT (*Internet-of-Things*). Na Seção 1.3.2 apresentamos o segundo artigo, intitulado "Plain: Ferramenta para Desenvolvimento de Aceleradores para Overlays em FPGA na Nuvem em Tempo de Execução" (Passe et al., 2020), que foi publicado no *XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, seu conteúdo descreve a ferramenta *Plain*, que permite a descrição de aceleradores usando um grafo de fluxo de dados e sua execução em FPGA de forma transparente ao programador. Finalmente na seção 1.3.3, o terceiro artigo, intitulado "Mind the Gap: Bridging Verilog and Computer Architecture" (Passe et al., 2020), publicado no *International Symposium on Circuits and Systems*, e que apresenta uma metodologia de ensino e uma ferramenta de visualização de projetos de processadores RISC, mais especificamente MIPS e RISCV, descritos na linguagem Verilog.

1.3.1 Grafo de Fluxo de Dados e Internet das Coisas

Para introduzir o assunto fluxo de dados e IoT sugerimos o uso da ferramenta Node-Red, devido a sua facilidade de uso e aprendizado, como mostrado em (Lekić and Gardašević, 2018) e (Chaczko and Braun, 2017). A ferramenta permite integrar diversos dispositivos úteis no aprendizado como Raspberry Pi e sensores como o DHT11

(temperatura). No trabalho de (Chanthakit and Rattanapoka, 2018) é apresentada uma aplicação para monitoramento da qualidade do ar, em que o Node-Red é peça central para junção das informações dos sensores. Também é possível integrar protocolos mais antigos (Tabaa et al., 2018) como o Modbus, muito utilizado na indústria. Monitorar formas de onda (Waveforms) com a adição dos sinais pelo nome ao invés de ter que localizar o sinal no grafo. Devido a presença de sobreposições, esta tarefa pode ser árdua.

Com a proposta de reduzir a complexidade que se tem para construir projetos de IoT, a ferramenta Node-Red traz diversas vantagens como a criação de servidores, a redução do tempo de implementação e a criação de interface para receber/enviar dados de/para dispositivos. O Node-red executa diretamente no navegador e processa informações de diversos meios como APIs REST, sensores, atuadores, placas IoT (ESP8266, ESP32, Arduino, etc), inteligência artificial como o Watson da IBM (que permite fazer analise de sentimento, reconhecimento de voz e imagens, entre outros recursos). Ou seja, possibilita o ensino em vários níveis de abstração. Além disso, a ferramenta é extensível e adaptável, não se restringe a um protocolo ou interface, permitindo ao usuário customizá-la.

O ambiente oferecido pelo Node-Red possui uma interface gráfica que permite a construção de fluxos de dados heterogêneos. Este trabalho complementa trabalhos anteriores (Lekić and Gardašević, 2018; Chaczko and Braun, 2017) na linha de aprendizado com exemplos para IoT e adiciona mais exemplos. As principais contribuições do trabalho, apresentadas no capítulo 2, foram:

- Divulgação da ferramenta e dos principais recursos oferecidos;
- Exemplos didáticos;
- Mostrar o funcionamento do protocolo MQTT, um dos mais usados em IoT;
- Mostrar o Dashboard (painel de controle), um recurso agregado que permite criar ambiente de visualização e análise de dados em tempo real a partir do grafo de fluxo construído.

1.3.2 Grafos de Fluxo de Dados para Aceleradores Reconfiguráveis

A escalabilidade de uma arquitetura sempre será um fator importante, mais motivado atualmente com a desaceleração da lei de Moore. Entretanto, o paralelismo a nível de instrução que um processador superescalar consegue explorar, está limitado pelas dependências de dados e de controle. Portanto, aumentar a quantidade de unidades funcionais para cálculo não implica diretamente em um ganho de desempenho. Assim, novas arquiteturas e técnicas estão sendo exploradas (Ferreira et al., 2014; Caldeira et al., 2018). Ademais, as arquiteturas tradicionais são baseadas no modelo de

Von Newmann, também conhecido como paradigma processador-memória, onde a busca centralizada de instruções e dados gera um problema de latência no sistema como um todo.

As arquiteturas de fluxo de dados são promissoras para resolver os principais problemas dos superescalares (Veen, 1986). Porém, na época em que surgiram, década de 70, seu custo de produção era inviável pois demandava o projeto e produção de circuitos específicos. Atualmente, com o rápido tempo de desenvolvimento oferecido pelas FPGAs, com as arquiteturas reconfiguráveis, tornou-se possível o uso de arquitetura de fluxos de dados como uma solução escalável, de alto desempenho, de baixo consumo e de fácil conectividade frente às arquiteturas tradicionais como as do superescalar (Zhang et al., 2011; Bansal et al., 2004; Budiu and Goldstein, 2002; Canesche et al., 2017; C. Penha et al., 2019).

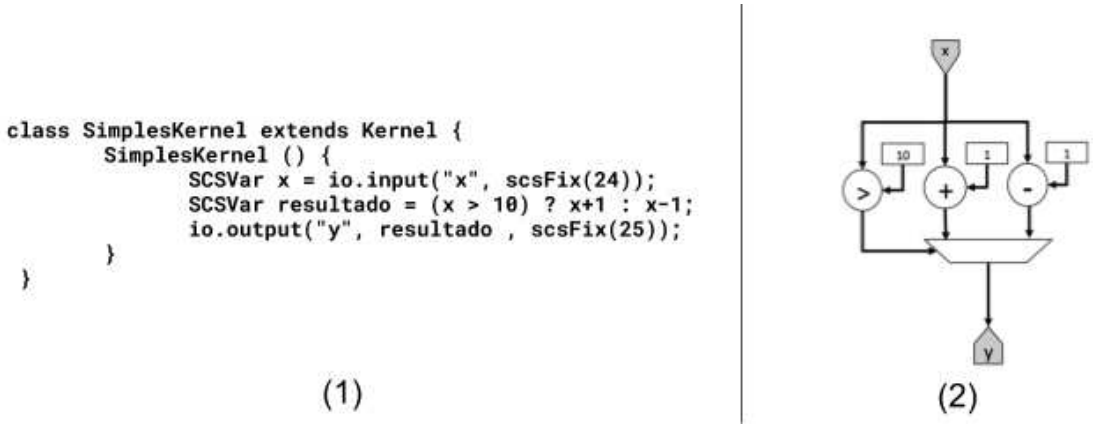


Figura 1.3: (1) Código em OpenSPL com desvio condicional; (2) Grafo de fluxo de dados - Figura extraída da referência (Consortium et al., 2013).

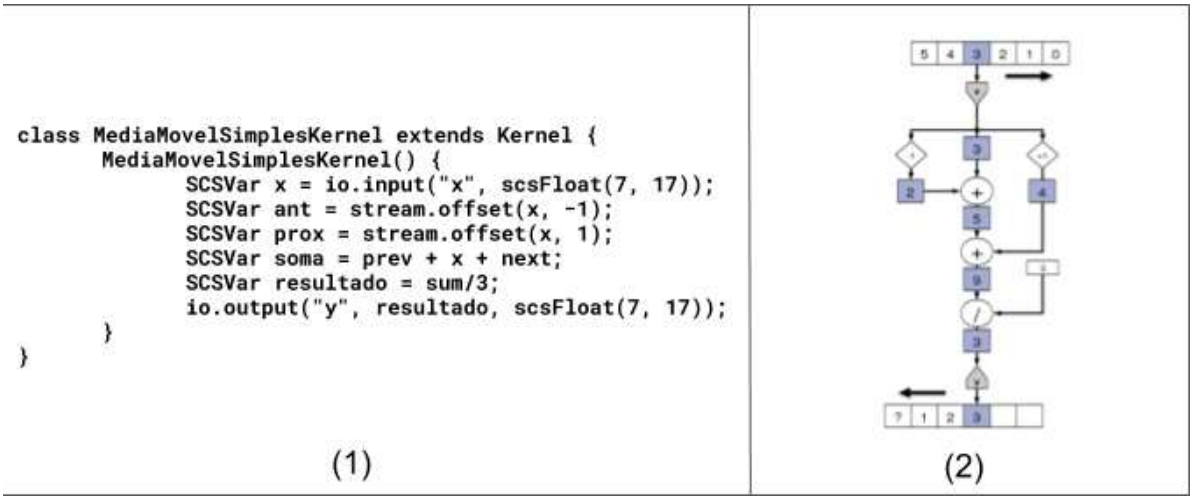


Figura 1.4: (1) Código em OpenSPL da Média móvel simples; (2) Grafo de fluxo de dados - Figura extraída da referência (Consortium et al., 2013).

```
...
for(int i=0; i<x.length; i++){
  if(x[i] % 3 == 0){
    out = x[i] + 3;
  }else if(x[i] % 2 == 0){
    out = x[i] + 2;
  }else {
    out = x[i];
  }
}
...
```

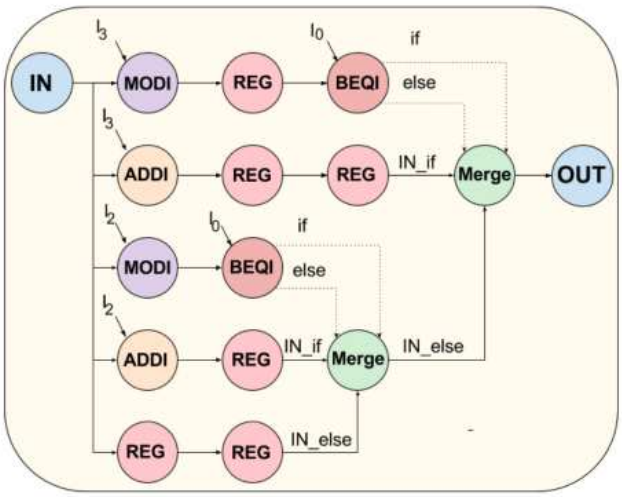


Figura 1.5: Exemplo de algoritmo com desvio e o grafo de fluxo da ferramenta ADD - Figura extraída da referência (Canesche et al., 2017).

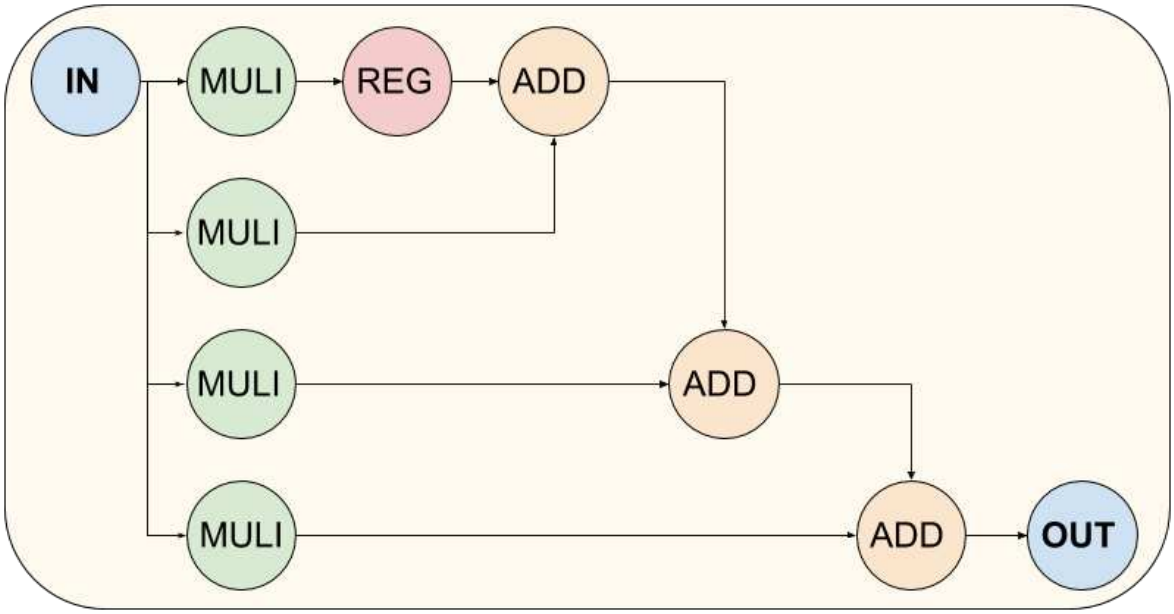


Figura 1.6: Exemplo do FIR4 na ferramenta ADD - Figura extraída da referência (Canesche et al., 2017).

Os FPGAs possibilitam a implementação da computação espacial, que oferece paralelismo explícito, onde todas as unidades que não dependem umas das outras podem executar ao mesmo tempo. No grafo, os dados percorrem diferentes caminhos em estágios variados. A figura 1.3 mostra um fluxo de controle simples onde são analisadas dois possíveis caminhos em paralelo e somente um seguirá o fluxo. Na figura 1.4 é mostrado como uma convolução em uma dimensão pode ser implementada em grafo de fluxo (Consortium et al., 2013). Ambas as soluções estão implementadas usando a linguagem OpenSPL, uma extensão de JAVA que permite a descrição dos

fluxos.

Outro exemplo é ilustrado na figura 1.5, que mostra um algoritmo com desvio implementado como um grafo de fluxo. O código foi manualmente mapeado no grafo usando os operadores disponibilizados pela biblioteca ADD (Canesche et al., 2017). A figura 1.6 apresenta o grafo de um filtro FIR com dimensão 4, também implementado com a biblioteca ADD.

Os FPGAs são flexíveis e permitem a reconfiguração dinâmica a nível de bit. Porém, a alta densidade de componentes resulta em tempo de síntese grande, podendo demorar horas para compilar. Os dois exemplos de ferramentas (OpenSPL e ADD) dependem do processo de síntese para execução em FPGA.

Uma solução é usar uma camada virtual, ou overlay, sobre o FPGA. A ferramenta READY (Silva et al., 2019) apresenta esta solução com uma arquitetura de um CGRA (Coarse-Grained Reconfigurable Array) baseada em uma interconexão multi-estágio (Ferreira et al., 2011). Entretanto não oferece muitas facilidades para entradas dos grafos e validação dos resultados. No capítulo 4, uma extensão da ferramenta será apresentada com as contribuições:

- Ferramenta Plain: uma ferramenta que permite o desenvolvimento de fluxos de dados diretamente no navegador, com geração de um arquivo JSON que possui uma estrutura simples para descreve todo grafo, possibilidade de importar/exportar sub-grafos, facilidade na inclusão de novos operadores, possibilidade de inclusão de metadados como trechos de implemetações em Verilog.

- Uso combinado da ferramenta Plain com o framework READY (Silva et al., 2019), para que o grafo de fluxo seja mapeado em tempo de execução e configure diretamente as estruturas do overlay no READY, evitando a necessidade de realizar o processo de síntese, reduzindo o tempo de mapeamento para microsegundos.

1.3.3 Simplificando o Ensino de Verilog

Uma forma de estimular o uso de fluxos de dados é o ensino de linguagens de descrição de hardware como Verilog, aliado a uma metodologia de ensino incremental (Nestor, 2005), com a introdução passo a passo de exemplos com complexidade crescente. Ao mesmo tempo, deve-se buscar a simplificação ao acesso ao uso de Verilog, e da simulação de exemplos através de interface simples como um navegador. O terceiro artigo apresentado nesta dissertação, no capítulo 3, aborda esta temática. A proposta é estender a ferramenta DigitalJS (digitaljs.tilk.eu, 2020; Materzok, 2019) para o ensino de projeto de processadores RISC, incluindo os processadores MIPS e RISC-V utilizados no livro clássico de ensino de arquitetura de computadores (Patterson and Hennessy, 2013; Patterson, 2017).

O DigitalJS oferece uma visualização interativa do grafo do circuito descrito em Verilog. A ferramenta foi desenvolvida para introdução aos sistemas digitais. Na avaliação realizada pelos estudantes da Universidade de Warsaw, o aspecto mais elogiado foi a forma interativa de visualizar o projeto (Materzok, 2019). A forma de visualizar circuitos mais complexos foi um dos pontos indicados pelo autor para melhorias. Apesar de possibilitar a organização com hierarquia, para projetos com muitas conexões, a visualização automática gera muitas sobreposições e a apresentação não fica clara.

A ferramenta foi desenvolvida com base no Yosys (Wolf et al., 2013), que é um framework de código aberto para síntese Verilog, sobre o qual foi desenvolvido um módulo em NodeJS que gera uma interface em JSON intermediária para representar a saída. Além disso, a biblioteca JoinJS (Client, 2020) foi utilizada para fazer a interface gráfica do projeto.

As principais contribuições do trabalho, apresentadas no capítulo 3, foram:

- Monitorar formas de onda (waveforms) com a adição dos sinais pelo nome ao invés de ter que localizar o sinal no grafo. Devido a presença de sobreposições, esta tarefa pode ser árdua.

- Criar um conjunto de projetos com os exemplos dos processadores RISC-V e MIPS do Livro texto (Patterson and Hennessy, 2013; Patterson, 2017).

- Extensão da interface dos módulos de memória para ler e escrever em arquivos. Assim, para validar os projetos ou as modificações nos projetos, o estudante pode facilmente carregar códigos de teste na memória de instruções e/ou arquivos de inicialização da memória de dados.

- Armazenar o grafo do circuito associado ao projeto Verilog, após a edição manual do usuário para facilitar a visualização. Assim, o estudante ao recarregar já terá uma visualização mais clara do projeto.

- Avaliar as diversas ferramentas para ensino de projeto de processadores, selecionando um subconjunto para o ensino com a inclusão do trabalho proposto para reduzir o "gap" semântico entre um diagrama de blocos do projeto e uma implementação em Verilog.

Para implementar as alterações na ferramenta DigitalJS foram utilizados recursos do NodeJS, um ambiente de execução Javascript server-side de código aberto, em conjunto com o Webpack (empacotador de módulo JavaScript). Além de explorar os recursos do DOM (Document Object Model), a interface que representa como os documentos HTML e XML são lidos pelo navegador.

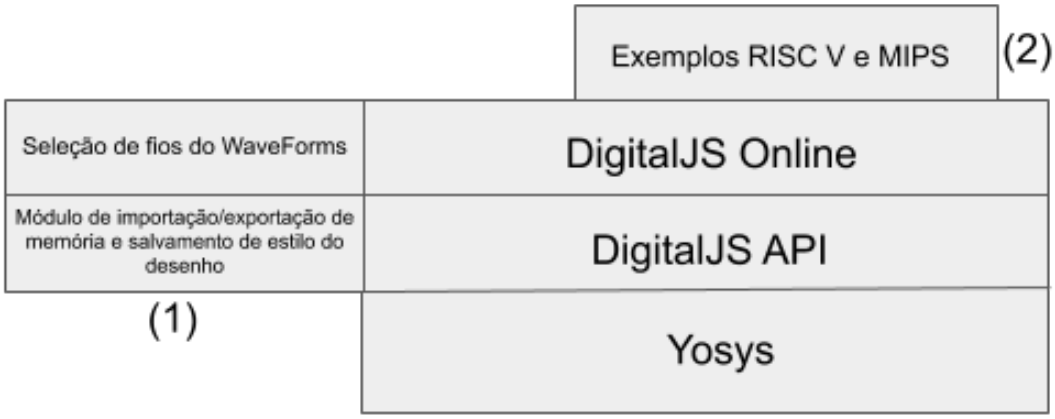


Figura 1.7: (1) Alterações do core do DigitalJS; (2) Exemplos adicionados para ensino de RISC V e MIPS.

Alguns recursos foram adicionados para melhorar a exibição dos exemplos. O modelo em JSON de cada exemplo foi adicionado para ser carregado sem requer a compilação. Este recurso preserva a exibição que pode ter sido editada para melhorar a visualização. A figura 1.7 mostra um pequeno diagrama das modificações realizadas na ferramenta.

Capítulo 2

Perspectivas para o uso do *Node-Red* no Ensino de IoT

Atualmente, uma possibilidade para o ensino de sistemas embarcados está diretamente ligada ao ensino de internet das coisas ou IoT (*Internet of things*), e este por sua vez exige diversos conhecimentos multidisciplinares. A conexão de dispositivos com a rede ainda exige conhecimento sobre protocolos *web* e construção de *web servers*. Este artigo propõe uma metodologia de ensino com exemplos simples e diversificados baseados no uso da ferramenta *Node-RED*. Esta nova ferramenta proposta pela IBM simplifica o projeto de IoT com diversas abstrações e ao mesmo tempo permite explorar tópicos avançados no ensino e pesquisa. O projeto é baseado em fluxos e apesar de recente, a ferramenta já disponibiliza muitos recursos aos programadores. Além do uso de *Node-RED* para ensino, este artigo também sugere o uso de MQTT (*Message Queue Telemetry Transport*) como protocolo base de comunicação.

2.1 Introdução

A Internet das Coisas (IoT) é uma realidade e deve fazer parte da grade curricular dos cursos de Ciência da Computação, Engenharia da Computação e Sistemas de Informação. Conforme previsto por Mark Weiser em seu artigo "*The Computer for the 21st Century*" (Weiser, 1999), a computação está se infiltrando em locais que antes eram impossíveis e hoje já faz parte do cotidiano. Os equipamentos comunicam-se localmente ou pela Internet, adicionando um grau de complexidade ao ensino de sistemas embarcados e redes. Em um futuro próximo, todos os equipamentos estarão conectados e farão parte de algum aplicativo. Para os estudantes iniciantes é difícil buscar respostas para vários assuntos devido ao excesso de informações fragmentadas, por exemplo, questões como "por onde começar", "como lidar com projetos de sistemas com vários padrões de conectividade e serviços", etc., assolam a todos.

A área de IoT exige conhecimento multidisciplinar para a construção e conexão de novos dispositivos e para lidar com as novas formas de interagir com o usuário. Plataformas de hardware e software para prototipagem e testes de sistemas como *Ar-*

duino e *Raspberry Pi* são opções viáveis para o ensino, devido ao seu baixo custo e sua usabilidade. Este artigo propõe utilizar o ambiente *Node-RED* (Blackstock and Lea, 2014) desenvolvido pela IBM para o ensino de IoT. A ferramenta *Node-RED* torna o ensino de IoT uma tarefa simples pois permite aos programadores modelar, implementar e validar diversas aplicações usando uma interface interativa, amigável e com uma curva rápida de aprendizado. A metodologia utilizada é a de projetos e ensino com exemplos.

Este documento está estruturado da seguinte forma: primeiramente, na seção 2.2 é apresentada a ferramenta *Node-RED*. Na seção 2.3 é introduzido o protocolo MQTT (*Message Queue Telemetry Transport*), um padrão utilizado em projetos com IoT. Posteriormente, na seção 2.4, os exemplos propostos detalham o uso do *Node-RED* além de mostrarem como desenvolver projetos e como adicionar códigos existentes ao projeto. Finalmente, há sugestão de documentação complementar e comentários finais para o ensino.

2.2 Node-Red

Criar soluções IoT envolve conhecimentos de eletrônica, redes e programação, no qual um dos fatores mais importantes é lidar com as diversas conexões com dispositivos (embarcados, servidores, *smartphones*) através de vários serviços na internet, sendo também necessário o desenvolvimento de programas para enviar, controlar e analisar os dados capturados. Assim para suprir as necessidades desse nicho e projetar os sistemas de IoT surgiram diversas ferramentas (Mineraud et al., 2016; Kleinfeld et al., 2014) e entre elas está o *Node-RED* (Blackstock and Lea, 2014; Guinard and Trifa, 2016; Giang et al., 2015).

O *Node-RED* é um software de código aberto desenvolvido pela IBM que permite a programação através de fluxos (*flow-based programming*), usando uma interface no navegador que pode ser vista na Figura 2.1.

A plataforma possui vários nós com diferentes funcionalidades que podem ser conectados de forma coerente permitindo a passagem do fluxo de informações e criando aplicações. O princípio é simples, cada nó possui uma funcionalidade bem definida e a maioria dos nós abstrai a implementação do programador. A Figura 2.2 apresenta o conjunto de nós que são instalados em conjunto com o *Node-RED*. Através do ambiente, o programador projeta diversos fluxos de informação naturalmente assíncronos e paralelos com orientação a eventos. Há diversas possibilidades para a criação destas aplicações com os nós disponíveis, como exemplos de funcionalidades podemos enumerar: conexão com *Twitter*, *Facebook*, *Telegram*, protocolos de rede (udp, tcp, http, mqtt, etc.), conexão local serial, conexão com banco de dados SQL e Não-SQL como *MongoDB*, *mySQL*, etc., painel para visualização de dados com diversos

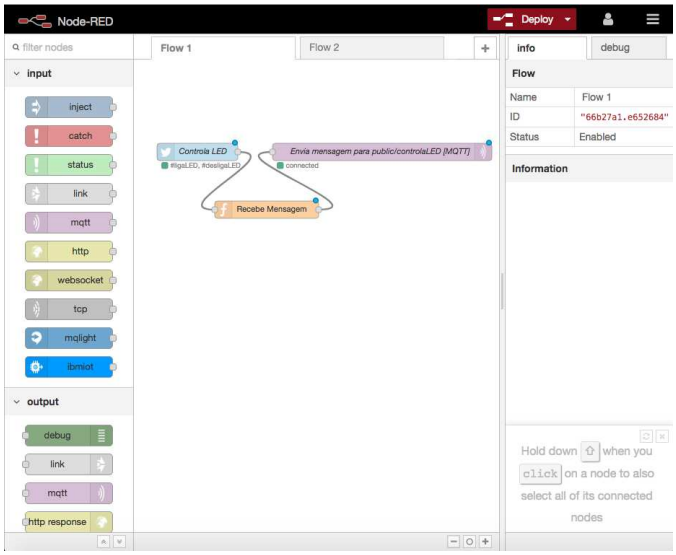


Figura 2.1: Exemplo da interface do software.

recursos (gráficos, tabelas, acionadores, etc.), envio de e-mails, conexão com *Chat-bots*, processamento de áudio e vídeo, conexão com o sistema de arquivos, análise de dados com recursos simples e complexos, implementação de funções em *Javascript*, *Python* e outras linguagens, funcionalidades específicas para *Raspberry Pi* e *Arduino*, conversão de texto em áudio e vice-versa, ou seja, existe um potencial de utilização para vários recursos com fácil integração e conectividade em uma interface simples, rápida e acessível através de um navegador.

Figura 2.2: Alguns nós disponíveis no *Node-RED*.

Há diferentes formas para a utilização do *Node-RED*, este projeto utiliza duas. A

primeira é o *Bluemix* (Kobylinski et al., 2014), plataforma desenvolvida pela IBM para computação na nuvem que possui cota gratuita em seus programas universitários. A interface *web* para o *Node-RED* executando no *Bluemix* pode ser vista na Figura 2.1. Para facilitar a busca por informações adicionais, este trabalho concentrou em uma página *web* (dpi.ufv.br, 2019) uma seleção de links que indicam onde buscar maiores informações sobre o *Bluemix* e sobre outros sistemas que serão apresentados durante o artigo. Outra forma de utilização do *Node-RED* é instalá-lo em um ambiente local ou em um servidor privado, que pode ser um *Raspberry Pi* ou um computador, conforme feito na Universidade Federal de Viçosa (dpi.ufv.br, 2019). As vantagens desta segunda opção são a gratuidade e o maior controle sobre o serviço, as informações sobre a instalação podem ser obtidas no *GitHub* do projeto (dpi.ufv.br, 2019).

2.3 MQTT

Em conjunto com o ambiente *Node-RED*, este artigo ilustra exemplos baseados no protocolo MQTT (Hunkeler et al., 2008), o qual vem se popularizando pelas suas funcionalidades adequadas ao suporte em IoT. Nos exemplos apresentados, o MQTT será utilizado para troca de mensagens entre os nós do *Node-RED* e os dispositivos embarcados (como *Arduino/ESP8266, Raspberry Pi*, entre outros). Lembrando que um nó no grafo da ferramenta *Node-RED* pode realizar um processamento ou pode ser um ponto de entrada/saída para comunicação em um ambiente com serviços na internet (*Facebook, Twitter, Web Servers*), dispositivos embarcados e sensores, serviços de comunicação (*email, telegram, SMS*), dentre outros.

O MQTT é uma forma leve e simples que pode ser integrada para comunicação neste ambiente heterogêneo, o qual funciona basicamente como um sistema com publicação (*publish*) e assinatura ou subscrição (*subscribe*), ou seja, uma informação é publicada em um servidor de MQTT que é denominado pelo termo *broker*. A informação é enviada a partir de um dispositivo e/ou servidor e no *broker* esta informação ou mensagem fica disponível para ser capturada por um ou mais dispositivo(es)/servidor(es), a Figura 2.3 ilustra seu funcionamento básico. A informação é organizada no padrão REST (*Representational State Transfer*) que representa o tópico que foi publicado.

Outra funcionalidade básica e simples do MQTT é o controle da qualidade do serviço ou QoS (*Quality of Service*). A Figura 2.4 ilustra o seu funcionamento. Há três diferentes níveis de qualidade definidos pelo MQTT, eles variam de acordo com a segurança na entrega da mensagem para o assinante, os níveis são:

1. QoS Nível 0: Neste nível, a mensagem publicada não será armazenada por quem a enviou ou reconhecida pelo receptor. Tal comportamento é conhecido,

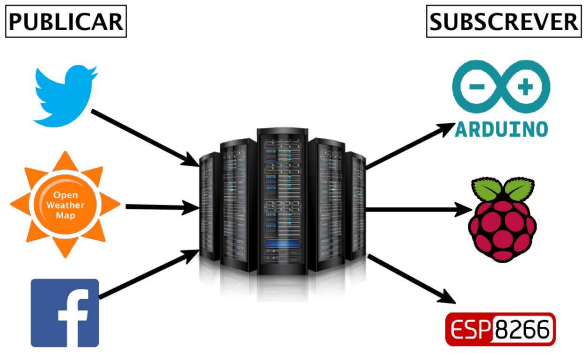


Figura 2.3: *Broker* com dispositivos e serviços conectados.

na maioria das vezes, como "*fire and forget*"e provêm a mesma garantia que o protocolo TCP (*Transmission Control Protocol*) submetido.

2. QoS Nível 1: O nível 1 garante que a mensagem vai ser entregue pelo menos uma vez para o receptor, mas ela também pode ser entregue mais de uma vez.

3. QoS Nível 2: Este nível garante que a mensagem é recebida apenas uma vez pelo receptor. É o nível mais seguro do MQTT porém o mais lento.

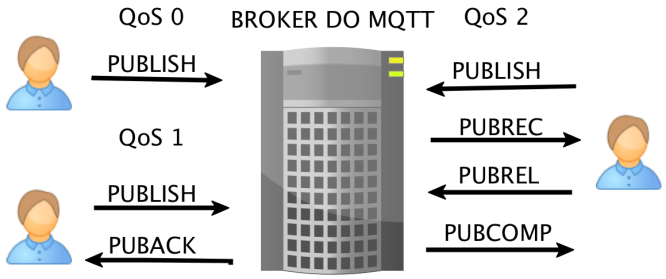


Figura 2.4: *Quality of Service* do MQTT

Como mencionado, uma publicação e um ou mais assinantes se comunicam através de tópicos no *broker*. O tópico é uma *string* e seus níveis são separados pelo caractere /, um exemplo de tópico seria universidade/disciplinas/arquiteturadecomputadores. Para se referir aos tópicos, também é permitido o uso de caracteres-curinga, que são o + e o #.

O caractere + é um substituto para um nível de tópico, universidade/+/arquitetura decomputadores poderia se referir a universidade/disciplinas/arquiteturadecomputadores ou a universidade/turmas/arquiteturadecomputadores por exemplo, mas não se referiria a universidade/disciplinas/estruturadedados.

O caractere # é um curinga para vários níveis mas só pode ser usado no final de um tópico, por exemplo, universidade/disciplinas/# poderia se referir a universi-

dade/disciplinas/estruturadedados/turmaA, universidade/disciplinas/estruturadedados/turmaB, universidade/disciplinas/arquiteturadecomputadores/turmaA entre outros.

2.4 Ambiente Experimental

A metodologia proposta aqui é baseada no aprendizado de IoT com exemplos no ambiente *Node-RED*. Inicialmente, a seção 2.4.1 ilustra exemplos que implementam diversas funcionalidades para explorar o ambiente. A seção 2.4.2 detalha a interface da ferramenta para projetos e a abordagem baseada em fluxos. Enquanto a seção 2.4.3, apresenta a interface para uso e comunicação, onde usando um ou mais painéis de controle (*dashboard*), o usuário do serviço, através de uma página em um navegador, pode se comunicar com diversos nós do sistema de IoT através de botões, gráficos, etc. A seção 2.4.4 ilustra o potencial de programação dos nós. A seção 2.4.5 enumera várias sugestões de projetos para ensino de IoT. Finalmente a seção 2.4.6 indica onde se pode buscar informação complementar.

2.4.1 Explorando os recursos

Os primeiros exemplos são simples e ilustram as diversas formas de conectividade. O primeiro exemplo ilustra como conectar: *Twitter* → *Node-RED* → MQTT → *Arduino/-Raspberry*, ou seja, como acender um Led em um dispositivo embarcado a partir do *Twitter*.

A Figura 2.5 ilustra uma possibilidade de comunicação permitida pelo *Node-RED*. Deste modo, um usuário pode *tweetar* #ligaLED ou #desligaLED. O nó *Twitter* é uma abstração onde só é necessário preencher algumas informações e todo o serviço de comunicação é configurado de forma transparente. A informação do *Twitter* é passada para um nó responsável por identificar estas palavras-chave, converter a informação em número, 0 ou 1, ligado ou desligado. Este nó é um exemplo que implementa uma computação que será executada no servidor do *Node-RED*. O código que identifica essa informação é simples e foi escrito em *JavaScript*. A função retorna uma *String* que será repassada para um nó MQTT. Este último nó é capaz de se comunicar com um *broker* MQTT e publicar em um tópico, neste caso o tópico é public/controlaLED. Na configuração do nó MQTT, o programador adiciona apenas o endereço do *Broker* e o tópico. A mensagem que será publicada vem do nó anterior. Novamente, uma abstração facilita o uso do protocolo e a comunicação através da rede de forma transparente ao programador.

O segundo exemplo explora o uso de APIs, para obtenção de dados e acionamento de dispositivos.

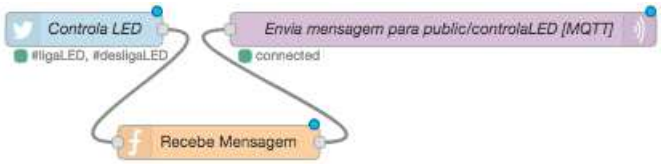


Figura 2.5: Exemplo de programação baseada em fluxo: *Twitter* → *Node-Red* → MQTT → *Arduino/Raspberry*

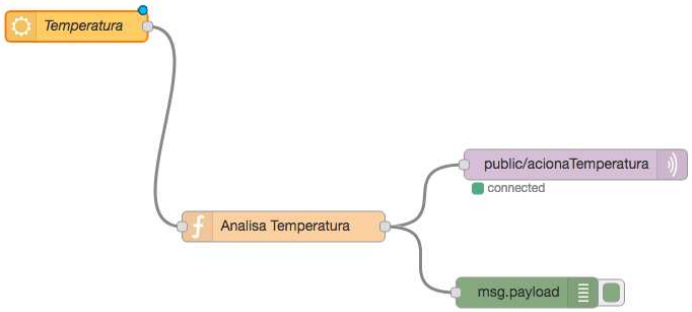


Figura 2.6: Obter dados de uma API.

Na Figura 2.6 foi utilizada uma API para obter a previsão do tempo, a configuração dos nós é bem simples, sendo necessário apenas informar a localidade a qual se deseja obter a previsão e a chave da API que pode ser obtida no site do *OpenWeatherMap*. A Figura 2.7 mostra a janela onde estas informações devem ser inseridas. O nó temperatura retorna a previsão do tempo em JSON que pode ser visto na Figura 2.8, e este é uma estrutura de dados que pode ser manipulada em *JavaScript*. A partir desta estrutura foi extraído a temperatura máxima utilizando um nó do tipo *function*, no qual é necessário apenas atribuir ao campo *payload* da mensagem a temperatura máxima obtida através do JSON e retornar essa mensagem para o próximo nó. O código tem apenas duas linhas e pode ser facilmente modificado para retirar as outras informações do objeto JSON ilustrado na Figura 2.8.

No exemplo seguinte foram combinados MQTT e alguns nós especiais como o *Twilio* que permite o envio de SMS.

Na Figura 2.9 é possível observar que um JSON é obtido a partir do MQTT, deste extrai-se um atributo e seu conteúdo é verificado. Os possíveis valores do atributo são verdadeiro ou falso e, em seguida, utiliza-se um nó *Switch* para criar dois fluxos, um para cada possível valor do atributo, repassando-se assim a informação para o MQTT e um SMS diferente é enviado para cada tópico.

2.4.2 Painel de Projeto

O painel de projeto do *Node-RED* é dividido em três seções. A primeira mostra os diversos nós disponíveis divididos em *inputs*, *outputs*, *debug* e serviços auxiliares como

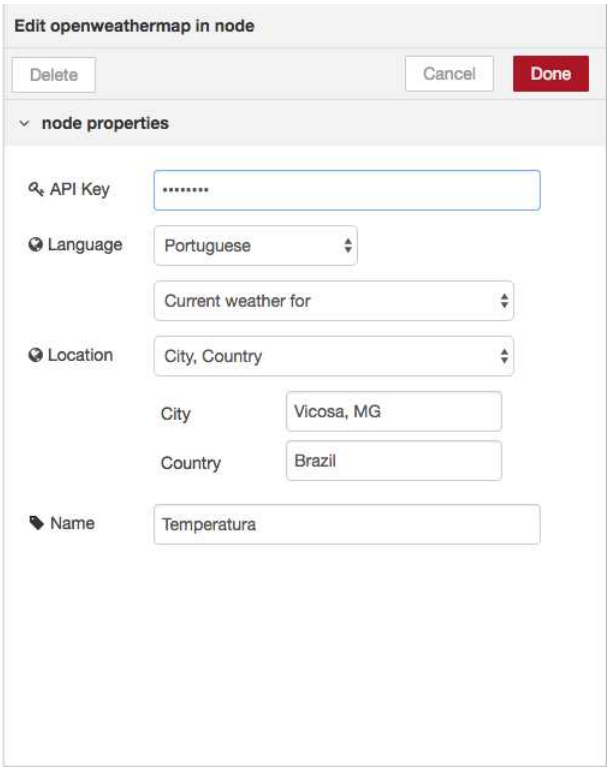


Figura 2.7: Exemplo de configuração do nó do *OpenWeatherMap*.


```
{
  "weather": "Clear",
  "detail": "céu claro",
  "tempk": 299.881,
  "tempc": 26.6,
  "temp_maxc": 26.6,
  "temp_minc": 26.6,
  "humidity": 47,
  "maxtemp": 299.881,
  "mintemp": 299.881,
  "windspeed": 4.37,
  "winddirection": 342.003,
  "location": "Vicosa",
  "sunrise": 1503220180,
  "sunset": 1503261593,
  "clouds": 0,
  "description": "The weather in
     Vicosa at coordinates:
     -20.75, -42.88 is Clear (céu
      claro)."
}
```


Figura 2.8: JSON retornado pelo *OpenWeatherMap*.

o *Watson*, Figura 2.10a. A segunda seção, Figura 2.10b, é onde o usuário constrói suas aplicações, podendo escolher e conectar os diversos nós disponíveis na plataforma. A

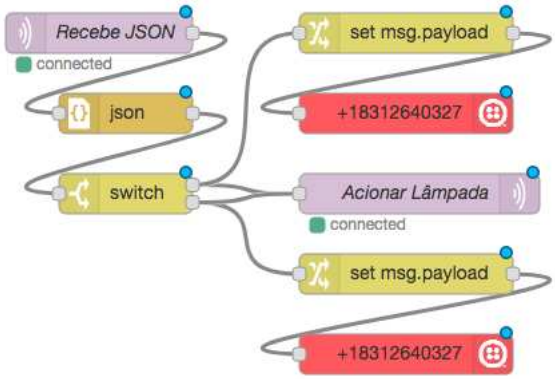


Figura 2.9: MQTT e SMS

terceira seção, Figura 2.10c, oferece opções de *debug*, porém, para utilizar tal funcionalidade, o usuário deve adicionar os nós próprios para *debug* durante a construção da aplicação.

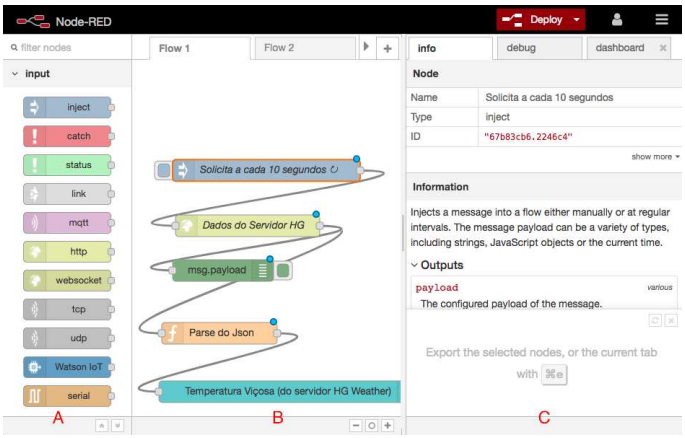


Figura 2.10: Tela de interface do Node-RED

Também estão disponíveis opções para exportação em JSON que permitem ao usuário levar os chamados *flows* de um *Node-RED* em um servidor para um *Node-RED* em outro servidor, o que é bastante útil quando se deseja manter os *designs* para uso posterior.

Além do recurso de exportar, existe a possibilidade de instalar pacotes para incluir mais nós na ferramenta e assim incluir mais recursos. Para realizar a instalação é necessário acessar a opção *Settings* → *Palette* → *Install*, onde é possível pesquisar pelos novos pacotes.

2.4.3 Painel de Controle

Um recurso muito interessante disponível na ferramenta é a possibilidade de construir painéis de controle como os da Figura 2.11, onde se pode agrupar diversas funciona-

lidades de controle ou monitoramento em uma interface gráfica amigável. Estão disponíveis diversas funcionalidades, como gráficos de barras, gráficos de pizza, botões, interfaces de entrada de dados, saída de áudio, entre outros.

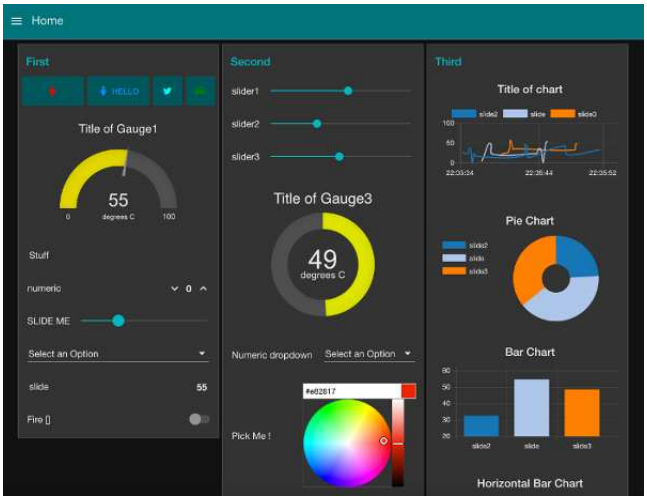


Figura 2.11: Exemplo com três painéis de controle (*dashboards*) que podem ser acessados por um navegador, mesmo de *smartphones*

Conforme foi mostrado nos exemplos da seção anterior, implementar este recurso é o mesmo que usar o nó desta funcionalidade. Entretanto, para a utilização do *dashboard* é necessário instalar um pacote chamado '*node-red-dashboard*'.

Após a instalação do pacote é criado no domínio de instalação do *Node-RED* uma pasta de "ui", ficando então o *dashboard* em dominio-node-red/ui, onde é possível ter acesso a todos os controles previamente configurados para o uso no painel.

Para o uso da funcionalidade é necessário que se ligue as saídas de dados às saídas do tipo painel. As opções de agrupamento e organização dos *Widgets*(como são chamados os objetos no painel), são mostradas nas opções do nó, facilitando a organização e visualização dos mesmos no resultado final do painel.

2.4.4 Programação

Além da biblioteca de nós que está disponível com mais de mil funcionalidades, existem mais de 600 exemplos de projetos com fluxos disponíveis na biblioteca de exemplos (dpi.ufv.br, 2019). O programador pode criar novos nós ou simplesmente usar nós que implementam funções para codificar seus algoritmos. O nó mais usado para programação é o nó *function* onde é inserido um código *JavaScript*, no qual ele pode retornar vários valores em saídas diferentes permitindo o controle de fluxo. Existem também nós para programação em *Python* nas versões 2.7 ou 3.0.

O *Node-RED* também pode ser incluído dentro de um código de aplicação ou executar um código no servidor local com uma simples chamada do sistema operacional.

2.4.5 Exemplos de Projetos

Além dos exemplos demonstrados neste artigo, existem diversas outras possibilidades de utilização do *Node-RED*. Com o tutorial encontrado em (dpi.ufv.br, 2019) por exemplo, é possível construir uma API para reconhecimento de imagem usando o *Watson*, o *Node-RED* e uma codificação bastante simplificada.

Há outros projetos que disponibilizam nós e fluxos para serem utilizados sozinhos ou como parte de um projeto maior, a Figura 2.12 mostra a utilização de um nó encontre meu *iPhone*, onde os dados necessários são apenas o login e a senha da conta da *Apple*. Em adição a estes nós, também é possível encontrar nós que se integram a plataforma *Trello*, que facilitam a utilização das APIs do *Google* e que criam um *ChatBot* para o *Telegram*, *Messenger*, *Facebook* ou *Slack*, entre outros. Também em (dpi.ufv.br, 2019), há um tutorial para integrar o *Amazon Echo* com o *Node-RED* e expandir as habilidades da *Alexa*. O tutorial já fornece um fluxo base pronto mas nada impede a criação de novos comandos adicionais.

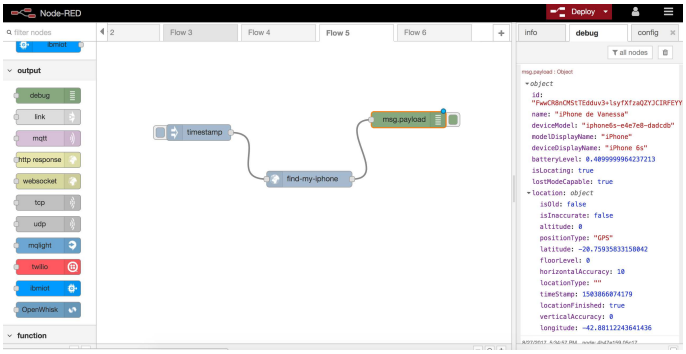


Figura 2.12: Nó Encontre meu *iPhone*.

A utilização de novos nós em conjunto é possível e um bom exercício para se familiarizar com a ferramenta. Um exercício interessante é criar um *Chatbot* com qual se pode conversar através do *Messenger*, perguntar onde está o meu *iPhone*, quais meus próximos compromissos no *Trello* ou mesmo como está o clima hoje e receber a resposta pelo *Chatbot* no mesmo *Messenger*.

2.4.6 Documentação

Como sugestão para os leitores de material introdutório há o tutorial disponível no site oficial do *Node-RED* (dpi.ufv.br, 2019), sendo ele organizado em lições, no qual a primeira lição apresenta três exemplos: uso de *Twitter* controlando uma saída do *Raspberry Pi*, acesso de previsão do tempo com processamento e envio de *email* e como terceiro exemplo um servidor web com apenas seis nós.

São 9 seções no total, 7 já disponíveis, e entre os tópicos é explicado como usar o nó *debug* e o nó de função para abstrair grande parte do código em *JavaScript*.

Também é explicado como usar os nós mais comuns, o MQTT, *websockets* e requisições TCP. É descrito o modelo de programação utilizado pelo *Node-RED*, como usar os nós intermediários e como tirar melhor proveito do *dashboard* oferecido. Além de disponibilizar vários exemplos em cada lição.

2.5 Considerações Finais

Existem diversas opções para ensino e projetos de IoT, dentre os quais este artigo apresenta uma ferramenta da IBM, o *Node-RED* de código aberto. Ao explorar a ferramenta, podemos observar que é flexível e já se encontra em um estado maduro de projeto para ser amplamente utilizada. A Figura 2.13 mostra uma avaliação no *Google Trends* dos últimos 12 meses relatando as pesquisas relacionadas a cinco ferramentas de IoT, no qual dentre estas o *Node-RED* se destaca. Outro ponto interessante é que pode ser usado para iniciantes e ao mesmo tempo pode ser usado em trabalhos avançados de pesquisa, onde os estudantes podem agregar novas funções ao ambiente. Além de ser código aberto, é disponibilizado documentação, exemplos e uma ampla biblioteca com mais de mil funcionalidades. Acreditamos que o *Node-RED* é uma ferramenta de IoT que irá se popularizar rapidamente com o paradigma de projeto orientado a fluxos. O conceito de nós e de que uma aplicação ou serviço pode ter vários pontos de entrada e saída, desacopla os serviços em fluxos, permitindo uma visão local dos componentes que estão interagindo e modela de forma simples um ambiente complexo onde tudo está potencialmente conectado.

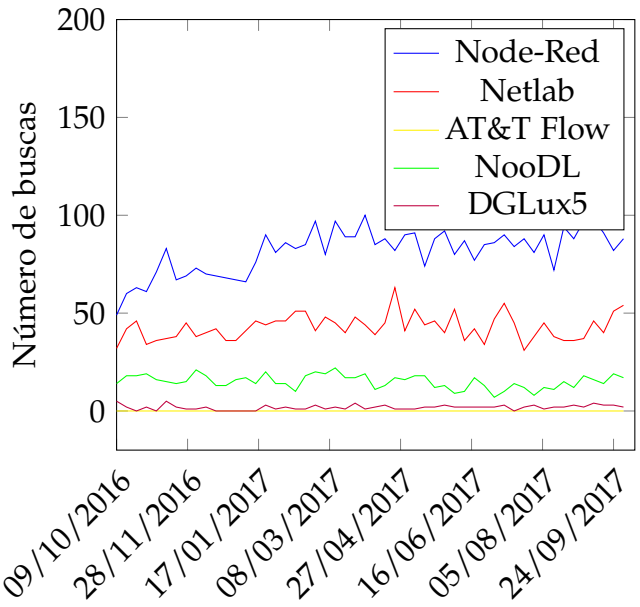


Figura 2.13: Tendências de Ferramentas de Projeto de IoT

2.6 Agradecimentos

Os autores agradecem ao CNPq, CAPES e FAPEMIG pelo suporte fornecido no desenvolvimento do trabalho.

Capítulo 3

Mind the Gap: Bridging Verilog and Computer Architecture

We present an approach to teach RISC processor design for an undergraduate computer architecture course specifically aimed to reduce the gap between a high-level datapath block diagram and a complete Verilog code specification. We propose a graphical approach designed to develop an understanding of the MIPS processor organization at the Verilog structural level by using an online browser-based simulator, from a single cycle design to pipeline design. The students are lead through a series of examples, step by step, and they can actively be involved in the processor design process. We believe that the best choice should not introduce excessive complexity that becomes a barrier in describing the interconnection of a high-level diagram and the Verilog implementation code.

3.1 Introduction

Amazon, Google, and Microsoft cloud-based systems contain around 10 million of 80x86-based servers, while only in 2017, 10 million RISC chips ship every four hours (Patterson, 2017), and 99% processors today are RISC. Most of the undergraduate courses in computer architecture use the Patterson and Hennessy best-seller book (Patterson and Hennessy, 2013), which provides an essential reference on RISC design. Moreover, these courses highlight both the importance and the challenges of computer architecture, which further motivative research in the field.

There are several options for teaching RISC, more specifically MIPS processor, supported by simulators at assembly level and/or datapath level (ntu.edu.sg, 2020; eng.cam.ac.uk, 2020; cs.sonoma.edu, 2020; jamesgart.com, 2020; umass.edu, 2020; Araujo et al., 2014; Vollmar and Sanderson, 2006b; Kabir et al., 2011; Nova et al., 2013; Marwedel et al., 2002; Kho and Uy, 2017; Ferreira et al., 2015). Recently, there are also many options for light browser-based compilers/simulators to introduce Verilog language (playground, 2020; tutorialspoint.com, 2020; jdoodle.com, 2020; techep.csi.cuny.edu, 2020). Even with the current tools, many students have difficul-

ties in writing a Verilog code. As long as there is nothing in between, we propose an approach to bridge the gap between high-level datapath diagram and a complete Verilog design. In contrast to other tools, this work is not yet another MIPS simulator at an assembly level or a graphical animated datapath interface. Our tool (UFV, 2020) has a simple light browser-based interface, where the student is motivated to grasp new ideas to extend basic design examples. Our approach allows students to modify Verilog code, enriching their processor realization. We follow the Patterson/Hennessy book text, which describes only a few instructions and gives opportunities for interactive learning by modifying/extending the instruction set directly working on the Verilog implementation. In addition to MIPS design, we add two more example sections to teach RISC-V, and branch prediction.

Recently, the first open processor RISC-V has been proposed, which belongs to a nonprofit foundation with more than 65 corporate members (riscv.org) (Patterson, 2017). While RISC-V documentation has 236 pages, the ISA manual for x86-32 is 2,198 pages. The RISC-V main advantages (Patterson, 2017) are: (i) prevents errors from previous designs; (ii) the base ISA is stable, and it will never change; (iii) provides reserve code to support domain-specific architectures (DSAs); (iv) modular and small ISA. Our material provides support for working with RISC-V, which is also covered in the new edition of Patterson and Hennessy's book (Patterson and Hennessy, 2017).

Also, we propose to teach caches and branch predictors by using simple implementations to take a step forward in building a more complex processor by extending the book MIPS/RISC-V design since the textbooks do not cover these implementations in detail.

This paper is structured as follows. Section 3.2 presents the previous works on MIPS simulators and online Verilog tools. Section 3.3 describes our approach to fill the gap. Section 3.4 presents a complete set of tools. Finally, Section 3.5 concludes and outlines future work.

3.2 Related Work

There are already several MIPS simulators (ntu.edu.sg, 2020; eng.cam.ac.uk, 2020; cs.sonoma.edu, 2020; jamesgart.com, 2020; umass.edu, 2020; Araujo et al., 2014; Vollmar and Sanderson, 2006b; Kabir et al., 2011; Nova et al., 2013; Marwedel et al., 2002; Ferreira et al., 2015; Patti et al., 2012), where some tools have animated graphical interface (eng.cam.ac.uk, 2020; Araujo et al., 2014; Kabir et al., 2011; Nova et al., 2013; Marwedel et al., 2002; Ferreira et al., 2015; Patti et al., 2012). However, most of them are too specific or not very flexible. In this section, we select a subset of MIPS and RISC-V tools. We base our selection criteria on the premise that a simulator should be flexible and easy to use, well-documented, and open-source to be easily extendable.

3.2.1 MIPS simulators

The Patterson/Hennessy book recommends SPIM, which is a simulator for assembly language or executable MIPS files (Larus, 1990). Another popular simulator is Mars (Vollmar and Sanderson, 2006b,a), which is intuitive and widely used for teaching MIPS assembly. In addition, Mars has syntax-highlighted code, and register/-memory values windows. Moreover, Mars is an extendable simulation framework written in Java, with support for the addition of plugins. One example is the graphical animation plugin for the single cycle MIPS datapath(Araujo et al., 2014). Assembly editors are useful to introduce simple code examples. Another approach is a high-level quantitative analysis to measure code properties. In general, a set of applications and/or benchmarks are compiled, and cycle level parameterized simulators such as Simplescalar (Burger and Austin, 1997) and Gem5 (Binkert et al., 2011) extract data to be analyzed for different high-level design decisions such as cache size or branch predictor. A comprehensive comparative literature survey of the state-of-art computer architecture simulation approaches is presented in (Akram and Sawalha, 2019).

While text-based assembly simulators serve as frontend tools, graphical approaches are often easier to introduce the processor implementation details, where the graphical animations highlight the control and data signals. Two examples of light and simple visual datapath approaches are the Java-based tools: Ravi (Marwedel et al., 2002) and ViSiMIPS (Stupefied, 2020; Kabir et al., 2011). Both tools intuitively offer animation, colors, and register/memory/wire visualization. Recent works provide similar graphical datapath for RISC-V processor in addition to a Web interface as the *WebRISC-V* (Giorgi and Mariotti, 2019; dii.unisi.it, 2020) and the *Ripes* (Mortbopet, 2020) as well as FPGA implementations (Dennis et al., 2017; Gür et al., 2018; Zang et al., 2019).

Recently, the BRISC-V simulator was released which presents a complete browser-based RISC-V C compiler and assembly editor/simulator (Agrawal et al., 2019) . Also, this tool has a high-level and parametrizable generator of the complete Verilog code for a single or multi-core RISC-V processor. However, this tool presents a straightforward block diagram and automatically generates the Verilog code. The student can only choose a few parameters, and there is still a considerable gap from the block diagram viewer to the advanced generated Verilog code.

3.2.2 Verilog Browser-based User Interfaces

Hardware description languages such as Verilog and VHDL are teaching challenges in computer engineering and computer science courses. Moreover, CAD tools are complex and huge frameworks. For instance, the Xilinx Vivado installation tool has around 10 GB. Recently, light-weight browser-based tools are available. Two examples

are the browser-based Icarus Compiler/Simulator (tutorialspoint.com, 2020) and the JDoodle (jdoodle.com, 2020). These tools are Verilog editors/simulators. There is also another browser-based tool with a separated window for test benches and waveform visualization (techep.csi.cuny.edu, 2020). Moreover, EDA playground (playground, 2020) provides free and commercial [1] simulators.

Recently, Marek Materzok from the University of Wroclaw presented a new approach for browser-based Verilog simulator, the DigitalJS (digitaljs.tilk.eu, 2020). The tool has three windows: code, waveform, and structural viewer, as shown in Figure 3.1. The student writes a Verilog code or loads an example. The tool simulates and draws, on-the-fly, a structural view which helps to understand the code behavior. It is important to highlight that the tool generates the structural viewer from scratch. Besides, the structural viewer is hierarchical, and by selecting a component, a new window shows the respective internal structure. The components and wires can be dragged and dropped to make a design visually more understandable. This project is also open source.

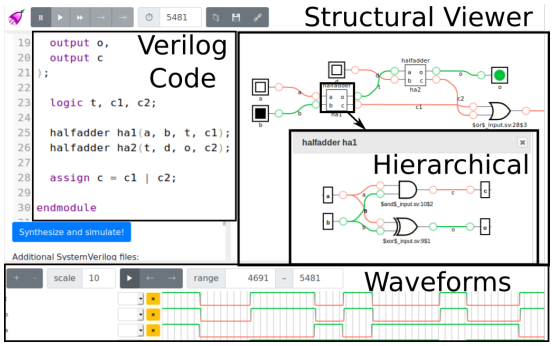


Figura 3.1: DigitalJS Screenshot for a full-adder example.

3.3 Design by Example

Some simulators can be configured using a file and through the graphical interface. Our approach uses the Verilog code as the starting point and, at the same time, displays the datapath graphically and hierarchically. We propose to extend the open-source *DigitalJS* tool for a set of lab activities (UFV, 2020) designated based on the Patterson/Hennessy books (Patterson and Hennessy, 2017, 2013). We have chosen *DigitalJS* since it is based on Yosys open-source framework(Wolf et al., 2013) for Verilog RTL synthesis, which can support complex design, and at the same time being stable.

The student could easily understand and modify the code. For instance, the following code implements the book MIPS ALU specification:

[1] Synopsis, Cadence

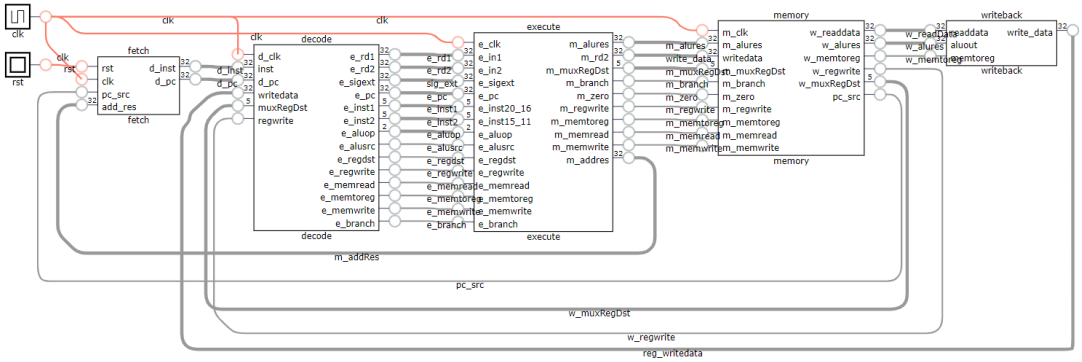


Figura 3.2: MIPS pipeline structural view in the extended *DigitalJS*.

```
module ALU(
  input [3:0] alucontrol,
  input [31:0] A,
  input [31:0] B,
  output [31:0] aluout,
  output zero
);
 reg [31:0] aluout;
 assign zero = (aluout == 0);
 always @(alucontrol, A, B) begin
  case (alucontrol)
   0: aluout <= A & B; // AND
   1: aluout <= A | B; // OR
   2: aluout <= A + B; // ADD
   6: aluout <= A - B; // SUB
   7: aluout <= A < B ?32'd1:32'd0;//SLT
   12: aluout <= ~(A | B); // NOR
   default: aluout <= 0; //default
  endcase
 end
endmodule
```

Even a beginner student can understand the code, correlating it with the textbook ALU description, where we have defined the AND opcode as 0 (zero), for instance. Morever, it is straightforward to add new operations by adding new opcodes and operations inside the case statement.

First, we have implemented a structured modular design of the MIPS version to generate a hierarchical view of the single-cycle MIPS, the pipeline MIPS with or without forwarding and hazard detection, and a static two-issue datapath. We label all wires, modules, and inputs/outputs following the book figures, which generates a mapping between the book datapath block diagrams and the Verilog codes.

Most simulators designed for undergraduate computer science and engineering courses provide an animated simulation focusing primarily on the instruction flow

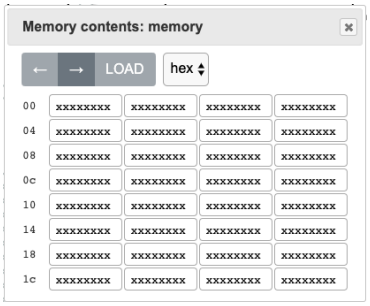


Figura 3.3: A memory window in the extended *DigitalJS* with load/save file options.

inside the datapath block diagram. This approach gives students fundamental theory but not practical implementation experience of the Verilog code. In our approach, the student should modify and simulate the code with the help of the structural viewer and the signal waveforms. Also, the graphic interface provides a visualization of the register/memory values. Moreover, the tool supports the addition of signal output displays.

3.3.1 Additional Features

We redesign the *DigitalJS* in a minimalist interface-based approach to customize for processor implementation. First, the student can load all MIPS datapaths presented in Patterson/Hennessy's book. We modify the *DigitalJS* tool to show the structural design examples in a visually organized fashion. Besides, the code is also modular to allow the student to navigate into the hierarchical structure. Figure 3.2 displays the five-stage MIPS pipeline. We implement one module for each stage. The student can visualize the signal interface between the stages. By selecting one or more module stages, the student can visualize the internal structure.

Figura 3.4: Monitoring five signals in the waveform window.

Moreover, since the Verilog code is required, the students can visualize, edit, and simulate the design. The students can also share the link to the new design. In contrast to the book datapath, the code and the structural viewer include all wires and modules required to simulate the processor. A modular decomposition shows the task breakdown from the high level to the detailed level.

Second, we extend the register bank and memory components to allow load/store program/data files to provide code examples, as shown in Figure 3.3. The students can execute the simulation at a given clock rate or step-by-step and select wires at all

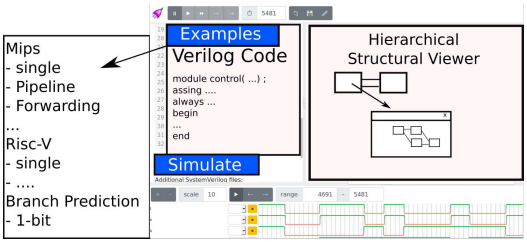


Figura 3.5: A simplified view of the extended *DigitalJS* with examples.

hierarchical levels for waveform visualization. Figure 3.4 depicts the PC value and the instruction code in hexadecimal, and the R_s,R_t and R_d instruction fields. Third, we add the book exercises and RISC-V and branch prediction implementations, as described in the following sections. Figure 3.5 depicts our MIPS *digitalJS* extension.

3.3.2 RISC-V

We also provide all datapaths for the RISC-V edition (Patterson and Hennessy, 2017). Moreover, we add two RISC-V example sections: structural hierarchical and book implementation. The Verilog code for the structural hierarchical is modularized, for many reasons, including making it easier to understand, change, and verify. The book implementation section has the Verilog codes presented in book section 4.13 (Patterson and Hennessy, 2017). We propose some labs like book exercises 4.11, 4.12, 4.13 (Patterson and Hennessy, 2017) to add a new instruction and modify the Verilog code, which requires to add new functional blocks, modify the existing functional blocks, add new datapaths and control signals to support the new instruction. The student can also extend the current design, and for instance, solve the exercise 4.23, where the load/store instructions use a register (without an offset) as the address. Thus, the new design overlaps the memory and execution stages resulting in a four-stage pipeline.

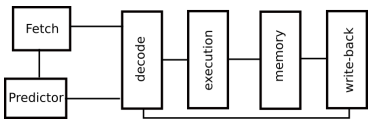


Figura 3.6: Block diagram of MIPS pipeline with branch predictor.

3.3.3 Branch Prediction

There is a lack of branch predictor design details in reference books (Patterson and Hennessy, 2013, 2017) compared to design the basic and advanced datapath with forwarding and hazards. Recent works (Kocher et al., 2018; Lipp et al., 2018) reveal new processor vulnerabilities, such as Meltdown and Spectre, where the branch

predictor and the cache are exploited in Spectre variant 2 (Kocher et al., 2018). We propose to teach branch prediction examples: 1-bit, two-bits, and two-level predictor by providing a structural Verilog description. The implementation is modular, as shown in Figure 3.6. Therefore, the student can read and implement the classical approaches (Yeh and Patt, 1992; Smith, 1981; Mutlu et al., 2016) by extending the MIPS/RISC-V design, from a high-level to a detailed level view, focusing on the signals and modules required to implement the predictors.

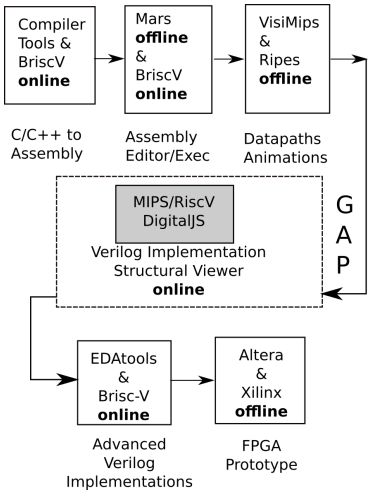


Figura 3.7: A set of tools to teaching MIPS/RISC-V processors.

3.4 Complete Framework

We also propose a set of tools to cover from the introduction, assembly language, and general control/datapath architecture to the Verilog design and programming of all MIPS implementations. Figure 3.7 depicts a sequence of tools. Starting from a C/C++ code, it is possible to generate MIPS and RISC-V assembly. We propose to use online Compiler Explorer (godbolt.org, 2020) and Brisc-V (Agrawal et al., 2019), which are both online tools and have color highlighted editors. For assembly, we suggest Mars (Vollmar and Sanderson, 2006b) and Brisc-V (Agrawal et al., 2019) for MIPS and RISC-V, respectively. Mars is an offline simulator. However, it is widely used and well established. Regarding the MIPS datapath simulation, Mars X-Ray plugin(Araujo et al., 2014) can be used for the single cycle datapath, and Visimips (Kabir et al., 2011) for the pipeline datapaths including forward and hazard units. Visimips (Kabir et al., 2011) is also an offline tool, however, it is a single Java file, and it is very intuitive and minimalist. For a more versatile tool including all book datapaths without forwarding and hazard units, we suggest DRmips (Nova et al., 2013). Since Brisc-V (Agrawal et al., 2019) has no datapath visualization, and we propose Ripes (Mortbopet, 2020) to animate the RISC-V datapaths.

Our approach fills the gap from the simplified datapath block diagram to the Verilog codes. Our *digitalJS* extension allows performing a step further towards a Verilog MIPS/RISC-V implementation. To the best of our knowledge, there is a lack of tools that provide a complete and straightforward Verilog code to simulate MIPS online, and at the same time, depict a structural and hierarchical design viewer. Furthermore, students can go deeper into the learning design, and we suggest Brisc-V (Agrawal et al., 2019), which has a Verilog parameterized code generator, including multiple processors and cache coherency. We also recommend EDA tools (playground, 2020) with online commercial Verilog simulators for MIPS design before moving to Altera/Xilinx tools to generate an FPGA prototype.

3.5 Conclusion

Students need to learn the fundamental characteristics of RISC processor design and the properties, not only for its usage in computing architecture but also for understanding simple, useful, and powerful fundamentals as a motivation for further research work. We present a compact browser-based editor/simulator (UFV, 2020) with all the necessary tools required for RISC processor design as well as Verilog implementation. Our approach can be useful for the labs in numerous courses. The Verilog descriptions still impose challenges since even a simple design should have all signals/modules. We believe that this work will be a useful resource both in support of processor design teaching and as a simple online tool. Future work includes new examples and execution in an FPGA in the cloud.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brasil (CAPES)– Finance Code 001. We want to thank FAPEMIG and CNPq for financial support.

Capítulo 4

Plain: Ferramenta para Desenvolvimento de Aceleradores para Overlays em FPGA na Nuvem em Tempo de Execução

Os FPGAs oferecem eficiência energética para o desenvolvimento de aceleradores para fluxo de dados na Nuvem. Porém, existem vários desafios para popularizar seu uso. Dentre eles, podemos citar o tempo de compilação (que pode demorar horas) e conhecimento de hardware para uso adequado de linguagens de síntese de alto nível. Recentemente, a ferramenta READY possibilitou a redução do tempo de compilação e configuração para microsegundos. O ambiente foi validado na plataforma em nuvem HARP 2 da Intel/Altera. Apesar da integração com a Linguagem C++ para o desenvolvimento das aplicações, o acelerador é descrito de forma textual como um grafo. Neste trabalho é apresentado a extensão PLAIN, que inclui uma interface *online* gráfica para descrição dos aceleradores, a automatização do fluxo de projeto, dois níveis de simulação e um nível de execução. A ferramenta também mostra estatísticas de desempenho e permite criação de novos operadores para exploração do espaço de projeto.

4.1 Introdução

A modelagem de um algoritmo com um grafo de fluxo de dados torna explícita a descrição do paralelismo. Além disso, o mapeamento do grafo diretamente no *hardware* permite eliminar que a busca e decodificação das instruções sejam realizadas a cada ciclo, economizando energia. Os dados irão fluir pela estrutura, gerando reuso e reduzindo as operações com estruturas temporárias de memória. Portanto, os grafos de fluxo oferecem eficiência energética e desempenho.

[0]Financiamento: FAPEMIG, CNPq, Nvidia, Funarbe. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Apoio dos laboratórios: Intel Academic Compute Environment e Paderborn Center for Parallel Computing.

O custo de projetos em *ASIC* (circuitos dedicados) é elevado para mapeamento dos grafos de fluxo. Uma alternativa é o uso de hardware reconfigurável como os FPGAs (C. Penha et al., 2019). Porém, o desenvolvimento com FPGA ainda exige um conhecimento detalhado do hardware e de uma linguagem de descrição de *hardware*, normalmente Verilog ou VHDL. Além disso, o processo de síntese (ou compilação) pode levar várias horas. Duas alternativas podem ser usadas: (a) linguagens de alto nível; (b) camadas virtuais ou *Overlays*.

Como linguagens de alto nível podemos destacar HLS baseadas em C/C++ (*High Level Synthesis*) (Nane et al., 2015), OpenCL (Krommydas et al., 2016) e Maxeler (Stanojević et al., 2019). HLS ou OpenCL tem a vantagem da programação em alto nível em C/C++. Porém, o grafo de fluxo de dados que irá implementar o hardware é gerado de forma implícita, sem muito controle do programador e pode não ser eficiente. Portanto, o programador precisa ter conhecimentos sobre o estilo de código que deve usar, diretivas e o processo de compilação HLS. A abordagem da Maxeler (Stanojević et al., 2019) utiliza a linguagem Java como uma extensão onde o programador descreve explicitamente o grafo de fluxo de operações. O código é compilado e sintetizado para FPGA. Entretanto, qualquer modificação requer que o projeto seja re-sintetizado e todo o FPGA deve ser reprogramado.

A segunda abordagem evita a re-síntese e a reprogramação do FPGA com *Overlays* (Nane et al., 2015), que em geral implementam um CGRA (*Coarse-Grained Reconfigurable Arrays*) como uma camada virtual sobre o FPGA. Os CGRAs são reconfigurados a nível de instrução, simplificando o processo de mapeamento. O CGRA é configurado apenas uma única vez sobre o FPGA. Posteriormente, o compilador precisa apenas gerar código para CGRA. Entretanto faltam ferramentas para compilação e execução de aplicações CGRA mapeados em FPGAs.

Desse modo, este artigo propõe o uso combinado das duas abordagens. Primeiro, a aplicação é integrada em um código alto nível escrito em C/C++, semelhante a abordagem CUDA de GPUs. Diferente de OpenCL/HLS, o grafo de fluxo de dados é explicitamente descrito para especificação do algoritmo (ou *kernel*) que o acelerador irá executar, semelhante a abordagem da Maxeler (Stanojević et al., 2019). Entretanto, não é necessário recompilar para FPGA a cada modificação, pois é feito o uso de um *Overlay*. Finalmente, todo o fluxo de projeto foi automatizado, sendo transparente para o programador. O grafo especificado irá executar em um FPGA na nuvem. A implementação foi realizada fazendo uma extensão da ferramenta READY (Silva et al., 2019), denominada PLAIN, que encapsula e automatiza todo fluxo, inclui uma interface gráfica para reduzir a curva de aprendizagem, dois modos de simulação que permite a exploração de novos operadores de maneiras distintas, apresenta relatório da execução de forma simplificada e ainda permite a criação de novos operadores para futuras expansões.

A estrutura deste artigo é descrita a seguir. A Seção 4.2 descreve o grafo de fluxo de dados. A Seção 4.3 apresenta a ferramenta READY com suas especificações, detalhes e funcionamento. A Seção 4.4 apresenta a ferramenta PLAIN e as Seções 4.5 e 4.6 realizam uma comparação com outras ferramentas. Por fim, a Seção 4.7 discute as principais conclusões e trabalhos futuros.

4.2 Grafos de Fluxo de Dados

Um grafo de fluxo de dados apresenta o paralelismo de uma forma explícita no nível de instruções e pode conter operadores com fluxo de controle (Ferreira et al., 2004; Nowatzki et al., 2017). Um conceito importante é a temporização para saber qual elemento da sequência está sendo trabalhado em cada nível. A vazão sempre será máxima e o número de operações executadas pelo grafo por ciclo é exatamente o número de operadores do grafo, ou seja, a cada ciclo todas as operações são executadas em paralelo.

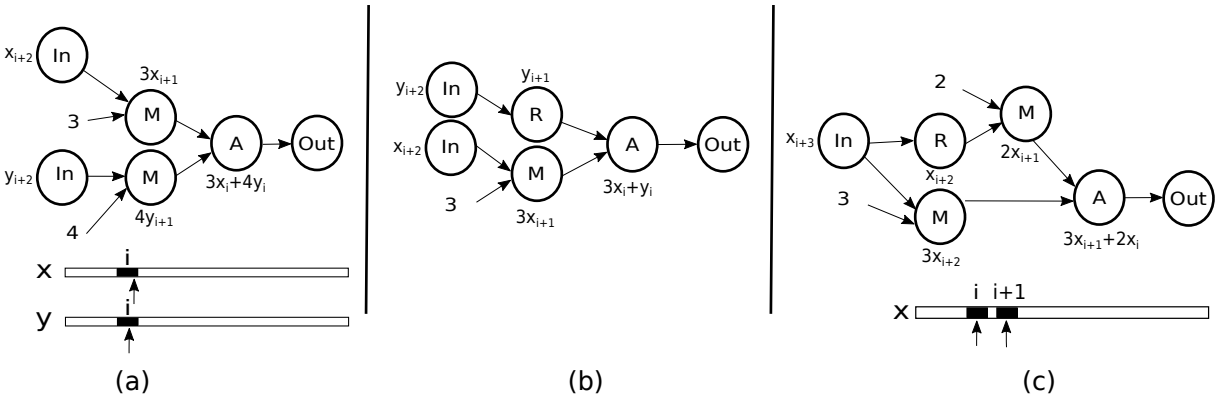


Figura 4.1: (a) Grafo com uma soma de vetores; (b) Soma de vetores que requer balanceamento; (c) Convolução com dois elementos consecutivos de um vetor.

A Figura 4.1 apresenta três exemplos. Primeiro, apresenta uma computação simples $out = 3 \cdot x_i + 4 \cdot y_i$ na Figura 4.1(a) onde os operadores m fazem multiplicações e o operador a faz uma adição. A cada ciclo, novos elementos de x e de y são inseridos no grafo mantendo o pipeline sempre cheio. O segundo exemplo faz a computação $out = 3 \cdot x_i + y_i$. Neste caso é necessário inserir um registrador (operador R) para equalizar os caminhos, garantindo que os elementos do vetor chegarão de forma sincronizada em a. Supondo que cada operação tem a latência de 1 ciclo, o balanceamento poderia ser automático, porém nas ferramentas apresentadas neste artigo (READY e PLAIN), o balanceamento deve ser explícito para permitir operações do tipo convolução com várias aplicações em aprendizado de máquina. O último exemplo calcula $out = 3 \cdot x_{i+1} + 2 \cdot x_i$. Como os caminhos estão desequilibrados, gerou-se computações com elementos distintos do vetor. Neste exemplo, há dois elementos

consecutivos. A linguagem MAXELER (Stanojević et al., 2019) oferece um operador para implementar o elemento posterior e anterior em uma sequência.

A ferramenta PLAIN, proposta neste trabalho, permite a visualização da sequência de dados fluindo pelo grafo na fase de desenvolvimento do algoritmo. O programador pode implementar um trecho do grafo, injetar valores e verificar se o fluxo e o balanceamento estão corretos. A Seção 4.4.3 apresenta este modo de simulação que é útil na fase inicial do projeto.

4.3 Ferramenta READY

A ferramenta READY (Silva et al., 2019) permite a descrição de um ou mais grafos de fluxo e seu acoplamento com um código C++. O grafo é mapeado em tempo de execução em um *overlay* CGRA que executa como um acelerador no FPGA. O CGRA utiliza uma rede multiestágio que permite uma implementação eficiente de posicionamento e roteamento do grafo de fluxo de dados (da Silva et al., 2017). A ferramenta possui também uma API para execução de múltiplos grafos (Silva et al., 2019). Os grafos podem ser réplicas do mesmo grafo ou grafos diferentes. Semelhante a uma GPU, múltiplos grafos (ou threads) são usados para esconder a latência do CGRA e garantir a vazão máxima no mapeamento dos grafos na arquitetura física.

4.3.1 Especificação

A Figura 4.2(a) representa a fase de compilação e a Figura 4.2(b) a fase da execução da ferramenta READY. A nova extensão proposta neste trabalho inclui a etapa (1), onde o grafo de fluxo é descrito de forma gráfica ou textual (ver Seção 4.4). Na ferramenta original o grafo deve ser incluído dentro do código C++ com uma descrição textual.

O grafo descrito na ferramenta PLAIN é transformado em um formato intermediário JSON. Trabalhos futuros poderão incluir a geração do grafo por compiladores e uso de DSL (linguagens de domínio específico). Na etapa (2), o JSON é carregado como um grafo de fluxo de dados pela biblioteca construída do READY. A ferramenta PLAIN agregou mais portabilidade onde a descrição JSON também pode ser carregada em tempo de execução. Posteriormente, o mapeamento é executado com uma implementação com complexidade polinomial em tempo de execução (Silva et al., 2019), gerando a configuração para o CGRA.

A configuração é transferida para o CGRA na etapa (3). As variáveis das entradas e saídas fazem o acoplamento entre o código C++ e o grafo. Na etapa (4) começa a execução com sobreposição do envio/recebimento dos dados e da computação no acelerador CGRA.

A ferramenta READY possibilita a execução síncrona ou assíncrona. Similar ao

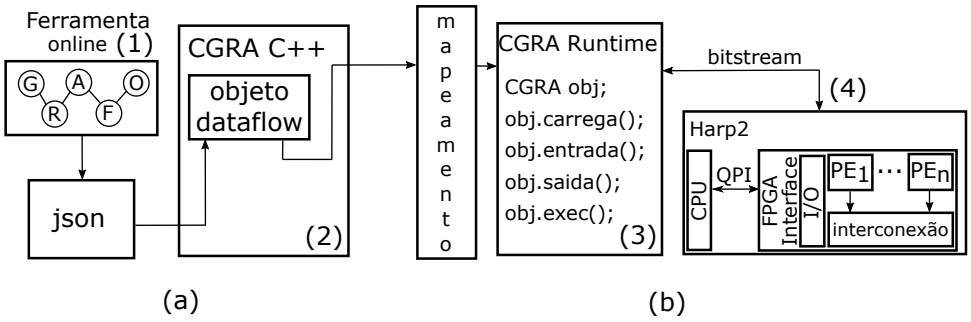


Figura 4.2: Etapas de (a) compilação (b) execução da ferramenta Ready

SDK Cuda da Nvidia (Nickolls et al., 2008), o grafo pode ser executado de maneira síncrona, bloqueando o processador até que a tarefa da aceleração esteja concluída ou executando de forma assíncrona, sobrepondo as tarefas do processador e do acelerador. Vale ressaltar que o sistema inicia enviando as palavras de configuração pelo canal de dados para configurar o acelerador. Uma vez configurado o CGRA, a execução começa assim que os primeiros dados são transmitidos da CPU para o acelerador. No caso de falha na cache, a entrada do grafo fica bloqueada aguardando os dados solicitados.

Outra contribuição da ferramenta READY é o encapsulamento da comunicação com FPGA da Intel/Altera. Primeiro, a API OPAE (Cross-Platform and Developers, 2013) da Intel/Altera foi encapsulada na camada superior do programa. OPAE é uma API desenvolvida inicialmente pela Intel/Altera e aberta para a comunidade com o objetivo de simplificar a integração de vários tipos diferentes de dispositivos aceleradores de FPGAs. Uma nova API SW/HW foi desenvolvida sobre a plataforma de pesquisa em arquitetura heterogênea Intel/Altera (HARP 2). Esta API pode ser ampliada para outros ambientes de FPGA na nuvem.

4.3.2 Execução

A ferramenta READY possui suporte para o HARP 2, disponível na Nuvem da web pela Intel Research (Intel, 2020), permitindo que projetos para FPGA possam ser executados *online*. Diferentemente das instâncias EC2 F1 da Amazon que são comerciais, a plataforma web do HARP 2 é voltada para centros de pesquisas. Outro diferencial do HARP é o acoplamento entre o FPGA Arria 10 de modelo 10AX115U3F45E2SGE3 com o processador Xeon E52680 2.4 GHz com 14 núcleos e 24 MBytes de cache L3 acoplado na FPGA via um barramento 2-PCI e um Intel QuickPath Interconnect (QPI) com a abordagem de memória compartilhada com cache transparente.

4.4 Ferramenta PLAIN

A ferramenta PLAIN foi construída para simplificar o desenvolvimento de aceleradores com grafos de fluxo de dados. Como a ferramenta executa na Web, permite ao usuário gerar toda a estrutura sem a necessidade de conhecimento de FPGA/CGRA. Portanto, reduz a curva de aprendizagem, cria um ambiente no navegador para desenvolvimento de algoritmos com execução em FPGAs na nuvem e ainda permite extensões que serão apresentadas a seguir.

A Figura 4.3 mostra a implementação de filtro de impulso finitos (FIR4) onde seu paralelismo é explicitado com o uso da ferramenta. Este grafo calcula uma convolução $out = x_i + 2 * x_{i+1} + 3 * x_{i+2} + 4 * x_{i+3}$, onde o índice i representa o i-ésimo elemento da sequência.

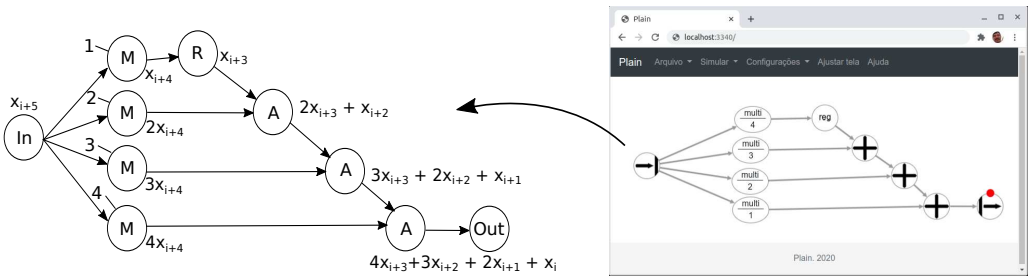


Figura 4.3: (a) Grafo de fluxo FIR4; (b) Representação do FIR4 na ferramenta PLAIN.

Além das facilidades de edição gráfica dentro de um navegador, sem a necessidade de instalação de ferramentas para FPGA, a ferramenta também permite ao usuário a criação dos seus próprios operadores. Este recurso é importante para definição de quais os operadores o CGRA deve implementar. Apesar de os primeiros CGRA terem sidos definidos há mais de duas décadas (Akram and Sawalha, 2019), ainda não existe um consenso sobre o conjunto de operadores. Por exemplo, para códigos x86 é conhecido que 90% do código é mapeado em apenas 25 *opcode* ou operações básicas (Mutigwe et al., 2013). No modelo de computação espacial e com as novas cargas de trabalho de aprendizado de máquina, quais são os operadores que um CGRA deve conter? Pensando neste aspecto, a ferramenta PLAIN possibilita a criação e validação de novos operadores que será detalhada na Seção 4.4.2. O usuário pode fornecer uma descrição comportamental do novo operador em C++ para validação da execução e/ou descrição comportamental/estrutural em Verilog para acoplamento em um novo CGRA. Esta modificação é implementada com inclusão de metadados na descrição JSON com os trechos de códigos.

4.4.1 Descrição do Grafo

A estrutura utilizada para criação do modelo do grafo foi o tipo JSON (JavaScript Object Notation) (Bray et al., 2014), sendo escolhida por ser uma estrutura aberta permitindo que sejam adaptado para o funcionamento em diversos sistemas. A partir desta estrutura foram definidos um conjunto de nodos, divididos em tipos lógicos (*and*, *or*, *mux* e *not*) e tipos aritméticos (*sub*, *subi*, *add*, *addi*, *mult* e *multi*). Além disso, possui nodos de entrada (*in*) e saída (*out*). Esse subconjunto de operadores predefinidos possibilita ao usuário realizar uma superposição dos mesmos para construção de um operador mais complexo.

Para o desenvolvimento da interface foi utilizado o Cytoscape JS (Franz et al., 2016) que é uma biblioteca JavaScript que permite a visualização/manipulação de grafos. Ela funciona no navegador e pode ser associado com ferramentas como o Node JS para permitir o processamento dos dados em um servidor. Além das funcionalidades de edição (adição e remoção de arestas), o Cytoscape JS oferece vários algoritmos de teoria de grafos que podem ser utilizados para análise de métricas como histograma de grau de entrada e saídas, algoritmos de centralidade, *betweeness*, *pagerank*, dentre outros. Além disso, a ferramenta PLAIN utiliza o *Webpack* para que tudo funcione só no navegador sem que seja necessário o uso de servidores ou algum *plugin* extra, independente do sistema operacional.

A tela principal da ferramenta dispõe de um espaço de desenho (área de trabalho) onde o usuário faz a edição do grafo. Além disso é possível realizar a simulação, criação de operadores, execução no FPGA e visualização de métricas após a execução.

4.4.2 Novos Operadores

É possível estender a ferramenta com a inclusão de novos operadores ou a importação de módulos que podem ser utilizados para criar grafos maiores. A Figura 4.4 mostra um exemplo de desenvolvimento de um novo operador. Primeiro, o novo operador é criado definindo um símbolo e suas funcionalidades (entradas, saídas, operação ou metadado em C++ com a descrição comportamental). As operações simples como soma, multiplicação ou pequenas combinações pode ser diretamente inseridas na descrição. Uma operação mais complexa pode ser descrita em C++ ou Verilog. Segundo, se o operador foi inicialmente definido em C++/Verilog pode ser acoplado ao código C++ da aplicação e simulado.

Como já foi demonstrado na ferramenta READY para aplicação K-Means (Silva et al., 2019), ao criar um operador complexo, o desempenho do CGRA passa de 25,6 Gops/s para 100 Gops/s. Pois cada operador complexo pode implementar cerca de 20 operações básicas. A Figura 4.4(a) ilustra um exemplo de um novo operador FIR4 que foi usado em cascata para aplicar três convoluções em sequência seguido de

dois operadores de uso geral. A Figura 4.4(b) mostra a implementação do operador na arquitetura *overlay*. Além do desempenho, operadores com maior granularidade economizam recursos de interconexão. A implementação atual do *overlay* possibilita no máximo 256 conexões entre os operadores. Ao usar um operador de granularidade maior, as conexões internas não fazem uso da rede, sobram mais recursos para a interconexões dos outros operadores. Ao mesmo tempo, os operadores simples e universais deixam a ferramenta flexível para descrição de um domínio maior de aplicações.

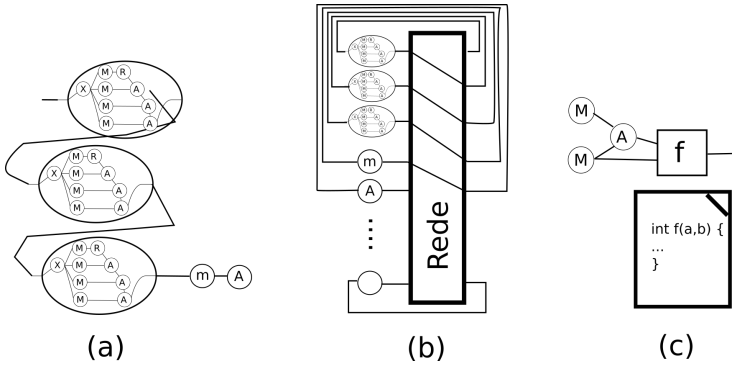


Figura 4.4: (a) Exemplo de um novo operador; (b) Uso do operador no *Overlay*; (c) Exemplo de um novo operador f com metadado para sua funcionalidade.

O mesmo princípio foi utilizado pela Nvidia com os operadores *Tensor cores*. Cada operador tensor encapsula mais de 100 multiplicações e somas em pipeline. Por isto, uma GPU V100 com 640 Tensores (1,5 Ghz) faz $640 \times 1,5G * 100 = 96$ Tera ops/s.

Como novos operadores é uma área em estudo, a ferramenta PLAIN permite descrever o comportamento do operador com um código C++ através da interface do navegador. Figura 4.4(c) mostra um exemplo de novo operador com descrição por código.

4.4.3 Simulação

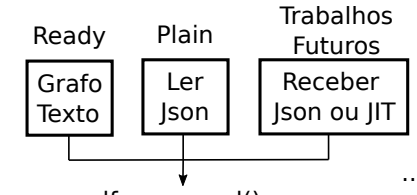
A ferramenta PLAIN possibilita dois níveis de simulação. Primeiro, o grafo pode ser simulado com a injeção de pequenas sequências de valores pela interface gráfica. Se um novo operador for criado, este recurso também pode ser usado pois o comportamento do operador estará descrito com operadores já existentes (subgrafo) ou através de um código C++.

O segundo nível de simulação emula o funcionamento do sistema como um todo, incluindo FPGA. O grafo será alimentado com os dados da aplicação completa descrita em C++. Todo o processo de mapeamento será simulado a nível funcional incluindo as interfaces de I/O e a execução no *overlay*. Entretanto, devido ao nível de detalhamento, a simulação demanda mais tempo para executar.

4.4.4 Execução

O diferencial das ferramentas PLAIN e READY é o mapeamento do grafo no CGRA em tempo de execução, ou seja de forma dinâmica, além de possibilitar a reconfiguração e execução do acelerador também de forma dinâmica. Este recurso permite a futura criação de compiladores Just-in-Time (JIT). O PLAIN acrescenta o retorno de informações para o usuário sobre a execução: número de ciclos, número de bytes transmitidos, tempo total, dentre outros. Os dados são referentes a execução no FPGA incluindo a transferência de dados e configuração do CGRA. Ou seja, da mesma forma que o usuário executa a nível de simulação, ele pode executar com um volume maior de dados no modo de execução.

A Figura 4.5 mostra dois momentos distintos da execução. No primeiro trecho o grafo é mapeado no objeto CGRA. O grafo pode estar descrito na ferramenta PLAIN ou gerado por um compilador JIT. No segundo trecho do código em C++, o objeto CGRA carrega o grafo já mapeado com o método *loadCGRAprogram*. Além disso é necessário acoplar os dados do processador com as entradas do grafo. Neste exemplo os vetores (a, b, c) são inicializados no processador. Depois com os métodos *SetCGRAInputStream* e *SetCGRAOutputStream*, os vetores de dados são acoplados ao acelerador. Vale destacar a facilidade de associar os dados da sua aplicação C++ para enviar/receber dados no acelerador. Finalmente, o método *syncExecute* realiza duas operações. Primeiro, o CGRA é configurado com a transmissão do bitstream para o *overlay* no FPGA. Depois automaticamente, o CGRA começa a requisitar, processar e enviar os resultados através da memória compartilhada entre o processador e o acelerador.




```
map = df->mapped();                    ...
...                                    cgra.loadCgraProgram(map);
Cgra cgra;
int a[1024], b[1024], c[1024];         cgra.setCgraInputStreamByID(df->id, df->in0.id, a, len);
int len = 1024 * sizeof(short);        cgra.setCgraInputStreamByID(df->id, df->in1.id, b, len);
for (int i = 0; i < len; ++i)          cgra.setCgraOutputStreamByID(df->id, df->out.id, c, len);
   a[i] = 0; b[i] = 0; c[i] = 0;
...                                    cgra.syncExecute(0); // Execução
                                       ...
```

Figura 4.5: Etapa do mapeamento com um exemplo de um trecho do código em C++ gerado automaticamente para executar o CGRA.

4.5 Resultados

A ferramenta PLAIN amplia e simplifica a ferramenta READY. O resultado final traz vantagens no processo de desenvolvimento de soluções de aceleradores em hardware modelados com grafos, pois todo o processo de elaboração e síntese foi abstraído em um processo automatizado com edição gráfica em um navegador e execução em hardware com CGRA *overlay* já carregado no FPGA. Reduzindo assim os esforços necessários para elaboração de soluções de aceleradores em hardware acoplados a processadores de alto desempenho.

A ferramenta PLAIN exibe um relatório de execução. A Figura 4.6 ilustra o relatório do exemplo da aplicação *Kmeans* e a Tabela 4.1 mostra o desempenho da execução de 4 exemplos em comparação com a execução em um Processador XEON com 1 e com 8 *threads* respectivamente. O CGRA pode executar até 128 operações por ciclo, explorando paralelismo espacial e temporal. Portanto, aplicações como o FIR e KMEANS, que são grafos com um grande número de operadores e reuso dos dados, terão mais desempenho. Se o grafo tiver poucas operações ou for limitado pelo baixo reuso dos dados, o CGRA terá uma baixa ocupação, resultando em uma taxa menor de Gops/s. Todos os exemplos da Tabela 4.1 tem o mesmo tamanho de dados na entrada e portanto, o tempo de execução é bem próximo.

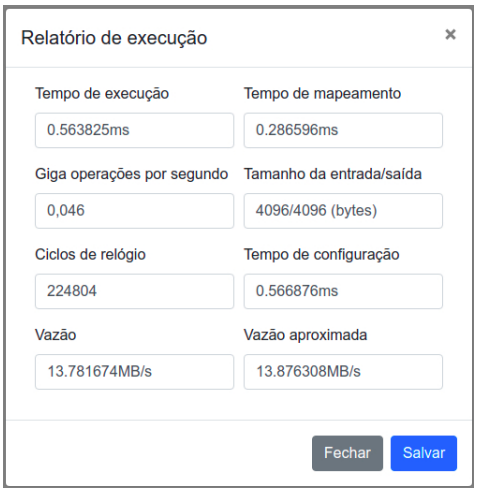


Figura 4.6: Relatório de Execução da Ferramenta PLAIN para aplicação FIR.

Para comparar o tempo de desenvolvimento e projeto com as abordagens tradicionais com síntese de alto nível, as 4 aplicações foram descritas em C++ na ferramenta HLS da Intel e compiladas com a versão 19.4. O Verilog gerado foi sintetizado com Quartus 19.4. A Tabela 4.2 mostra o tempo de compilação (C++ $\rightarrow$ Verilog) e o tempo de síntese (Verilog $\rightarrow$ FPGA) e os recursos utilizados. Podemos observar uma diferença significativa nos tempos de compilação e síntese em comparação com Ferramenta PLAIN variando de 5 a 6 ordens de grandeza, que são apresentados na coluna

Tabela 4.1: Tempo de Compilação e Reconfiguração para as 4 aplicações.

Bench	Tempo Compilação+Configuração (µs)	Tempo de Execução CGRA (ms)	CGRA Gop/s	Tempo de Execução Xeon (ms)		Xeon Gop/s
				1 Thread	8 Threads	
fir	778,6	42.9	24	360.8	84.9	11,9
kmeans	793,6	42.3	16,4	285.4	72.5	9,5
paeth	792,2	42.5	7,2	50.0	17.5	17,8
sobel	784,4	42.3	7	277.8	68.2	4,3

ganho da Tabela 4.2. O *ganho* é a razão do tempo gasto para compilar/sintetizar em HLS comparado com o tempo para compilar e reconfigurar no PLAIN. A última linha mostra os recurso gastos pelo CGRA. Apesar de consumir 46% do FPGA, além de ganhar no tempo de mapeamento, o tempo de execução é bem próximo pois o CGRA executa em pipeline com uma frequência de 200 Mhz. O CGRA é compilado uma única vez em tempo de projeto, portanto, esta operação é transparente para o usuário e não afeta o tempo de compilação e execução dos novos algoritmos que podem ser avaliados.

Tabela 4.2: Benchmarks no Intel HLS: Tempo de Compilação e Síntese, Recursos gastos no FPGA. Comparação com o CGRA *overlay*.

Bench	Tempo Comp.	Tempo Sintese	Recursos do FPGA				Ganho
			ALMs	BRAM(bits)	DSPs	FMax(MHz)	
fir	22s	14m33s	20.838 (5%)	2.048 (<1%)	20 (1%)	196	9×10^5
kmeans	6m23s	1h21m	139.776 (33%)	1.783.232 (3%)	0 (0%)	192,94	5×10^6
paeth	34s	16m02s	24.236 (6%)	547.968 (<1%)	0 (0%)	232,56	1×10^6
filtro Sobel	40s	30m52s	57.544 (13%)	698.240 (1%)	96 (6%)	237,42	2×10^6
CGRA	-	3h19m23s	195.532 (46%)	5.892.848 (11%)	128 (8%)	200	-

4.6 Trabalhos Relacionados

A Tabela 4.3 resume a comparação com outras ferramentas e abordagens para desenvolvimento de aceleradores. A coluna *Comp* exige a ordem de grandeza para compilação, assim como as colunas *Config* para a configuração dinâmica e *Sint* para síntese. A coluna *Nav* se a ferramenta oferece suporte para navegador. A coluna *overlay* se executa como *Overlay* em um FPGA e a coluna *Cloud* se executa na nuvem. A coluna *DFG* mostra se o grafo é explícito ou implícito. Finalmente a coluna *D/E* se

a ferramenta oferece recursos gráficos para depuração e relatórios de execução para refinamentos.

Primeiro, a ferramenta ADD (C. Penha et al., 2019) permite a descrição de grafos de forma explícita, porém gera um código Verilog que deve ser sintetizada, que pode demorar horas, para ser depois implementado no FPGA. O compilador CCF (Dave and Shrivastava, 2017) não requer síntese, extraí o grafo de um código C/C++, porém requer minutos para compilar e não foi validado em FPGA. A execução foi validada em um processador ARM utilizando a ferramenta de simulação Gem5. No PLAIN para o mesmo benchmark, o tempo de mapeamento é da ordem de milissegundos.

Já a ferramenta CGRA-ME foi integrada a um gerador de *overlay* por (Chin et al., 2018) para executar sobre um FPGA, mas o trabalho não apresenta medidas de tempo de execução. A compilação demora minutos/horas por usar programação inteira com soluções exatas, além de ser restrita a grafos com no máximo 25 operadores. O CGRA alvo tem apenas 16 elementos de processamento, portanto a 200 MHz só irá ter no máximo 200M*16 = 3,2 Gops/s de desempenho, que é 8x menor que o desempenho do PLAIN que requer apenas milissegundos para mapear/compilar. Por fim, nenhuma destas ferramentas tem suporte para execução na nuvem e nem facilidades de desenvolvimento com implementações em navegadores. A segunda parte da Tabela 4.3 mostra soluções de ferramentas comerciais. Primeiro, o uso de linguagem de síntese de alto nível com HLS da Intel, HLS da Xilinx e a ferramenta Legup (Nane et al., 2015). A principal vantagem é o uso de C/C++ para descrição. Entretanto existe um abismo entre o código e a implementação, que exige conhecimentos de hardware do programador para escrever um bom código. Além disso, todas tem um alto custo para compilação e ajustes finos que podem requerer horas ou dias. A abordagem com OpenCL também compartilha estas vantagens e desvantagens. Outro ponto que o grafo do acelerador é gerado de forma indireta ou implícita. Tanto a Xilinx como a Intel oferecem ferramentas para visualização do grafo gerado, estimativas de uso de recursos do FPGA e da memória com interface gráfica em suas ferramentas de projeto. Porém, caso haja a necessidade de alteração no grafo, só é possível por meio de mudança no código e, desta maneira indireta, depende do compilador para regerar o grafo e assim prosseguir na ferramenta. Por outro lado, a abordagem PLAIN/READY trabalha explicitamente com os grafos, podendo modificá-lo facilmente executando em um navegador.

A Maxeler (Stanojević et al., 2019) permite a descrição explícita porém a etapa de síntese é necessária e pode demorar minutos/horas. Ela trabalha com o grafo semi-explícito, ou seja, o grafo é descrito junto com o código com uma extensão de JAVA. Para modificações existe a necessidade de recompilar todo o processo. A Maxeler também oferece uma ferramenta para desenvolvimento com interface gráfica e visualização dos grafos gerados.

Portanto, a ferramenta READY reúne vantagens das abordagens anteriores com mapeamento e configuração em milissegundos, uso transparente do FPGA sem necessidade de síntese e descrição explícita. A contribuição da ferramenta PLAIN foi simplificar ainda mais o processo, encapsular o ambiente para executar em um navegador, permitir a exploração de novos operadores com dois modos de simulação e gerar relatórios de execução real (incluindo tempos de configuração do FPGA, considerando transmissão de dados, falhas de cache, etc.) para refinamento do algoritmo do acelerador.

Tabela 4.3: Comparação de características entre as ferramentas.

Ferramentas	Comp	Config	Nav	Sint	overlay	D/E	Cloud	DFG
ADD	seg-min	-	-	min/hs	-	sim	-	E
CCF	min-hs	-	-	min/hs	-	-	-	I
CGRAME	min-hs	-	-	hs	sim	-	-	E
HLS	seg	-	-	min/hs	-	sim	sim	I
OpenCL	seg	-	-	hs	-	sim	sim	I
Maxeler	seg	-	-	min/hs	-	sim	sim	E
Ready	ms	ms	-	ms	sim	-	sim	E
Plain	ms	ms	sim	ms	sim	sim	sim	E

4.7 Conclusão

O desenvolvimento de aplicações com FPGAs ainda tem muitos desafios. A ferramenta PLAIN apresentada neste trabalho simplifica o ensino e pesquisa na construção de algoritmos modelados com fluxo de dados para execução em aceleradores com FPGA. O uso de um CGRA como *overlay* reduz o tempo de compilação e evita a reprogramação do FPGA. Atualmente, o uso de FPGA em nuvem apresenta um novo desafio de segurança. Neste aspecto, o *overlay* isola o acesso físico ao FPGA. Como trabalhos futuros, um ponto importante é o conjunto de operadores e operadores de domínio específico para obter desempenho com eficiência energética. O FPGA HARP 2 que foi avaliado tem o consumo de 23 Watts. Como novos operadores pode-se obter de 100 a 200 Gops/s, ou seja, uma eficiência de até 10 Gops/W.

Referências Bibliográficas

Agrawal, R., Bandara, S., Ehret, A., Isakov, M., Mark, M., and Kinsy, M. A. (2019). The brisc-v platform: A practical teaching approach for computer architecture. In *Proceedings of the Workshop on Computer Architecture Education*, page 1. ACM.

Akram, A. and Sawalha, L. (2019). A survey of computer architecture simulation techniques and tools. *IEEE Access*.

Ali, F. (2015). Teaching the internet of things concepts. In *Proceedings of the WESE'15: Workshop on Embedded and Cyber-Physical Systems Education*, pages 1–6.

Araujo, M. R. D., Padua, F. L. C., Andrade, F. V., and Correa-Junior, F. L. (2014). Mips x-ray: A mars simulator plug-in for teaching computer architecture. *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, 2(2):36–42.

Bansal, N., Gupta, S., Dutt, N., Nicolau, A., and Gupta, R. (2004). Network topology exploration of mesh-based coarse-grain reconfigurable architectures. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 474–479 Vol.1.

Belsa, A., Sarabia-Jacome, D., Palau, C. E., and Esteve, M. (2018). Flow-based programming interoperability solution for iot platform applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 304–309. IEEE.

Binkert, N., Sardashti, S., Sen, R., Sewell, K., et al. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7.

Blackstock, M. and Lea, R. (2014). Toward a distributed data flow platform for the web of things (distributed node-red). In *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39. ACM.

Bray, T. et al. (2014). The javascript object notation (json) data interchange format. .

Budiu, M. and Goldstein, S. C. (2002). Compiling application-specific hardware. In Glesner, M., Zipf, P., and Renovell, M., editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pages 853–863, Berlin, Heidelberg. Springer Berlin Heidelberg.

Burger, D. and Austin, T. M. (1997). The simplescalar tool set. *ACM SIGARCH Computer Architecture News*, 25(3):13–25.

C. Penha, J., B. Silva, L., M. Silva, J., Coelho, K. K., P. Baranda, H., M. Nacif, J. A., and S. Ferreira, R. (2019). Add: Accelerator design and deploy-a tool for fpga high-performance dataflow computing. *Concurrency and Computation: Practice and Experience*, 31(18):e5096.

Caldeira, P., Penha, J. C., Bragança, L., Ferreira, R., Nacif, J. A. M., Ferreira, R., and Pereira, F. M. (2018). From java to fpga: An experience with the intel harp system. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 17–24. IEEE.

Canesche, M., Vasconcelos, V., Passe, F., Penha, J., and Ferreira, R. (2017). Exemplos e aplicações utilizando a ferramenta add. *WIC-WSCAD*.

Chaczko, Z. and Braun, R. (2017). Learning data engineering: Creating iot apps using the node-red and the rpi technologies. In *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8. IEEE.

Chanthakit, S. and Rattanapoka, C. (2018). Mqtt based air quality monitoring system using node mcu and node-red. In *2018 Seventh ICT International Student Project Conference (ICT-ISPC)*, pages 1–5. IEEE.

Chin, S. A., Niu, K. P., Walker, M., Yin, S., Mertens, A., Lee, J., and Anderson, J. H. (2018). Architecture exploration of standard-cell and fpga-overlay cgras using the open-source cgra-me framework. In *Proceedings of the 2018 International Symposium on Physical Design*, pages 48–55.

Client, I. (2020). Jointjs/rappid-html 5 diagramming toolkit. https://www.jointjs.com/.

Consortium, O. et al. (2013). Openspl: Revealing the power of spatial computing. Technical report, Technical report, Dec.

Cross-Platform, F. and Developers, A. (2013). Simplify software integration for fpga accelerators with opae. In .

cs.sonoma.edu (2020). Online - MIPS Assembly Simulator. https://rivoire.cs.sonoma.edu/cs351/wemips/.

Culler, D. E. (1986). Dataflow architectures. *Annual review of computer science*, 1(1):225–253.

da Silva, L. B., Almeida, D., Nacif, J. A. M., Sánchez-Osorio, I., Hernández-Martínez, C. A., and Ferreira, R. (2017). Exploring the dynamics of large-scale gene regulatory networks using hardware acceleration on a heterogeneous cpu-fpga platform. In *IEEE Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*.

Dave, S. and Shrivastava, A. (2017). Ccf: A cgra compilation framework. .

Dennis, D. K., Priyam, A., Virk, S. S., Agrawal, S., Sharma, T., Mondal, A., and Ray, K. C. (2017). Single cycle risc-v micro architecture processor and its fpga prototype. In *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, pages 1–5. IEEE.

digitaljs.tilk.eu (2020). Open-source DigitalJS: digital logic simulator, by Marek Materzok, University of Wrocław. http://digitaljs.tilk.eu/.

dii.unisi.it (2020). Web-risc-v. http://x.dii.unisi.it:8098/~giorgi/WebRISC-V/index.php.

dpi.ufv.br (2019). Coleção de links para o ensino e uso de Node-Red. http://iot.dpi.ufv.br/links_artigo.html.

eng.cam.ac.uk (2020). Offline - MIPS-Datapath - Graphical MIPS CPU Simulator. http://mi.eng.cam.ac.uk/~ahg/MIPS-Datapath/.

Etiemble, D. (2018). 45-year cpu evolution: one law and two equations. *arXiv preprint arXiv:1803.00254*.

Ferreira, R., Cardoso, J., and Neto, H. C. (2004). An environment for exploring data-driven architectures. In *Field-Programmable Logic and Applications*.

Ferreira, R., Cardoso, J. M., Toledo, A., and Neto, H. C. (2005). Data-driven regular reconfigurable arrays: design space exploration and mapping. In *International Workshop on Embedded Computer Systems*, pages 41–50. Springer.

Ferreira, R., Denver, W., Pereira, M., Quadros, J., Carro, L., and Wong, S. (2014). A run-time modulo scheduling by using a binary translation mechanism. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 75–82. IEEE.

Ferreira, R., Nacif, J., Magalhaes, S., de Almeida, T., and Pacifico, R. (2015). Be a simulator developer and go beyond in computing engineering. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE.

Ferreira, R., Trullemans, A.-M., Costa, J., and Monteiro, J. (2000). Probabilistic bottom-up rtl power estimation. In *Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525)*, pages 439–446. IEEE.

Ferreira, R., Vendramini, J., and Nacif, M. (2011). Dynamic reconfigurable multicast interconnections by using radix-4 multistage networks in fpga. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 810–815. IEEE.

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., and Bader, G. D. (2016). Cytoscape. js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311.

Giang, N. K., Blackstock, M., Lea, R., and Leung, V. C. (2015). Developing iot applications in the fog: A distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 155–162. IEEE.

Giorgi, R. and Mariotti, G. (2019). Webrisc-v: A web-based education-oriented risc-v pipeline simulation environment. In *Proceedings of the Workshop on Computer Architecture Education*, WCAE'19, pages 3:1–3:6, New York, NY, USA. ACM.

godbolt.org (2020). Compiler Explorer: an interactive online compiler which shows the assembly output of compiled. https://godbolt.org/.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

Guinard, D. and Trifa, V. (2016). *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co.

Gür, E., Sataner, Z. E., Durkaya, Y. H., and Bayar, S. (2018). Fpga implementation of 32-bit risc-v processor with web-based assembler-disassembler. In *2018 International Symposium on Fundamentals of Electrical Engineering (ISFEE)*, pages 1–4. IEEE.

Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE.

IEEE (1988). Ieee standard vhdl language reference manual. *IEEE Std 1076-1987*, pages 1–218.

IEEE (1996). Ieee standard hardware description language based on the verilog(r) hardware description language. *IEEE Std 1364-1995*, pages 1–688.

Intel (2020). Intel xeon with integrated fpga systems at pc².

jamesgart.com (2020). Offline - a Visual MIPS R2000 Processor Simulator - freeware. http://jamesgart.com/procsim/.

jdoodle.com (2020). Jdoodle- Online Verilog compiler/editor/runner. https://www.jdoodle.com/execute-verilog-online/.

Johnston, W. M., Hanna, J. P., and Millar, R. J. (2004). Advances in dataflow programming languages. *ACM computing surveys (CSUR)*, 36(1):1–34.

Kabir, M., Bari, M., and Haque, A. (2011). Visimips: Visual simulator of MIPS32 pipelined processor. In *Computer Science & Education (ICCSE)*. IEEE.

Kho, N. M. D. and Uy, R. L. (2017). Mipsers: Mips extension release 6 simulator. In *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pages 1–6. IEEE.

Kleinfeld, R., Steglich, S., Radziwonowicz, L., and Doukas, C. (2014). glue. things: a mashup platform for wiring the internet of things with the internet of services. In *Proceedings of the 5th International Workshop on Web of Things*, pages 16–21. ACM.

Kobylinski, K., Bennett, J., Seto, N., Lo, G., and Tucci, F. (2014). Enterprise application development in the cloud with ibm bluemix. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, pages 276–279. IBM Corp.

Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*.

Krommydas, K., Sasanka, R., and Feng, W.-c. (2016). Bridging the fpga programmability-portability gap via automatic opencl code generation and tuning. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 213–218. IEEE.

Larus, J. R. (1990). *Spim s20: A mips r2000 simulator*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin.

Lekić, M. and Gardašević, G. (2018). Iot sensor integration to node-red platform. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5. IEEE.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. (2018). Meltdown. *arXiv preprint arXiv:1801.01207*.

Marwedel, P., Cong, K., and Schwenk, S. (2002). Ravi: Interactive visualization of information system dynamics using a java-based schematic editor. .

Materzok, M. (2019). Digitaljs: a visual verilog simulator for teaching. In *Proceedings of the 8th Computer Science Education Research Conference*, pages 110–115.

Mineraud, J., Mazhelis, O., Su, X., and Tarkoma, S. (2016). A gap analysis of internet-of-things platforms. *Computer Communications*, 89:5–16.

Mortbopet (2020). A graphical 5-stage risc-v pipeline simulator & assembly editor. https://github.com/mortbopet/Ripes.

Mutigwe, C., Kinyua, J., and Aghdasi, F. (2013). Instruction set usage analysis for application-specific systems design. *Int'l Journal of Information Technology and Computer Science*, 7(2).

Mutlu, O., Belgard, R., Gross, T. R., Hennessy, J. L., Patt, Y. N., et al. (2016). Common bonds: Mips, hps, two-level branch prediction, and compressed code risc processor. *IEEE Micro*, 36(4):70–85.

Nane, R., Sima, V.-M., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y. T., Hsiao, H., Brown, S., Ferrandi, F., et al. (2015). A survey and evaluation of fpga high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604.

Nestor, J. A. (2005). Teaching computer organization with hdls: An incremental approach. In *2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*, pages 77–78. IEEE.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6(2):40–53.

Nova, B., Ferreira, J. C., and Araújo, A. (2013). Tool to support computer architecture teaching and learning. In *2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)*, pages 1–8. IEEE.

Nowatzki, T., Gangadhar, V., Ardalani, N., and Sankaralingam, K. (2017). Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 416–429. IEEE.

ntu.edu.sg (2020). MIPS 101 - Online 32-bit MIPS Instruction Set Architecture. https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/index.html.

Passe, F., Bragança, L., Canesche, M., Cathoud, F., Nacif, J. A., and Ferreira, R. (2020). Plain: Ferramenta para desenvolvimento de aceleradores para overlays em fpga na nuvem em tempo de execuç ao. *WSCAD*.

Passe, F., Canesche, M., Neto, O. P. V., Nacif, J. A., and Ferreira, R. (2020). Mind the gap: Bridging verilog and computer architecture. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Passe, F. F., Vasconcelos, V. C. R., Canesche, M., and Ferreira, R. (2017). Perspectivas para o uso do node-red no ensino de iot. In *International Journal of Computer Architecture Education*, volume 6, pages 46–51.

Patterson, D. (2017). Reduced instruction set computers then and now. *Computer*, 50(12):10–12.

Patterson, D. A. and Hennessy, J. L. (2013). *Computer organization and design MIPS edition: the hardware/software interface*. Newnes.

Patterson, D. A. and Hennessy, J. L. (2017). *Computer Organization and Design RISC-V 13 Edition: The Hardware Software Interface*. The Morgan Kaufmann Series.

Patti, D., Spadaccini, A., Palesi, M., Fazzino, F., and Catania, V. (2012). Supporting undergraduate computer architecture students using a visual mips64 cpu simulator. *IEEE Transactions on Education*, 55(3):406–411.

Penha, J. C., Fontes, G., and Ferreira, R. (2016). Mipsfpga - um simulador mips incremental com validação em fpga. *International Journal of Computer Architecture Education*, 5:19–25.

playground, E. (2020). Edit, save, simulate, synthesize systemverilog, verilog, vhdl and other hdls from your web browser. https://www.edaplayground.com/.

Red, N. (2017). A dashboard ui for node-red. https://github.com/node-red/node-red-dashboard.

Sharma, V., Rai, S., and Dev, A. (2012). A comprehensive study of artificial neural networks. *International Journal of Advanced research in computer science and software engineering*, 2(10).

Silva, L. B. D., Ferreira, R., Canesche, M., Menezes, M. M., Vieira, M. D., Penha, J., Jamieson, P., and Nacif, J. A. M. (2019). Ready: A fine-grained multithreading overlay framework for modern cpu-fpga dataflow applications. *ACM Trans. Embed. Comput. Syst.*, 18(5s).

Smith, J. E. (1981). A study of branch prediction strategies. In *Proceedings of the IEEE Annual Symposium on Computer Architecture*.

Stanojević, I., Kovačević, M., and Šenk, V. (2019). Application of maxeler dataflow supercomputing to spherical code design. In *Exploring the DataFlow Supercomputing Paradigm*, pages 133–168. Springer.

Stupefied (2020). A java swing based visual simulator of educational mips 32 pipeline. https://github.com/stupefied/visimips.

Tabaa, M., Chouri, B., Saadaoui, S., and Alami, K. (2018). Industrial communication based on modbus and node-red. *Procedia computer science*, 130:583–588.

techep.csi.cuny.edu (2020). Verilog Online Editor, Testbench, and Waveforms. http://www.techep.csi.cuny.edu/~zhangs/v.html.

tutorialspoint.com (2020). Compile and execute verilog online (icarus v10.0). https://www.tutorialspoint.com/compile_verilog_online.php.

UFV, D. (2020). RiscVerilog - Framework to teach MIPS/RiscV Verilog. http://www.ufv.br/riscVerilog.

umass.edu (2020). Online - Pipeline floatpoint variable latency MIPS. http://www.ecs.umass.edu/ece/koren/architecture/windlx/main.html.

Veen, A. H. (1986). Dataflow machine architecture. *ACM Computing Surveys (CSUR)*, 18(4):365–396.

Veneri, G. and Capasso, A. (2018). *Hands-on Industrial Internet of Things: Create a Powerful Industrial IoT Infrastructure Using Industry 4.0*. Packt Publishing Ltd.

Vollmar, K. and Sanderson, P. (2006a). Mars. *Proc. of SIGCSE technical symposium on Computer science education*.

Vollmar, K. and Sanderson, P. (2006b). Mars: an education-oriented MIPS assembly language simulator. In *ACM SIGCSE Bulletin*, pages 239–243. ACM.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11.

Wolf, C., Glaser, J., and Kepler, J. (2013). Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*.

Wongsuphasawat, K., Smilkov, D., Wexler, J., Wilson, J., Mane, D., Fritz, D., Krishnan, D., Viégas, F. B., and Wattenberg, M. (2017). Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12.

Yeh, T. and Patt, Y. N. (1992). Alternative implementations of two-level adaptive branch prediction. In *ACM SIGARCH Computer Architecture News*, pages 124–134. ACM.

Yourdon, E., EDWARD, Y., Constantine, L., and Press, Y. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press computing series. Prentice Hall.

Zang, Z., Liu, Y., and Cheung, R. C. C. (2019). Reconfigurable risc-v secure processor and soc integration. In *2019 IEEE International Conference on Industrial Technology (ICIT)*, pages 827–832.

Zhang, D., Zhao, R. C., Han, L., Liang, W. F., Qu, J., and Liu, X. N. (2011). A fast design space exploration method for reconfigurable architecture based on loop optimization. In *Materials, Mechatronics and Automation*, volume 467 of *Key Engineering Materials*, pages 812–817. Trans Tech Publications Ltd.