Escalonador em Hardware com Interface Avalon

Luiz Henrique de Lorenzi Cancellier, Vinicius Marino Calvo Torres de Freitas

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

luizhenriquecancellier@gmail.com, vinicius.mctf@grad.ufsc.br

Resumo. *Sistemas orientados a aplicação permitem a implementação de componentes híbridos de hardware e software. Neste trabalho foi feita a implementação do escalonador em hardware para EPOS, sendo agora possívei usá-la no barramento Avalon da FPGA's DE2 da Altera. O hardware desenvolvido foi submetido a 26 testes de unidade para cada uma de suas funcionalidades e uma classe de referência foi desenvolvida para testes de sistema, garantindo que a adaptação foi bem sucedida.*

1. Introdução

O desenvolvimento de sistemas embarcados tem se tornado cada vez mais complexo. Sistemas orientados a aplicação podem utilizar componentes híbridos de *hardware* e *software* [Marcondes and Fröhlich 2008], uma vez que diversos componentes desse tipo de sistema podem ser implementados das duas formas.

No trabalho feito por Marcondes alguns componentes do Sistema Operacional (SO) foram desenvolvidos para operarem de forma híbrida. O escalonamento de processos e semáforos foram implementados em *hardware* e o EPOS (*Embedded Parallel Operating System*) foi adaptado para lidar com esses componentes [Marcondes 2009]. Na implementação feita, cabia ao projetista configurar o EPOS para compilar os componentes funcionando em *software* ou em *hardware*, de forma completamente transparente para o restante do sistema. No trabalho de Marcondes, os componentes foram feitos para funcionar apenas sobre a *Field Programable Gate Arrays* (FPGA) da Xilinx.

Com o objetivo de permitir uma melhor visibilidade do projeto e facilidade de reprodução dos experimentos na Universidade Federal de Santa Catarina (UFSC), este trabalho realiza uma adaptação do escalonador para a FPGA DE2 da Altera. A principal motivação para realizar essa adaptação é que as placas da Altera estãos disponíveis nos laboratórios de ensino da UFSC, usados em disciplinas do curso de graduação em Ciências da Computação.

O restante deste trabalho está organizado da seguinte forma: na Seção 2 será detalhado o funcionamento do componente de *hardware* e na Seção 3 será explicado como o mesmo foi conectado ao barramento da FPGA da Altera. Na Seção 4 será apresentada a forma como a funcionalidade do projeto foi testada. Por fim, as conclusões finais serão apresentadas na Seção 5.

2. Escalonador em Hardware

O funcionamento do escalonador baseia-se em guardar um endereço de memória e a respectiva prioridade de cada tarefa. A prioridade é definida pelo *software* de acordo com

uma política de escalonamento. Como a política e o funcionamento do escalonador são conceitos isolados no EPOS, o componente pode funcionar com a prioridade sendo passada por parâmetro, tornando o sistemas flexível para uma implementação híbrida.

Para permitir que ele tenha todas as funcionalidades de um escalonador em *software*, os principais comandos executados pelo escalonador são:

- *Create*: cria uma nova thread no escalonador
- *Delete*: deleta uma thread do escalonador
- *Insert*: adiciona uma thread já criada a fila do escalonador
- *Remove*: remove uma thread da fila do escalonador
- *Update Running*: atualiza por prioridade qual é a thread em execução
- *Get ID*: retorna o endereço interno de uma thread no escalonador
- *Chosen*: retorna o endereço interno da thread executando
- *Size*: retorna o número de threads na fila de pronto

Há também comandos que permitem o controle de tempo da execução, que são:

- *Set quantum*: atualiza o valor do quantum
- *Reset ticks*: reinicia o contador de tempo restante
- *Interrupt acknowledge*: sinaliza que a interrupção foi reconhecida
- *Enable*: habilita que interrupções ocorram no fim do quantum
- *Disable*: desabilita as interrupções

Existem ainda outros comandos, mas são apenas especializações dos que já foram listados. Quase todos os comandos executam em apenas um ciclo de relógio, com exceção daqueles que fazem buscas em tabelas internas ou precisam de alguma configuração antes da execução. Os comandos *insert*, *remove*, *destroy* e *Get ID* são exemplos que levam mais de um ciclo.

Foi adicionado ainda um comando para resetar a arquitetura de forma síncrona. Essa funcionalidade garante que o bloco possa ser resetado individualmente via *software*, não só na forma global do sistema. Tal adição foi feita apenas para simplificar os testes de unidade, que assumem que o *hardware* está entrando em operação no começo do teste. Esse comando pode ser removido para a versão final do componente sem afetar seu funcionamento.

3. Adaptador

Como o compoente não foi desenvolvido originalmente para o barramento da *Avalon* [Altera 2015], foi necessário desenvolver um adaptador que faz o intermédio entre esses dois elementos. Nesta seção serão apresentados inicialmente os sinais da interface *Memory Mapped Slave* para se comunicar com o barramento da Avalon. Também será apresentado em detalhes o controle que o adaptador faz para traduzir os sinais entre componente e barramento.

3.1. Barramento Avalon

O barramento fornece como entrada 6 sinais para o bloco. Os sinais de *Clock* e *Reset* na rede de relógio e ao sinal de *reset*, respectivamente. O sinal *write* indica que algum dado está sendo escrito no barramento e o análogo ocorre para as leituras, com o sinal *read*.

Escritas são feitas através do sinal *writedata* de n bits (parametrizado). Quando um dado é lido ou escrito, também é identificado o endereço em seu espaço de memória, com o sinal *address* (de três bits). Há ainda um sinal chamado de *chipselect* que fica ativo quando o Escalonador é o bloco que está recebendo ou enviando informações pelo barramento.

Além dos sinais de entrada supracitados, existem ainda três sinais de saída. O primeiro deles é uma conexão de dados com o barramento, por onde passarão as informações do Escalonador em *hardware* para o *software*. O segundo sinal é um sinal de interrupções, que é conectado diretamente na CPU. O terceiro sinal é o de *readdata*, de n bits, que é por onde saem os dados lidos.

A figura 1 ilustra os sinais de entrada e saída do barramento. Os sinais *write*, *read*, *writedata*, *readdata*, *address* e *chipselect* se comunicam com o *Avalon Memory-Mapped Slave*, enquanto *clock* e *reset* se comunicam com *clock_sink* e *reset_sink*, respectivamente. O sinal de interrupções se comunica com o *interrupt handler*.

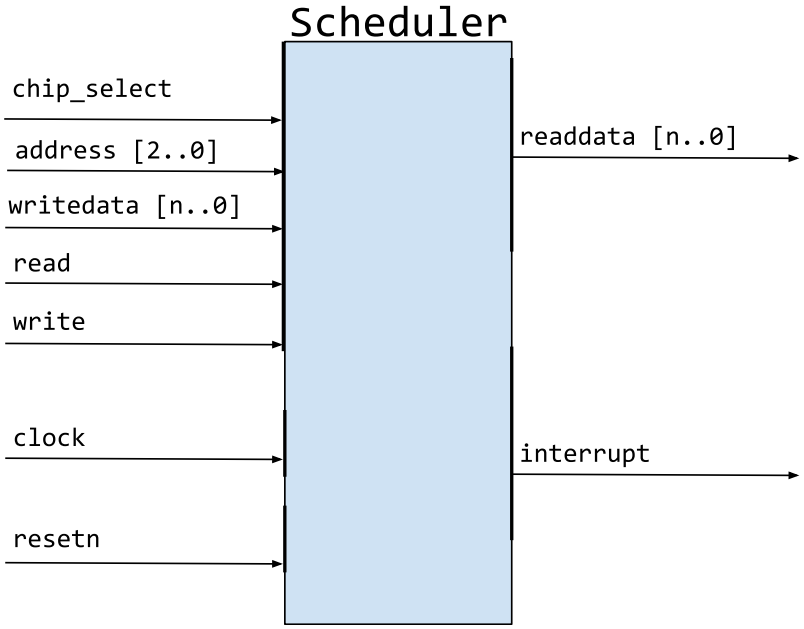

Figura 1. Bloco do escalonador gerado pelo Qsys

3.2. Controle dos Sinais

A Figura 2 apresenta a organização do *hardware* do ponto de vista do Escalonador. Para fins de clareza e organização do código, há uma interface responsável por se conectar ao barramento externo e fazer as ligações corretas entre os blocos Adaptador e Escalonador. O Adaptador recebe os sinais do barramento e realiza o tratamento necessário para que o bloco Escalonador receba corretamente suas entradas. Após o Escalonador realizar o processamento, seus sinais de saída retornam ao Adaptador que os transfere corretamente para o barramento.

Como o barramento permite transferir apenas um dado, o bloco Adaptador precisa fazer com que essa limitação seja transparente para o Escalonador. Para isso, os sinais escritos nos primeiros endereços são temporariamente armazenados em registradores. Quando o comando é recebido, todos os dados são transferidos para o Escalonador e

ficam estáveis por apenas um ciclo de relógio, sendo zerados na sequência. Após realizar todo o processamento, o Escalonador retorna um status e um dado. Cada uma dessas informações passam pelo barramento quando há uma leitura no seu respectivo endereço.

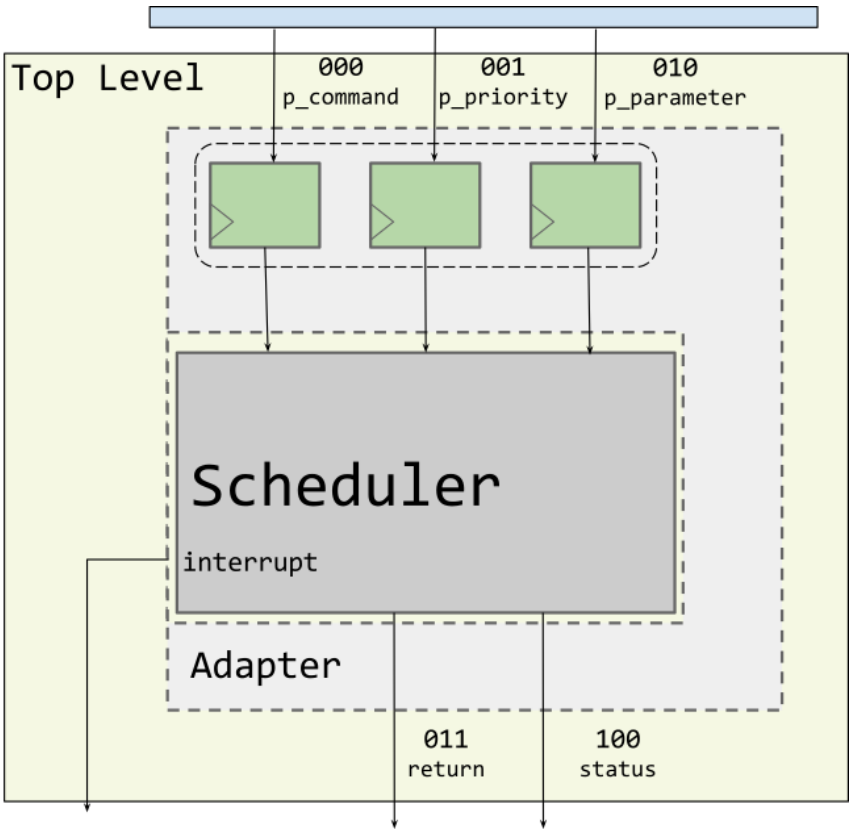

Figura 2. Organização do componente. O conteúdo do bloco *Adapter* e a comunicação com o barramento foram abstraídas, apresentando apenas a visão que o bloco *Scheduler* tem do sistema.

4. Validação em Software

Nesta seção serão apresentados os trabalhos desenvolvidos em *software* para controlar o Escalonador. Foram desenvolvidos testes de unidade para comprovar o funcionamento de cada comando e, uma vez validado o projeto, foi implementado um escalonador em C++ seguindo o modelo dessa mesma classe no EPOS.

4.1. Testes de Unidade

Foi elaborado um conjunto com um total de 26 testes de unidade. As funções mais básicas foram verificadas primeiro, para posteriormente serem usadas em testes envolvendo operações mais complexas. A Figura 4.1 ilustra alguns fluxos de testes executados para o comando *remove*. As operações envolvendo controle de tempo e interrupções foram implementadas, mas não há nenhum mecanismo formal para verificação delas. O principal motivo para isso é que a arquitetura não foi projetada viabilizando testes de caixa cinza e, portanto, não permite a verificação parcial de sinais internos como contadores do *timer*.

A classe *scheduler* foi desenvolvida após a compravação do funcionamento de todas as operações que o escalonador executa. Uma vez que cada comando foi verificado com testes de unidade e corrigido, toda a implementação da classe foi feita sem que qualquer erro de funcionalidade se manifestasse.

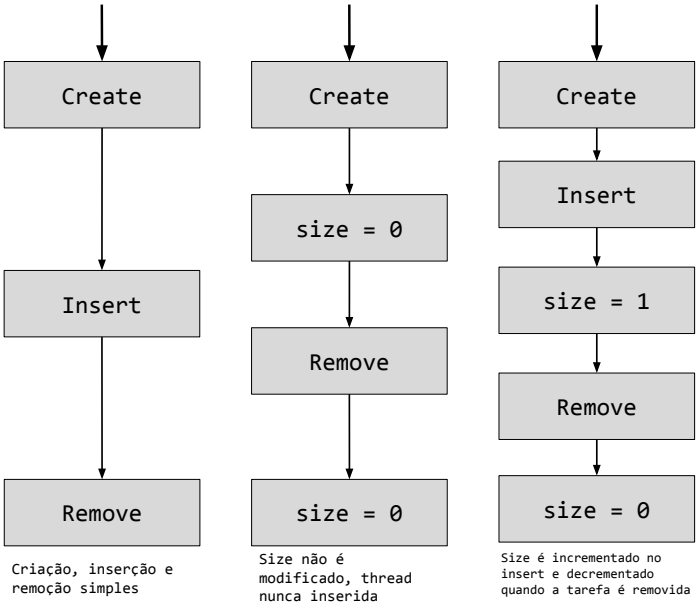

Figura 3. Exemplo de fluxo dos testes de *Remove*

4.2. Escalonador do EPOS

Os métodos dessa classe e suas respectivas assinaturas seguem o mesmo padrão do escalonador do EPOS. Apesar das modificações necessárias para poder lidar com as características do *hardware*, toda a comunicação feita diretamente com o componente é isolada em um único método chamado de "*execute_cmd*". Tal implementação permite que, mesmo com outra interface de comunicação, seja simples desenvolver novo código para outras plataformas. A implementação foi feita com base naquela apresentada por Marcondes para o *scheduler*, mas a mesma organização não foi observada em outros códigos disponíveis no mesmo trabalho.

O Escalonador apresenta os seguintes métodos:

- *Chosen*: retorna endereço da thread executando
- *Insert*: insere uma nova thread no escalonador
- *Remove*: remove thread do escalonador
- *Suspend*: coloca thread na fila de espera
- *Resume*: atualiza por prioridade qual é a thread em execução
- *Choose*: Coloca a thread de maior prioridade ou escolhida para executar
- *Choose Another*: força que outra thread substitua a que está executando

Quase todos os métodos apresentados não podem ser mapeados diretamente para um comando específico do bloco em *hardware*, sendo necessário usar uma sequência de vários deles para executar a operação necessária. Existe ainda um método que é assinalado para tratar as interrupções do bloco [Li and Edwards 2012]. No EPOS, o tratador chama

um método da classe Thread, mas como não há como realizar essa operação, adotou-se a implementação do método "*Choose Another*" para quando uma interrupção ocorre.

Durante a execução de um comando há a necessidade de fazer espera ocupada, esperando pelo status de pronto do bloco antes de efetivamente ler o resultado. Tal abordagem já era usada na implementação de Marcondes. No barramento Avalon existe o conceito de *delay* de leitura e escrita, que atrasa a operação do barramento em uma quantidade prédefinida de ciclos. Esse *delay* poderia resolver o problema de espera ocupada, mas não foi usado por dois motivos: Cada comando apresenta uma quantidade de ciclos variadas, sendo necessário limitar o sistema ao pior caso. Não foram encontradas aplicações com uso prático de tal recurso, sendo recomendado o padrão de um ciclo de *delay*.

5. Conclusão

Neste trabalho o escalonador do SO desenvolvido por Marcondes como um componente de *hardware* foi adaptado para a interface *Avalon* usando a interface *Memory-Mapped Slave*. Cada comando do escalonador foi testado individualmente, comprovando seu correto funcionamento com um conjunto de 26 testes de unidade. Além da adaptação feita em *hardware*, também foi desenvolvida uma classe em *software* baseada na implementação do escalonador do EPOS.

Não foi possível testar a implementação realizada em conjunto com o EPOS, uma vez que ele não foi portado para a arquitetura sintetizada na FPGA da Altera. Apesar de tal limitação, todo o material produzido neste trabalho foi elaborado visando uma futura integração com o escalonador do EPOS, onde deverá ser possível realizar a resolução em tempo de compilação de qual componente será usado, *software* ou *hardware*, e a reprodução dos experimentos de Marcondes para a FPGA da Altera.

Todo o código desenvolvido, incluindo documentação detalhada sobre cada comando da arquitetura, está disponível em `https://github.com/viniciusmctf/hybridscheduler/`.

Referências

Altera (2015). Avalon interface specifications. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf`. Acesso em 06 de Dezembro de 2016.

Li, S. and Edwards, S. A. (2012). Creating peripherals with interrupts in altera's sopc builder. Tutorial.

Marcondes, H. (2009). Uma Arquitetura de Componentes Híbridos de Hardware e Software para Sistemas Embarcados. Master's thesis, Federal University of Santa Catarina, Florianópolis. M.Sc. Thesis.

Marcondes, H. and Fröhlich, A. A. (2008). On Hybrid Hw/Sw Components for Embedded System Design. In *17th IFAC World Congress*, pages 9290–9295, Seoul, Korea.