

MC851 - Projeto em Computação I

arRISCado - Entrega 2

Ângelo Renato Pazin Malaguti - 165429

Claudio dos Santos Júnior - 195727

Elton Cardoso do Nascimento - 233840

Gabriel Costa Kinder - 234720

Iago Caran Aquino - 198921

João Pedro de Moraes Bonucci - 218733

1. Introdução

Neste relatório, continuamos com o trabalho iniciado no documento anterior, concentrando nossos esforços no desenvolvimento das entregas pendentes. Isso inclui a criação de um ambiente de execução funcional tanto para o processador quanto para o processador em FPGA. Além disso, nosso objetivo é desenvolver um cache L1, um periférico integrável com o processador em FPGA e expandir o conjunto de instruções de RV32I para RV32IMA, incorporando instruções de multiplicação e divisão inteira, bem como instruções atômicas.

2. Planejamento

Com o objetivo de reduzir o atraso em relação à entrega anterior, decidimos originalmente dividir a equipe em três frentes de trabalho, com duas delas focadas no desenvolvimento da segunda entrega, enquanto a terceira concentrou-se na finalização do código para a entrega anterior. As atividades realizadas serão listadas e detalhadas nas seções a seguir.

3. Desenvolvimento

a. Finalização do Pipeline

A divisão do desenvolvimento do pipeline em grupos separados levou a discrepâncias na nomenclatura de fios (wires) e registradores, o que causou dificuldades na integração dos componentes no pipeline. Um dos problemas encontrados foi a existência de sinais com o mesmo propósito, mas com nomes ligeiramente diferentes. Enquanto o Iago ficou encarregado da integração, Angelo revisou a nomenclatura da maioria dos sinais.

Após a conclusão dessa etapa, a dupla realizou a validação por simulação dos fluxos do pipeline e sua integração no módulo "CPU", obtendo resultados positivos.

b. Implementação do Pipeline na FPGA

O Iago iniciou os testes do pipeline na FPGA, observando que diferentes partes isoladas funcionam corretamente. Durante essa fase, foram realizadas algumas correções de comportamento, como a inversão dos botões e LEDs, além de ajustes na sensibilidade ao clock e correções nas saídas que afetavam o mesmo fio (wire) ou registrador.

Entretanto, ao integrar todos os estágios, diversos sinais saíram do controle. Ao conectar os LEDs a diferentes pontos do pipeline, foi observado que algumas saídas se comportam corretamente

nos estágios iniciais, mas apresentam problemas nos estágios posteriores, sugerindo possíveis inconsistências na estabilidade dos sinais. Devido ao grande número de problemas, decidimos envolver João para auxiliar nos testes com a FPGA.

c. Criação da UART

O Gabriel foi responsável por implementar uma interface UART com o objetivo de permitir a programação da placa sem a necessidade de refazer a síntese e o bitstream. Como resultado, o módulo UART passou a desempenhar duas funções: receber dados por meio da comunicação serial e armazenar/distribuir instruções do programa, substituindo o antigo módulo ROM. Um script em Python também foi desenvolvido para enviar corretamente os testes por meio da comunicação serial.

Após uma série de testes e iterações, alcançamos uma versão do módulo que consegue armazenar com sucesso todas as informações recebidas na memória da placa. Além disso, por meio de dois fios, um para a entrada do endereço e outro para a saída da instrução, o módulo é capaz de se comunicar com a CPU e transmitir as instruções a serem executadas.

d. Definição de testes

O Cláudio ficou responsável por completar os testes e gerar modelos executáveis para a FPGA. A primeira tarefa foi aprimorar os testes em assembly existentes, abrangendo instruções do tipo RV32M e RV32A, e convertê-los para formatos binário e hexadecimal. A dificuldade nessa etapa estava relacionada às informações nos binários gerados pelos métodos padrões, que precisaram ser depuradas. Além disso, foi necessário realizar uma conversão de little endian para big endian.

Subsequentemente, foram criados códigos em alto nível utilizando a linguagem C. Para esses códigos, um script foi desenvolvido para gerar o código em assembly e convertê-lo para os formatos binário e hexadecimal, utilizando o conjunto de instruções RV32IMA. A maior dificuldade nessa fase foi a eliminação das informações adicionais inseridas durante a compilação no código assembly.

e. Ampliação do conjunto de instruções

O Ângelo ficou encarregado de adicionar as instruções faltantes do conjunto RV32IMA nos módulos de decode, execute e na ALU. Do conjunto, as instruções do conjunto RV32A ainda não foram implementadas, devido à prioridade dada às instruções do conjunto RV32IM.

No módulo de decode, o opcode das instruções foi declarado como "localparam" no início do módulo, visando facilitar a compreensão e a implementação do código, tornando-o mais claro e legível. Nesta etapa, também foram instanciados os sinais de controle associados a cada instrução.

No módulo execute, os sinais de controle AluOP e AluSrc são utilizados para determinar os operandos a serem enviados para a ALU, enquanto na ALU, o sinal AluControl é empregado para determinar a operação aritmética ou lógica a ser executada entre os dois operandos. Os possíveis valores para o AluControl também foram definidos como "localparam" no início do módulo da ALU.

Até o momento, as instruções adicionadas e a utilização de "localparam" estão na Branch do Pull Request #41.

f. Implementação do Periférico

O Elton e o João ficaram responsáveis pela implementação de periféricos. Optamos por empregar periféricos cujos endereços de memória começam com prefixos distintos de "000", indicando, dessa forma, que se trata de endereços associados à memória principal. Endereços que se iniciam com valores diferentes (001, 010, ...) são atribuídos a diferentes periféricos, permitindo que o processador acomode até 7 dispositivos externos. Embora essa abordagem possa resultar em uma considerável redução na quantidade de endereços de memória disponíveis, foi escolhida devido à sua simplicidade de implementação e à improbabilidade de requerer uma grande quantidade de memória durante este projeto. A implementação do mapeamento foi concluída, porém ainda não foi submetida a validação.

O periférico inicialmente selecionado para implementação consiste em uma porta PWM (Pulse Width Modulation), que serve como uma interface de saída para o processador, permitindo simular a variação de potência entre desativado (0 V) e a ativado (tensão máxima) de uma porta da FPGA. Esse método é baseado na modulação da duração em que a porta mantém seu valor ativo (tensão máxima) ou desativado dentro de um ciclo, conhecido como *duty cycle* (ciclo de trabalho).

A porta é mapeada para os endereços iniciados em 001. Nossa implementação utiliza dois endereços de memória para a porta: o endereço 001xx...xx0 (PWM_1) indica a quantidade de ciclos que a porta deve permanecer ativa, enquanto que o endereço 001xx...xx1 (PWM_2) indica a quantidade de ciclos da FPGA dentro de um *duty cycle*. A porta fica então o máximo entre a PWM_1 e PWM_2 ativo a cada ciclo. Essa implementação foi realizada pensando em uma flexibilidade ajustável entre a resolução do sinal e a ilusão de um sinal analógico, pois quanto maior a resolução desejada para a tensão, mais ciclos serão necessários em um *duty cycle*, e menor será a percepção da transferência de potência de um sinal analógico.

A implementação da porta em si foi validada na FPGA utilizando um LED e verificando a diferença em seu brilho. Até o momento, as alterações estão na Branch do Pull Request #25.

g. Implementação da cache L1

O Iago ficou responsável pelo desenvolvimento da cache L1, tendo alguns avanços no entendimento e análise das dificuldades de sua implementação. No entanto, devido a alta prioridade das mudanças necessárias no pipeline esse desenvolvimento teve que ser postergado, não tendo avanços significativos na implementação dessa entrega.

4. Pendências

Abaixo segue uma lista de pendências para a próxima entrega:

- Corrigir problemas da pipeline na FPGA
- Corrigir bugs da integração do periférico
- Criação de um script para execução do periférico
- Adicionar as instruções do conjunto RV32A
- Implementação da Cache L1

5. Aprendizados

Abaixo segue uma lista de aprendizados que tiramos desse mês de trabalho:

- A ferramenta de síntese e a ferramenta de simulação têm suporte diferente a recursos da linguagem verilog. Por vezes, o código funciona perfeitamente em simulação, porém não se comporta da mesma maneira na FPGA.
- A ferramenta de síntese tenta sempre otimizar o circuito gerado, assim acompanhar o número de elementos utilizados é uma maneira de entender se ele está ou não gerando o que queremos.
- As ferramentas de compilação geram o código em little endian, é preciso ficar atento a isso e converter para big endian.
- É possível realizar o uso das ferramentas de simulação para automatização de testes.
- Uso da simulação para visualização da onda do sinal gerado pela FPGA, que foi importante para testes da porta PWM.