

UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
DEPARTAMENTO DE TECNOLOGIA
TEC498 PROJETO DE CIRCUITOS DIGITAIS
**PROJETO DA ARQUITETURA DE UM PROCESSADOR COM
PARALELISMO EM NÍVEL DE INSTRUÇÃO**

Cássio Silva de Sá Santos, Khaíck Oliveira Brito

Tutor: João Carlos Nunes Bittencourt

1 INTRODUÇÃO

Este trabalho tem como objetivo apresentar a síntese parcial, devido uso de somente algumas instruções determinadas a partir de programas previamente escolhidos, do projeto da arquitetura de um processador com paralelismo em nível de instrução, bem como trazer avaliações sobre a metodologia seguida, resultados junto a discussões sobre testes e conclusão.

Existem nesse documento explicações acerca da descrição do processador na linguagem de descrição de hardware *Verilog*, *test benches* e demais elementos utilizados para teste e validação do funcionamento do microprocessador, conceito de caminho crítico de um circuito, microarquitetura e características de um dispositivo FPGA.

Foi-se requerido para o desenvolvimento desse projeto conceitos básicos da arquitetura de processadores MIPS, de circuitos digitais e um contato mínimo com a ferramenta *Altera Quartus*, usada para simular o circuito do processador e coletar dados pertinentes acerca da produção. Também fora requisitado que o produto fosse capaz de operar a 100 MIPS (Microinstruções por segundo), possua uma memória compartilhada de 64KBytes e tenha como tecnologia alvo o dispositivo *FPGA Cyclone IV EP4CE30*.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 MICROARQUITETURA

Em acordo com (SORATO, 2008), uma microarquitetura pode ser tratada como a descrição de um circuito elétrico contido num computador, um processador e digital de sinais, uma *CPU*, unidade central de processamento, a partir dos quais é possível descrever ricamente o comportamento de um hardware.

Consoante (SORATO, 2008), uma microarquitetura assemelha-se e possui relações com a arquitetura do *instruction set* dos processadores, no entanto, existem pontos importantes que diferenciam tais conceitos, tais como o fato da microarquitetura ser responsável por descrever as partes de um processador e o modo de operação e interconexão, enquanto a arquitetura de instruções é atenta aos registradores presentes no processador, modos de endereçamento e outros

mais conceitos.

Tal conceito também pode ser descrito como a estrutura responsável por comandar e atentar, no baixo nível, detalhes que estão aquém do campo da programação, bem como consumo de energia de um sistema, complexidade da lógica, conectividade, facilidade para testar e outros pontos referentes a microarquitetura.

A implementação de uma microarquitetura resulta num circuito eletrônico, o qual pode conter elementos como gates, registradores, multiplexadores, *ALUs*, *LUTs*, Extensores, Contadores e outros elementos.

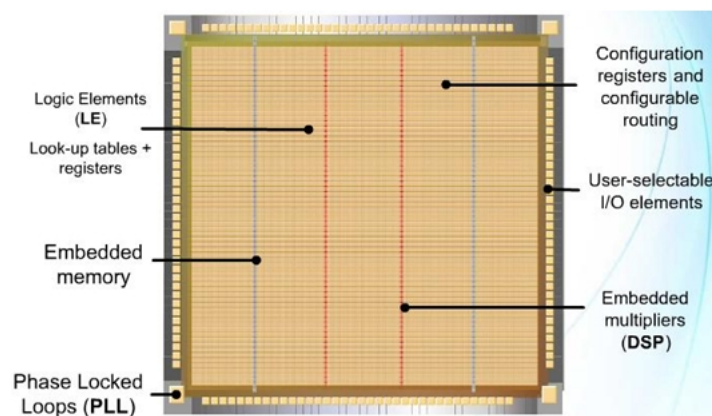
2.2 FPGA

2.2.1 Visão geral

Consoante (FONTANINI; OLIVEIRA, 2013) e (FUNDAMENTOS. . ., 2013), a tecnologia FPGA, que são arranjos de portas programáveis em campo ou, em inglês, *field programmable gate arrays*, foi desenvolvida em 1985 pelo engenheiro Ross Freeman. Esse dispositivo agrega os principais e melhores recursos dos circuitos integrados para aplicações específicas (*ASICs*, sigla em inglês) e sistemas baseados em processadores. Características marcantes desse ambiente programável são serem confiáveis, fornecerem velocidade temporizada por hardware e sua capacidade de reprogramação.

2.2.2 Arquitetura das FPGAs

Figura 1 – Elementos básicos da arquitetura de um FPGA



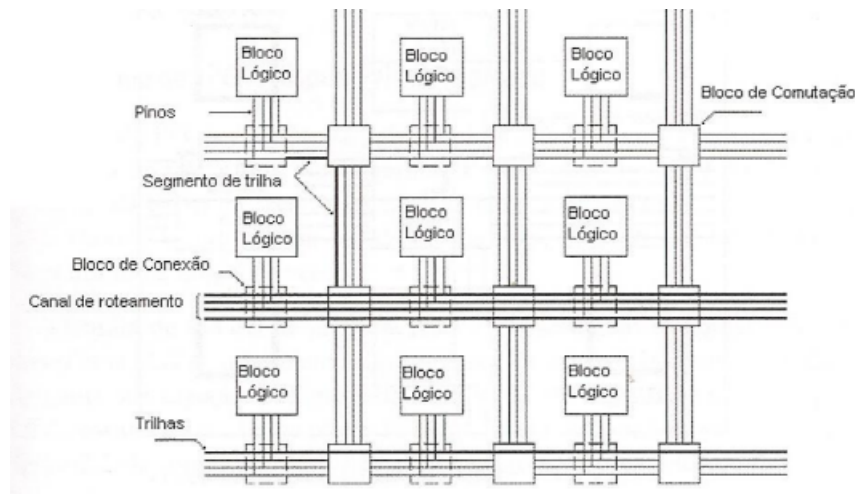
Fonte: (CARDOSO, 2011)

A arquitetura de um FPGA, a qual pode ser observada na Figura 1, é em sua grande maioria composta por Elementos Lógicos (*LE*, *Logic Elements*), aparelhados em malhas ao longo de todo o equipamento. Separam-se os elementos lógicos em dois tipos de unidades: *Look-Up Tables (LUT)*, que implementam funções lógicas mais comuns, como *OR* e *AND*, e registros que podem implementar lógica síncrona, como *flip-flops*. (CARDOSO, 2011)

As memórias embutidas (*Embedded Memory - EM*) podem ser dispostas em série ou paralelo visando obter-se memórias mais robustas. Outros recursos são os multiplicadores embutidos, os quais podem ser cascadeados de forma a melhorar o processamento digital de sinais (PDS), colaborando na implementação de funções avançadas.(CARDOSO, 2011)

Os FPGAs também possuem elementos de entrada e saída, identificados na Figura 1 por *User-selectable I/O Elements*, que podem ser aparelhados e utilizados a medida que o arquiteto necessite. Estes elementos podem ser posicionados e configurados para comunicar o FPGA com outros dispositivos externos presentes no circuito impresso. (CARDOSO, 2011) ()

Figura 2 – Elementos básicos da arquitetura de roteamento de um FPGA



Fonte: (FONTANINI; OLIVEIRA, 2013)

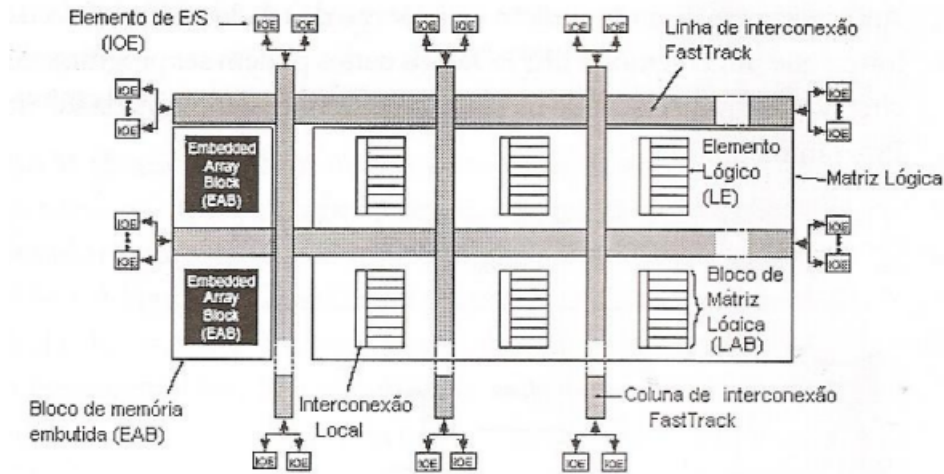
A arquitetura de roteamento de um FPGA, apresentada na Figura 2, pode ser explicada como a forma organizacional e posicional dos barramentos das chaves de comutação, a qual permite a interconexão entre as células lógicas. (FONTANINI; OLIVEIRA, 2013)

Segue breve explicação dada em (FONTANINI; OLIVEIRA, 2013) dos itens dispostos na Figura 2:

- Pinos: Entrada e Saída de blocos lógicos.
- Conexão: Ligação elétrica de um par de pinos
- Rede: Conjunto de pinos que estão conectados.
- Bloco de Comutação: Utilizado para conectar os segmentos da trilha.
- Segmentos da Trilha: Segmento não interrompido por chaves programáveis.
- Canal de Roteamento: Grupo de 2 ou mais 3 trilhas paralelas.
- Bloco de Conexão: Permite a conectividade das entradas e saídas de um bloco lógico com os segmentos de trilhas nos canais.

2.2.3 Dispositivo alvo

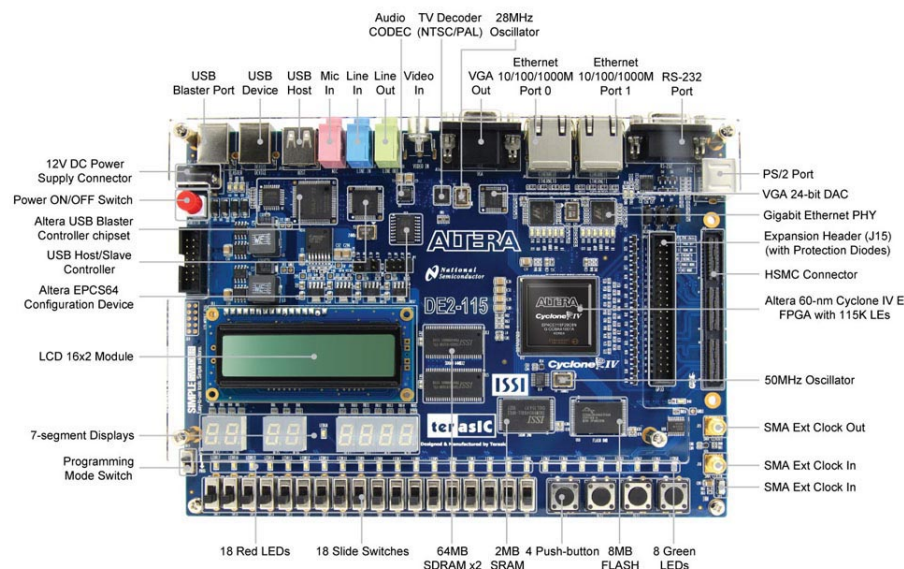
Figura 3 – Elementos básicos da arquitetura um FPGA Altera



Fonte: (FONTANINI; OLIVEIRA, 2013)

A família de FPGAs da Altera Corp. possui como característica uma hierarquia de três níveis muito similar à encontrada nos *CPLDs*, que são Dispositivos Lógicos Programáveis Complexos, e é baseada em tecnologia SRAM, também possuindo uma LUT de quatro entradas como seu elemento lógico básico LE. A arquitetura interna básica da família Altera, conforme a Figura 3, contém três tipos de células lógica: Elemento lógico(LE), bloco de matriz lógica (LAB) e bloco de memória embutido (EAB). (FONTANINI; OLIVEIRA, 2013)

Figura 4 – Elementos físicos da FPGA Cyclone IV



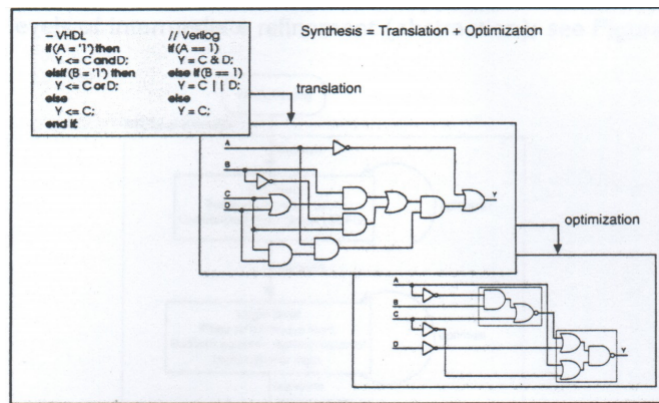
Fonte: <http://blog.terasic.com/>

O FPGA Cyclone IV EP4CE30, apresentado na Figura 4, possui 28.848 LEs, memória dedicada de 594 Kbits, 4 PLLs e 532 pinos de E/S. Os elementos lógicos presentes na Cyclone IV são divididos em dois modos de operação: O modo normal é aplicável para aplicações lógicas gerais e funções combinacionais. Nesse modo, 4 entradas de dados proveniente do LAB local interconnect são inseridas em um LUT de 4 entradas. O modo aritmético é ideal para implementação de somadores, contadores, simuladores e comparadores. Um elemento lógico no modo aritmético implementa um somador completo de 2 bits e um *carry chain* básico. (ALTERA CORPORATION, 2016)

2.3 PROCESSO DE SÍNTESE

2.3.1 Síntese lógica

Figura 5 – Representação de uma síntese de um circuito



Fonte: (VIEIRA, 2009)

A Figura 5 ilustra uma síntese lógica, a qual é um processo efetuado por ferramentas de síntese, como por exemplo o Quartus, pelo qual o comportamento de uma forma abstrata de circuito, geralmente encontrada em RTL, *register transfer level*, e escrita em linguagens de descrição de hardware, como verilog e vhdL, é gerada uma *netlist*, que é lista de componentes e suas interligações. Tem como objetivo maximizar a performance do circuito, reduzir custos energéticos como também minimizar a área ocupada.

Esse processo, também chamado de fluxo de síntese é dado através de algumas etapas, essas descritas por (VIEIRA, 2009):

- Preparação da descrição do circuito (HDL/VHDL)

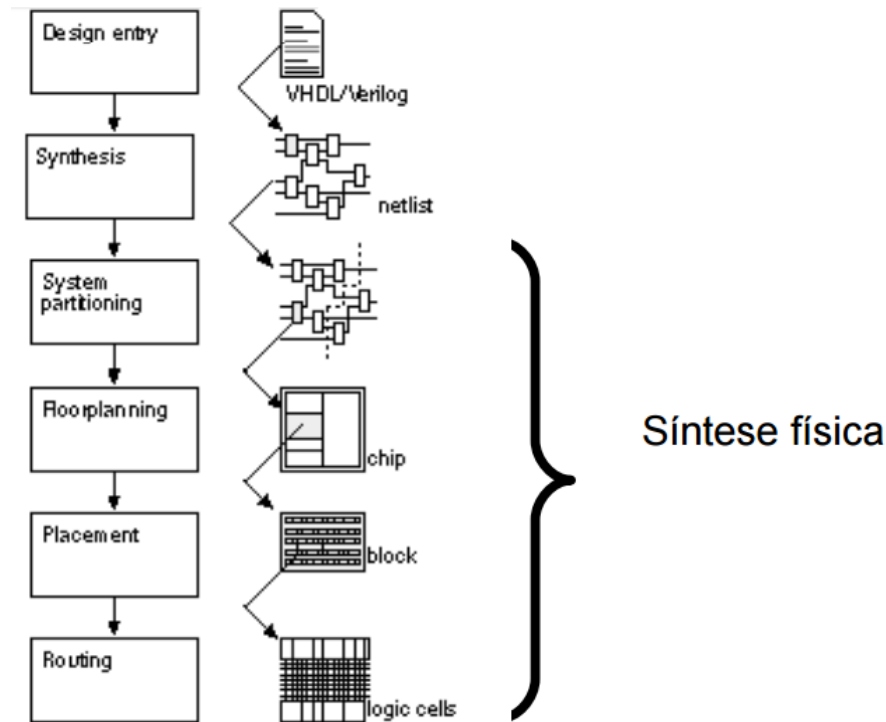
Síntese Tradução para componentes sintéticos (*DesignWare*) e para tecnologia genérica (*GTECH*).

- Otimização e mapeamento para biblioteca-alvo
- Síntese de infra-estrutura de teste (*scan*)
- Re-síntese após síntese física

2.3.2 Síntese Física

O processo de síntese física de um circuito é a capacidade de se criar um circuito físico a partir de um código RTL. Através desse processo é possível se verificar valores importantes para um circuito, como valores reais de tempo, que podem ser atrelados ao caminho crítico do sistema, que significa o processo mais longo e atrasado que um circuito possui, o qual determina o valor do pulso de relógio no sistema. (AGAH, 2011)

Figura 6 – Representação de uma síntese física de um circuito



Fonte: (VIEIRA, 2008)

Assim como a síntese lógica, também é dado através de algumas etapas ilustradas na Figura 6:

- Partição do sistema
- *Floorplanning* - Preparação de ambiente
- *Placement* - Formação de blocos
- *Routing* - Interligações dos blocos, montando células lógicas

3 DESCRIÇÃO DO DESENVOLVIMENTO

3.1 LEVANTAMENTO DE REQUISITOS

A partir da análise do problema apresentado, percebeu-se a indispensabilidade da aplicação do conceito de paralelismo no processador a ser desenvolvido em virtude da requisição de que 5 instruções fossem executadas simultaneamente a cada ciclo de clock, tirando o máximo proveito dos recursos de hardware. Assim, após uma busca por conceitos acerca da construção de processadores MIPS com paralelismo a nível de instrução, fora reconhecida a inevitabilidade de implementação dos registradores interestágios, responsáveis por todo o processo de funcionamento do pipeline junto a unidade de control do processador.

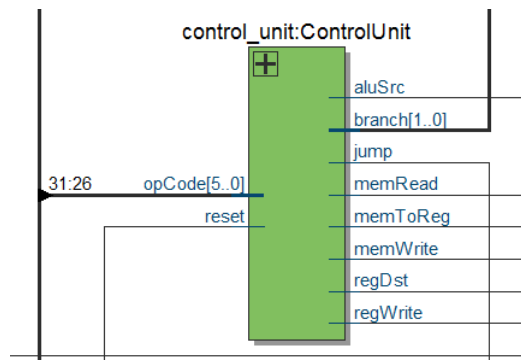
Levantou-se também a necessidade de desenvolver estruturas responsáveis para tratar conflitos que seriam provenientes das novas estruturas implementadas responsáveis por dar funcionalidade ao pipeline, seriam estas as unidades de controle de conflito *hazard unit* e *forwarding unit*.

Por conta do tamanho de 8 bits de palavra na memória no padrão MIPS, o qual é seguido pelo processador aqui explanado, e das palavras trabalhadas fora da memória serem de 32 *bits*, avaliou-se a viabilidade da multiplicação do *clock* do processador por 4 no bloco da memória para assim ser possível efetuar as ações de modo síncrono ao sistema.

3.2 PRINCIPAIS BLOCOS DO *DATAPATH*

3.2.1 Unidade de controle

Figura 7 – Bloco da unidade de controle do circuito



Fonte: Proprio autor

A unidade de controle, a qual é apresentada na Figura 7, presente no circuito é responsável por controlar as operações do processador. Ela coordena como os estágios de *Pipeline*, a memória de dados, a unidade lógica aritmetica e outras unidades irão corresponder mediante as instruções de um programa. A Tabela 1 apresenta os sinais de controle existentes na unidade de controle essenciais para o funcionamento do sistema.

Tabela 1 – Sinais da unidade de controle

INPUTS	BITS	SIGNIFICADOS
Opcode	5	Opcode da instrução a ser processada
Reset	1	Sinal para resetar a unidade de controle
OUTPUTS		
branch	01	A instrução é do tipo BEQ
	10	A instrução é do tipo BNE
	00	A instrução não é uma instrução de desvio
Jump	0	Não corresponde a uma instrução de jump
	1	Corresponde a uma instrução de jump
regDst	0	Registrador de destino é o campo [20:16] da instrução
	1	Registrador de destino é o campo [15:11] da instrução
memRead	0	Não será feita leitura na memória de dados
	1	Indica que será efetuada uma operação de leitura na memória de dados
memToRead	0	O dado que deverá ser armazenado no banco de registradores é proveniente do resultado da operação na ALU
	1	O dado que deverá ser armazenado no banco de registradores é proveniente da memória de dados
memWrite	0	Não será feita escrita na memória de dados
	1	Indica que será efetuada uma operação de escrita na memória de dados
aluSrc	0	Indica que o segundo operando na ALU deve ser o dado no registrador RT
	1	Indica que o segundo operando na ALU deve ser o valor do imediato com sinal estendido
regWrite	0	Indica que um dado deve ser escrito no banco de registradores
	1	Não haverá escrita no banco de registradores

Fonte: Próprio autor

3.2.2 Unidades controladoras de conflitos

3.2.2.1 Hazard Detection Unit

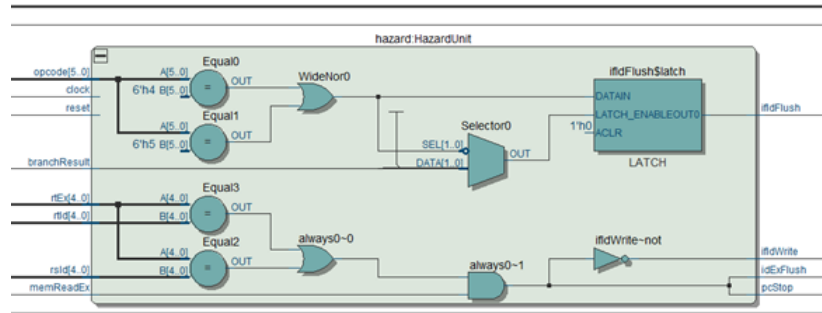
Esta unidade é responsável pela detecção de conflitos estruturais, conflitos de dados e conflitos de controle. No projeto desenvolvido, a memória possui dois canais de leitura, logo o Hazard não tem a necessidade de analisar conflitos estruturais.

Esta estrutura é responsável por gerar os stalls(bolhas) quando uma instrução buscada tem dependência de uma instrução anterior a ela e esta instrução é uma instrução Load, no caso, é necessário uma bolha pois não seria possível antecipar o dado para a instrução visto que o dado ainda não teria sido carregado da memória.

Quanto ao conflito de controle, essa estrutura verifica se um desvio devido ao branch foi tomado, e ativa os sinais responsáveis por limpar a instrução já buscada(considerando que o desvio não seria tomado) e recarregar a nova instrução no endereço de desvio.

A Figura 8 representa o bloco *Hazard unit*.

Figura 8 – Bloco de controle de *hazard* no circuito



Fonte: Próprio autor

Segue um Pseudo-Código do funcionamento da *Hazard Unit*:

SE: (VaiLerMemoria e ((rsId == rtEx) || (rtId == rtEx)))

ENTÃO:

1. Não atualiza PC
2. Flush em Registradores interestagios IDEX
2. Desabilita escrita em Redistradores interestagios IFID

//Stall em instruções de desvios

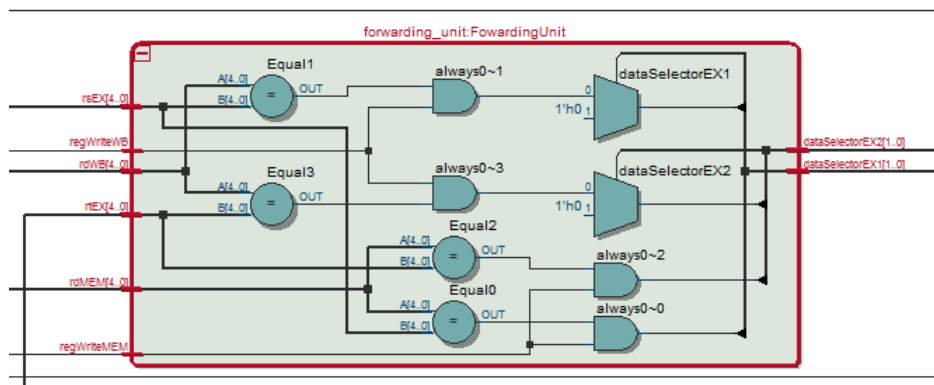
SE: (Instrução = BEQ ou Instrução = BNE) E Desvio foi tomado

ENTÃO:

1. Flush em Registradores interestagios IFID
2. Desabilita escrita em registradores IFID

3.2.2.2 Forwarding Unit

Figura 9 – Bloco da *Forwarding unit* no circuito



Fonte: Próprio autor

A unidade de adiantamento, cujo bloco é apresentado na Figura 9, seleciona os *inputs* corretos da ULA do estágio EX e verifica se existe ou não um conflito. Não havendo, os operandos da unidade lógica aritmética virá do banco de registradores, no entanto, caso exista um conflito, os dados destinados à ULA serão provenientes do registrador interestágio EX/MEM ou MEM/WB. Esse controle é dado a partir de seletores nomeados dataSelectorEX1 e dataSelectorEX2, ambos com 2 bits.

Segue um Pseudo-Código da *Forwarding Unit* para ilustrar melhor seu funcionamento:

```
//Antecipação de dados para RS
```

```
SE: EstagioMemVaiSalvarEmBancoDeRegistradores E rsEX == rdMEM
```

```
ENTÃO:
```

```
1. Antecipar Dado RS do estagio MEM para estagio EX
```

```
SE NÃO, CASO: EstagioWBVaiSalvarEmBancoDeRegistradores E rsEX == rdWB
```

```
ENTÃO
```

```
1. Antecipar Dado RS do estagio WB para estagio EX
```

```
SE NÃO:
```

```
1. Não Antecipar dados
```

```
//Antecipação de dados para RT
```

```
VIDE RS
```

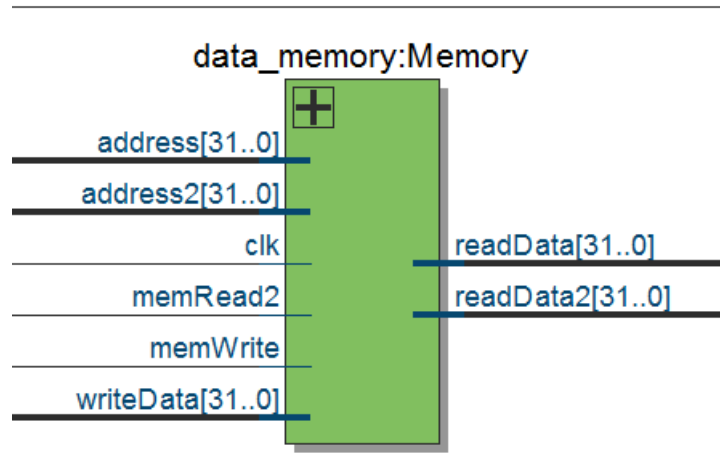
3.2.3 Memória

O Quartus possui a funcionalidade de identificar um template padrão de memória de modo que a memória descrita em verilog seja assimilada pela placa e, ao invés de sintetizar utilizando suas LEs internas, ele redireciona para a interna da placa.

A estrutura interna da memória segue os requisitos do projeto e, portanto a arquitetura do MIPS, onde cada linha da memória possui 1 byte e o seu endereçamento é a partir de uma palavra de 2 bytes.

Uma vez que a memória era compartilhada, a memória possui dois barramentos de entrada e dois barramentos de saída, ambos de 32 para evitar conflito de dados entre os estágios de Instruction Fetch e Memory. A figura 10 exibe o bloco de memória sintetizado pelo quartus, com as suas respectivas entradas e saídas.

Figura 10 – Bloco de memória no circuito

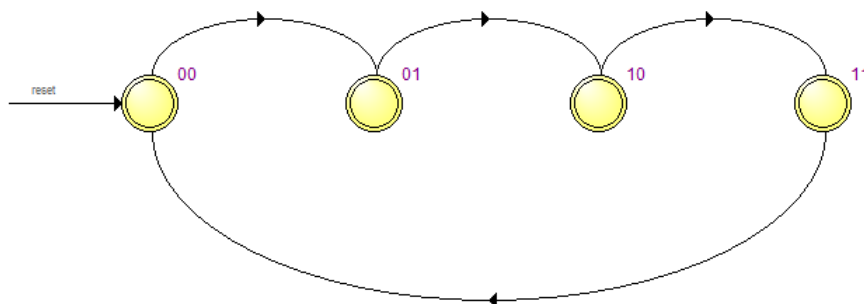


Fonte: Próprio autor

Uma vez que a memória e o processador tem tamanhos de palavra diferentes (a memória trabalha com palavras de 1 byte e o processador com palavras de 4 bytes), houve a necessidade de se criar uma estrutura anexa a estrutura de memória de maneira que os dados oriundos do processador pudessem ser armazenados na memória sem erros.

Para realizar esta função foi criado um acumulador, com o clock 10 vezes mais rápido do que o clock do sistema (o aumento do clock acontece para que os dados já estejam prontos quando o clock do sistema pulsar). Seu objetivo era capturar as palavras de 4 bytes que chegavam para ser gravadas na memória, dividi-las em 4 partes e gravar cada parte em uma linha da memória, já para a saída o acumulador capturava 4 linhas consecutivas e formava a palavra de 4 bytes necessárias para a leitura da cpu. A Figura 11 exibe o diagrama de estados da máquina responsável por acumular os dados.

Figura 11 – Ilustração da máquina de estados da memória



Fonte: Próprio autor

4 RESULTADOS E DISCUSSÕES

4.0.1 Testes

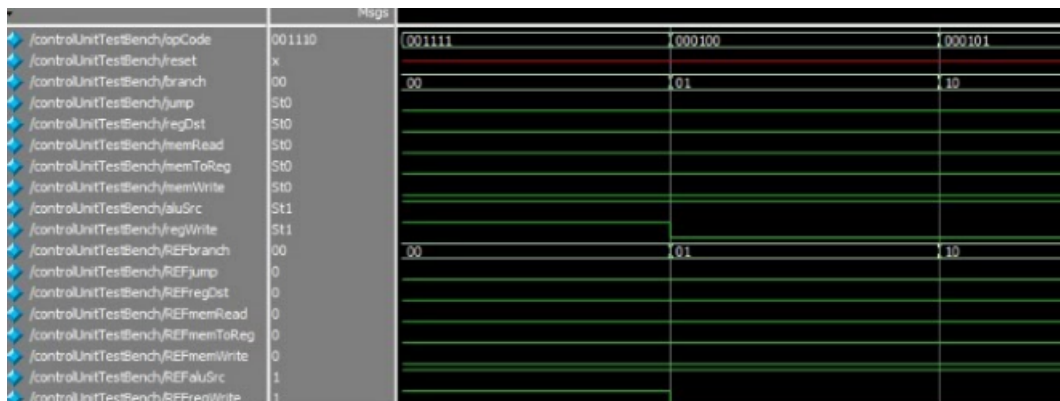
4.0.1.1 Visão geral

Com o objetivo de detectar possíveis erros de desenvolvimento das unidades do processador, foram desenvolvidos alguns testes que executavam as funcionalidades destas unidades de maneira individual, ou seja, não automatizadas. As unidades que tiveram testes foram a unidade de controle, o pc e seu respectivo somador, a unidade de lógica aritmética e a cpu em si. Este processo foi essencial para descobrir eventuais erros de lógica e de construção da descrição do circuito.

Pra construir os testes, foram utilizados como base os algoritmos desenvolvidos na etapa anterior do projeto, os seus binários foram gerados pelo montador e inseridos na memória do dispositivo através do comando *readmemb*, para cada ciclo de clock o teste era capaz de exibir todos os dados que estavam em cada pipeline tornando possível a depuração da unidade de controle, unidade de conflitos, unidade de adiantamento, registradores inter-estágios, extensor de sinal e da memória.

4.0.1.2 Teste da Unidade de controle

Figura 12 – Teste da unidade de controle no ModelSim



Signal	Value	Signal	Value	Signal	Value
/controlUnitTestBench/opCode	001110	/controlUnitTestBench/reset	x	/controlUnitTestBench/branch	00
/controlUnitTestBench/jump	St0	/controlUnitTestBench/regDst	St0	/controlUnitTestBench/memRead	St0
/controlUnitTestBench/memToReg	St0	/controlUnitTestBench/memWrite	St0	/controlUnitTestBench/aluSrc	St1
/controlUnitTestBench/regWrite	St1	/controlUnitTestBench/REFbranch	00	/controlUnitTestBench/REFjump	0
/controlUnitTestBench/REFregDst	0	/controlUnitTestBench/REFmemRead	0	/controlUnitTestBench/REFmemToReg	0
/controlUnitTestBench/REFmemWrite	0	/controlUnitTestBench/REFaluSrc	1	/controlUnitTestBench/REFregWrite	1

Fonte: Próprio autor

O teste da unidade de controle do processador, apresentado nas Figuras 12 e 13, buscou verificar se os sinais gerados pela unidade através da decodificação do opcode da instrução estavam corretos. Foram feitas entre todos os opcodes possíveis e verificando se a saída do modulo correspondia aos sinais esperados (definidos no teste). Foram encontrados alguns problemas na geração de sinais para instruções de jump, em virtude disso, o teste possibilitou a detecção e correção desse erro, a qual foi executada e a unidade foi aprovada mediante as avaliações.

Figura 13 – Teste da unidade de controle no ModelSim - 1

```
# *****PASS*****
# *****OPCODE:000100*****
# Branch: Esperado: 01, Encontrado: 01
# Jump: Esperado: 0, Encontrado: 0
# regDst: Esperado: 0, Encontrado: 0
# memRead: Esperado: 0, Encontrado: 0
# memToReg: Esperado: 0, Encontrado: 0
# memWrite: Esperado: 0, Encontrado: 0
# aluSrc: Esperado: 1, Encontrado: 1
# regWrite: Esperado: 0, Encontrado: 0
# *****PASS*****
# *****OPCODE:000101*****
# Branch: Esperado: 10, Encontrado: 10
# Jump: Esperado: 0, Encontrado: 0
# regDst: Esperado: 0, Encontrado: 0
# memRead: Esperado: 0, Encontrado: 0
# memToReg: Esperado: 0, Encontrado: 0
# memWrite: Esperado: 0, Encontrado: 0
# aluSrc: Esperado: 1, Encontrado: 1
# regWrite: Esperado: 0, Encontrado: 0
```

Fonte: Próprio autor

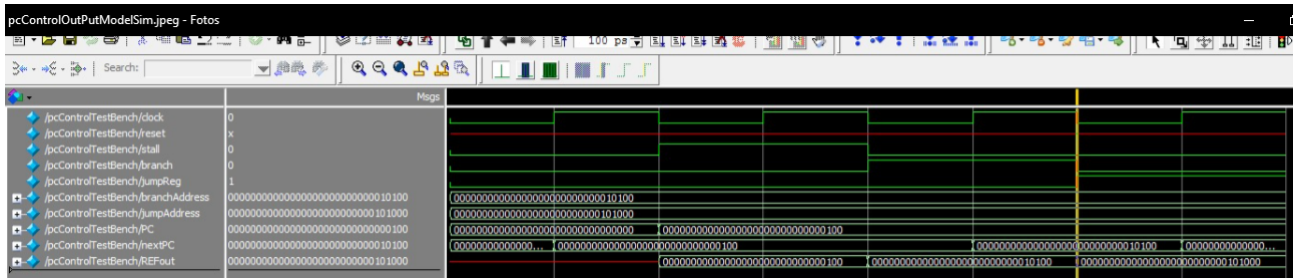
4.0.1.3 Teste da Unidade de controle de PC

Figura 14 – Teste da unidade de controle de PC - Testbench

```
# Top level modules:
#   pcControlTestBench
# End time: 19:06:52 on Sep 20, 2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# vsim -t lps -L altera_ver -L lpm_ver -L sgate_ver -L altera_mf_ver -L altera_lnsim_ver -L cycloneive_ver -L rti_work -L work -vop
# vsim -t lps -L altera_ver -L lpm_ver -L sgate_ver -L altera_mf_ver -L altera_lnsim_ver -L cycloneive_ver -L rti_work -L work -vop
# Start time: 19:06:55 on Sep 20, 2016
# Loading work.pcControlTestBench
# Loading work.pc_control
# ** Warning: (vsim-3017) D:/Downloads/Tec499-T03-Antares-R2-master/ProcessorAntares/QuartusProject/./sim/tb/pcControlTestBench.v(
#   Time: 0 ps Iteration: 0 Instance: /pcControlTestBench/pc File: D:/Downloads/Tec499-T03-Antares-R2-master/ProcessorAntares/sr
# ** Warning: (vsim-3722) D:/Downloads/Tec499-T03-Antares-R2-master/ProcessorAntares/QuartusProject/./sim/tb/pcControlTestBench.v(
#
# add wave 1
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Teste do PC
# PASS Saída esperada      4, Saída encontrada      4
# PASS Saída esperada      4, Saída encontrada      4
# PASS Saída esperada     20, Saída encontrada     20
# PASS Saída esperada     40, Saída encontrada     40
```

Fonte: Próprio autor

Figura 15 – Teste da unidade de controle de PC - Modelsim



Fonte: Próprio autor

O teste da unidade de controle de PC, podendo ser visualizado através das Figuras 14 e 15 buscou verificar se os sinais que manipulam o estado do PC, entre eles, sinais de Jump, Branch e Stall, satisfaziam o resultado esperado, que é conferir se o valor do PC após o ciclo era o especificado pelo endereço de branch, endereço de jump, pc anterior +4 ou o mesmo valor. O teste foi feito variando os sinais de controle de entrada e emitindo um pulso de clock. O teste comprovou a exatidão do circuito controlador do program counter(PC) na qual a cada novo ciclo de clock, o valor de PC era atualizado de acordo com seus sinais de controle, ou se mantia o mesmo dependendo do sinal de *Stall*.

4.0.1.4 Teste da Unidade Lógica Aritmética

Os testes da unidade lógica aritmética foram feitos através de vetores de testes, testes que geraram valores aleatórios e testes *HardCodes*, visando validar os valores que a ULA estava disponibilizando após seu processamento. Os resultados foram satisfatórios pois obtiveram 100 por cento de sucesso, comprovando o bom funcionamento dessa unidade, a qual é de extrema importância para o processador.

4.0.1.5 Teste da CPU

No teste geral do processador existem 2 módulos principais, um módulo da memória e outro da cpu. Nesse teste é feita uma interligação entre a memória e o processador e verificará os resultados do arquivo de teste que é passado como parâmetro. Esse processo mostra em cada ciclo de clock quais são os estados e quais são os dados que estão em cada estágio do pipeline, podendo apresentar possíveis erros que possam acontecer em registradores, na unidade de controle e em unidades em geral, pois trata-se de um teste abrangente em termo de quantidade de módulos testados.

4.0.2 Síntese lógica do projeto

Através da síntese lógica do Quartus, foi possível determinar os recursos utilizados pela CPU na FPGA, a descrição do circuito ocupou um total de 482 das 1803 LABs disponíveis (corresponde a 27%), 6079 das 28848 LEs disponíveis (correspondente a 21%) destas 4748 geraram

circuitos combinacionais sem registradores, 456 foram utilizadas apenas como registradores e 875 como circuitos combinacionais com registradores. Dentre as LEs utilizadas 3322 eram de 4 entradas, 1963 de 3 entradas e 338 com duas ou menos entradas, 4302 dos LEs estavam operando no modo normal e 1321 no modo aritmético

5 CONCLUSÃO

O produto resultante de todo o trabalho foi um processador com paralelismo a nível de instrução atrelado a uma memória compartilhada de 64KB. A efetividade do produto não se dá em 100 por cento devido falhas em casos de execução em algumas instruções, tais como instruções de branch que podem ser executadas incorretamente caso a instrução logo anterior use um dos seus registradores de leitura como registrador de destino.

O problema pode ser resolvido inserindo *forwarding* no estágio *Instruction Decode* ou usando o circuito de antecipação de *branch* apenas como um preditor e fazer a operação efetiva do *branch* no estágio EX, onde será possível fazer *forwarding* de acordo com o projeto já implementado.

Vale salientar que através dos testes desenvolvidos, principalmente o teste geral da cpu, é possível detectar tais erros juntos a outros futuros, o que torna totalmente viável a correção e o aprimoramento do produto a curto prazo.

Como requerido no texto prévio de requisição do processador parcial, a cpu desenvolvida só deveria ser capaz de efetuar operações encontradas em testes específicos previamente elaborados pela mesma equipe desenvolvedora do processador, sendo assim, o projeto pode ser bem mais aperfeiçoado através da inserção de novas instruções. Também pode ser considerado um possível trabalho futuro o acréscimo de uma tabela de predição de branch, a qual melhoraria a performance da cpu.

REFERÊNCIAS

- AGAH, H. *Understanding Physical Synthesis and Timing Closure*. 2011. Understanding Physical Synthesis and Timing Closure. Disponível em: <http://www2.units.it/marsi/elettronica2/manuals/xc_physyn.pdf>. Acesso em: 19 sep. 2016. Citado na página 6.
- ALTERA CORPORATION. *Cyclone IV Device Handbook, Volume 1*. San Jose, CA 95134, 2016. 490 p. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-iv/cyclone4-handbook.pdf>. Citado na página 5.
- CARDOSO, W. *Arquitetura das fpgas*. 2011. Arquitetura das fpgas. Disponível em: <<http://wolgrandcardoso.blogspot.com.br/2011/12/arquiteturas-das-fpgas.html>>. Acesso em: 19 sep. 2016. Citado 2 vezes nas páginas 2 e 3.
- FONTANINI, J. A. M.; OLIVEIRA, C. de. *Dispositivos Lógicos e Programáveis*. 2013. Dispositivos Lógicos e Programáveis. Disponível em: <<http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/logica-programavel.pdf>>. Acesso em: 19 sep. 2016. Citado 3 vezes nas páginas 2, 3 e 4.
- FUNDAMENTOS da tecnologia FPGA. 2013. Fundamentos da tecnologia FPGA. Disponível em: <<http://www.ni.com/white-paper/6983/pt/>>. Acesso em: 19 sep. 2016. Citado na página 2.
- SORATO, E. *Sistemas Operacionais*. 8 p. Dissertação (Mestrado), 2008. Disponível em: <<http://www.lisha.ufsc.br/teaching/os/ine651600-2008-2/work/sorato.pdf>>. Citado na página 1.
- VIEIRA, J. C. *Síntese Física*. 2008. Síntese Física. Disponível em: <<http://paginas.fe.up.pt/~jcf/ensino/disciplinas/mieec/pcvlsi/2007-08b/phy.pdf>>. Acesso em: 19 sep. 2016. Citado na página 6.
- VIEIRA, J. C. *Síntese Rtl*. 2009. Síntese Rtl. Disponível em: <<http://paginas.fe.up.pt/~jcf/ensino/disciplinas/mieec/pcvlsi/2009-10/syn.pdf>>. Acesso em: 19 sep. 2016. Citado na página 5.