Relatório - Problema 1

CRC-32

UEFS

2018.1

Histórico de Revisões

Date	Descrição	Autor(s)
01/05/2018	Introdução	Higor Vital
02/05/2018	Visão Geral	Antônio Raian
02/05/2018	Descrição Geral	Higor Vital
02/05/2018	Resultados	Diogo Lima
03/05/2018	Revisão Final	Diogo Lima

SUMÁRIO

1 | Introdução

1. Propósito do Documento

Este documento descreve a arquitetura do projeto CRC-32 , referente ao primeiro problema da disciplina de MI-Sistemas Digitais. Para o primeiro problema, os alunos foram incumbidos da tarefa de programar um código capaz de gerar um código verificador do tipo CRC-32 para sequências de 1KB de dados.

Nas próximas páginas, estão incluídas especificações do circuitos internos e máquinas de estados de cada componente. Também são apresentados diagramas de classe, definições de entrada e saída e diagramas de temporização. O principal objetivo deste documento é definir as especificações do projeto CRC-32 e prover uma visão geral completa do mesmo.

2. Stakeholders

Nome	Papel/Responsabilidades
Antonio Raian	Programador/Projetista
Diogo Lima	Programador/Projetista
Higor Vital	Programador/Projetista
Anfranserai Dias	Tutor

3. Visão Geral do Documento

O presente documento é apresentado como segue:

- **Arquitetura do Processador** – Este capítulo apresenta uma visão geral da arquitetura, com foco em entrada e saída do sistema e arquitetura geral do mesmo;

4. Acrônimos e Abreviações

Acronym	Description
RISC	Reduced Instruction Set Computer
GPR	General Purpose Registers
FPGA	Field Gate Programmable Array
CRC	Cyclic Redundancy Check
LED	Light Emitting Diode
KB	Kilo Bytes

2 | Visão Geral da Arquitetura

1. Diagrama da arquitetura do Nios II

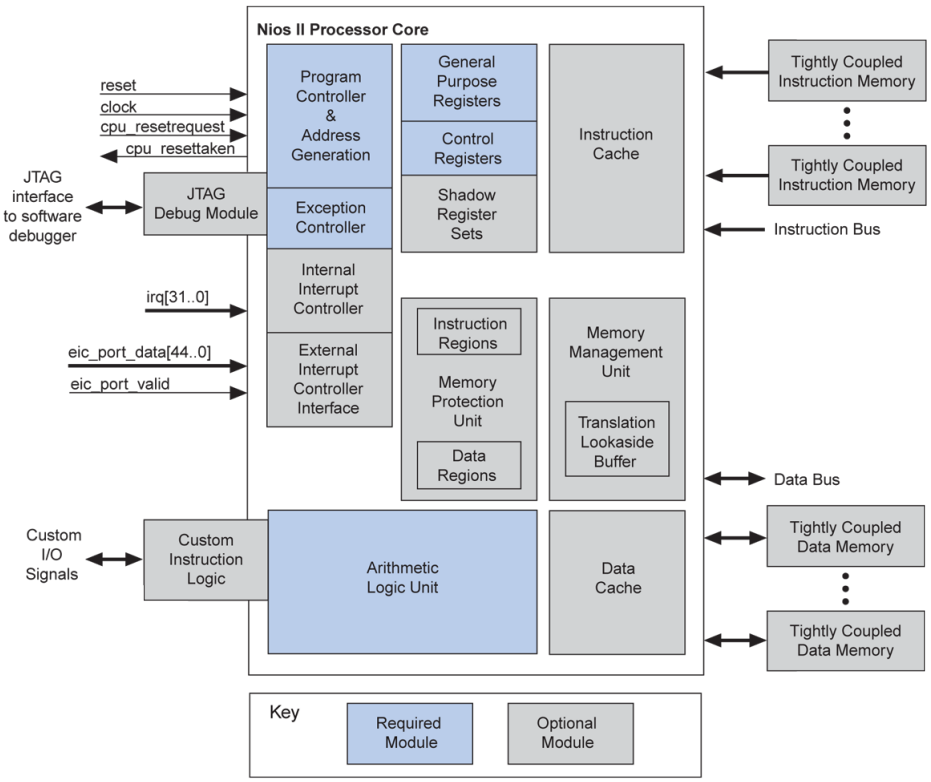

Figura 2.1: **Imagem da estrutura arquitetural do NIOS II;**

2. Características do Nios II

Algumas das características principais do Nios II incluem:

- Processador RISC com pipeline;
- 32 registradores de propósito geral;
- 3 formatos de instrução;
- Instruções, endereçamento e data path de 32bits;

3. Familia NIOS

3.1. NIOS II/f (fast)

O NIOS II/f foi desenvolvido para fornecer a performace máxima do seu tamanho de core. Essa versão do NIOS possui:

- Separação de memória para instruções e cache de dados (512 B to 64 kB);
- MMU opicional;
- Acesso a até 2 GB de espaço externo adicional;
- Memória fotemente acoplada para instruções e dados;
- Seis estágios de pipeline para chegar ao máximo de DMIPS/MHz;
- Até 256 instruções customizadas e ilimitados aceleradores de hardware;
- Depurador JTAG;
- Possibilita aprimoramentos para o JTAG, incluindo interrupção de hardware, acionadores de dados e rastreamento em tempo real;

3.2. NIOS II/e (economy)

As especificações do *NIOS II economy* fo concebida para o menor uso possível das FPGAs. Essa versão possui:

- Acesso a até 2 GB de espaço externo adicional;
- Depurador JTAG;
- Sistemas completos em menos de 700 Elementos lógicos (LEs);
- Aprimoramentos opicionais para o depurador;
- Até 256 instruções customizadas;
- Não é necessária uma licensa para utilizar;

3.3. NIOS II/s (standard)

Desenvolvido para manter o desempenho e o custo num mesmo nível, o *NIOS II standard* possui:

- Cache de instruções;
- Acesso a até 2 GB de espaço externo adicional;
- Memória fotemente acoplada para instruções;
- Estágio cinco de pipeline;
- Até 256 instruções customizadas;

- Depurador JTAG;
- Possibilita aprimoramentos para o JTAG, incluindo interrupção de hardware, acionadores de dados e rastreamento em tempo real;

3 | Arquitetura do conjunto de instruções

No Nios II há três tipos de formatos de instruções: Tipo I, Tipo R e Tipo J.

1. Tipo I

A característica do Tipo I é que contém um valor imediato dentro da palavra que compõe a instrução. Intruções do Tipo I contém:

- Um campo para as operações, de 6 bits;
- Dois campos para registradores, de 5 bits, A e B;
- Um campo de 16 bits para o dado imediato;;

Na maioria dos casos, os campos A e Imm16 especificam os os operandos e o campo B o registrador de destino. IMM16 é considerado com sinal exceto por operações logicas e comparações se sinal.

Instruções do Tipo I incluem operações aritméticas e logicas como *addi* e *andi*; operações de branch; operações de carregamento e armazenamento; e operações de gerenciamento de cache.

2. Tipo R

Esse tipo de instrução define operações entre registradores especificados. Geralmente usa-se 3 registradores A, B e C, esses registradores são, respectivamente, 2 registradores origem e 1 de destino.

Essas instruções são compostas por:

- Um campo de 6 bits para OPcode;
- Três campos de 5 bits para registradores (A,B e C);
- Um campo, para extenção do OPcode, de 11 bits;

Algumas instruções desse tipo colocam um valor padrão nos 5 bits menos significativos do OPX, e todas colocam os bits não usados como 0.

Ainda encontramos algumas instruções aritméticas e lógicas como *and* e *nor*, operações de comparação *cmpeq* e *cmplt*.

3. Tipo J

Instruções do Tipo J contém:

- Um campo de 6 bits para OPcode;
- Um campo de 26 bits para dados;

Intruções do Tipo J, como *call* e *jmpi*, transferem execução para qualquer lugar no raio de 256Mb.

4 | Resultados

1. Códificação CRC-32

O código gerado em linguagem Assembly, compatível com o processador Nios II, pode ser dividido em três etapas principais. Cada etapa é responsável por uma parte fundamental do código, e serão explicados à seguir.

1.1. Pré-processamento

Este é o bloco inicial do código, que compreende todas as operações realizadas antes do *main* e a própria função *main*. Nesta parte, são escolhidos os registradores que serão usados para os cálculos e decidido o valor dado à sequência de 1KB que será aplicada ao código. O polinômio usado nas operações da geração do CRC-32 também é inserido aqui.

Na *main*, os dados são colocados na memória. Dois registradores são utilizados para operar no dado: O primeiro (r3) armazena os 32 bits onde será feito a XOR; e segundo (r5) os 32 bits seguintes. O registrador (r8) armazena a quantidade de partes de 32 bits que compõem o dado, no caso, 250. O registrador (r4) armazena a quantidade de bits restantes no registrador auxiliar. O registrador (r7) armazena o polinômio de 32 bits. O polinômio na verdade tem 33 bits, mas sabe-se que o primeiro bit é sempre 1, então devido a limitação dos registradores, somente o 32 bits menos significativos são armazenados.

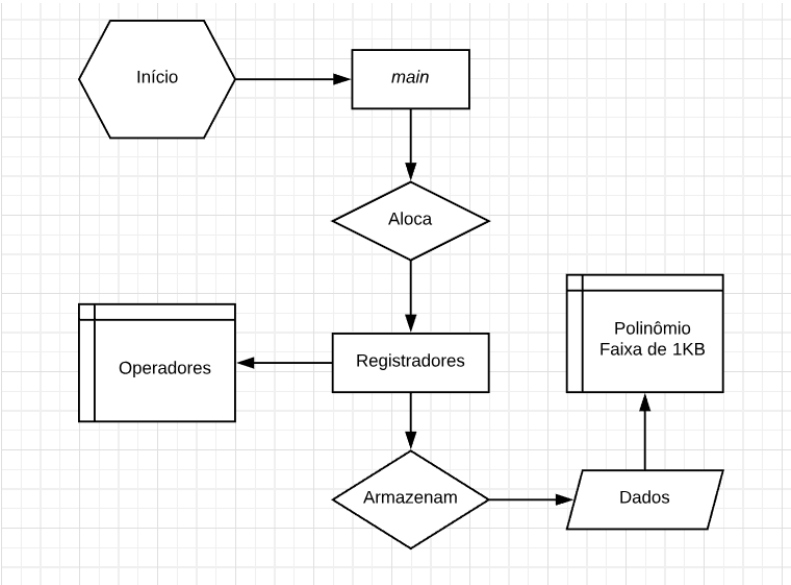

Figura 4.1: Fluxograma do pré-processamento.

1.2. Loop de Cálculo

Este é o cerne do código, compreendendo três funções que, juntas, fazem o cálculo do código de verificação do CRC-32.

Ao entrar em *loop*, é verificado a quantidade de bits restantes no registrador auxiliar, caso seja 0, o código entra em *resetar*. Caso contrário é verificado se o valor no registrador principal é negativo ou positivo. Se for negativo, quer dizer que o primeiro bit é 1, e entra no bloco *calculo*. Caso seja positivo, o primeiro bit é 0, e a registrador é simplesmente deslocado para a esquerda em 1 bit. Toda vez que o registrador é deslocado, é verificado se o primeiro bit do registrador auxiliar é 0 ou 1. No caso de ser 1, é feita uma OR com o valor imediato 1, para que esse bit seja adicionado no final do registrador principal. O loop é realizado até não haja mais bits no registrador auxiliar, e até que todos os dados na memória tenham sido percorridos.

No bloco *resetar* é checada a quantidade restante de partes de 32 bits do dado de entrada. No caso de ser maior que 0, os próximos 32 bits na memória são carregados no registrador auxiliar. Quando a quantidade é 1, nada é adicionado ao registrador auxiliar para que o cálculo dos próximos 32 bits sejam feitos com apenas 0s. Isso é feito pois no cálculo de CRC-32 é adicionado ao final da mensagem uma sequência de bits 0s com 1 bit a menos do polinômio, que é de 33 bits.

Quando o primeiro bit do registrador auxiliar é 1, é realizado o cálculo da XOR, bloco *calculo*. Como no cálculo do CRC, o primeiro bit sempre zera, o registrador principal é deslocado em 1 bit, e assim como descrito anteriormente, o primeiro bit do auxiliar é adicionado ao seu final. É então feito o cálculo da XOR do registrador principal com o polinômio. O resultado é armazenado no registrador principal.

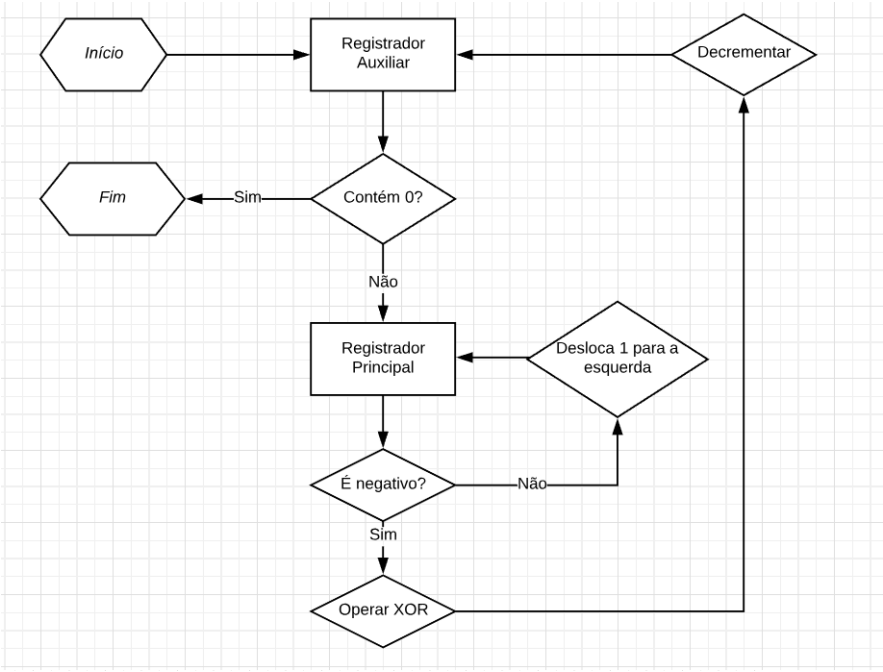

Figura 4.2: Fluxograma do processo de cálculo.

1.3. Saída de Dados

O último bloco do código é responsável pela saída de dados. A saída de dados esperada para a descarga do código numa placa FPGA, é feita à partir de quatro LEDs disponívels na placa Cyclone IV, disponível em laboratório da UEFS.

No módulo *led*, o resultado é armazenado na posição de memória correspondente aos LEDs. Como se está limitado a utilização de 4 LEDs, o resultado é exibido de 4 em 4 bits. Para o deslocamento desses bits, é utilizado um botão. O valor do botão fica armazenado em uma posição de memória específica. Como a frequência que o botão é apertado é muito maior que a frequência com que o botão é apertado e liberado por um ser humano, o deslocamento é muito rápido.

Para lidar com isso foi utilizado a seguinte estratégia: Em vez de deslocar quando o botão é apertado, fazê-lo quando o botão é liberado, dessa maneira, o código entra num loop, do qual só sai quando o botão é apertado, seguido de um loop do qual o só se sai quando o botão é liberado. Ao apertar o botão 8 vezes a mensagem completa pode ser vista. Ao apertar mais vezes, a mensagem começa a se repetir.

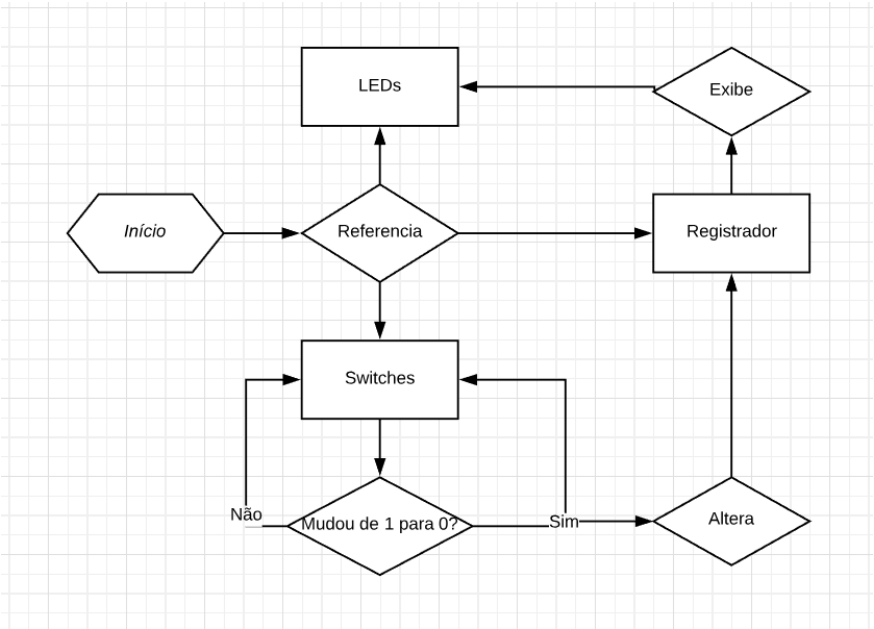

Figura 4.3: Fluxograma da saída de dados.

2. Área total ocupada pelo circuito em função dos elementos internos do dispositivo FPGA

- Número total de elementos lógicos: 1687/28848 (6%)
  Combinacionais sem registradores: 778
  Somente registradores: 95
  Combinacionais com registradores: 824

- Elementos lógicos por modo

Modo normal: 1465
Modo aritmético: 137

- Total de registradores: 919/30431 (3%)
  Dedicados: 919/28848 (3%)
  Registradores de entrada e saída: 0
- Total de pinos: 17/329 (5%)
  Clock: 1/7
  Pinos de entrada dedicados: 3/9
- Total de memória bits: 44032/608256 (7%)
- Clocks globais: 3/20 (15%)
- JTAG: 1/1
- PLL: 0/4
- Total de LABs: 124/1803 (7%)