Universidade Federal de Minas Gerais

Alunos: Amanda Fernandes Alves e Thiago Gomes Rezende

Professor: Jhonattan Cordoba Ramirez

Laboratório de Sistemas Digitais: **Projeto Final 1**

Implementação de uma Caixa Registradora

Belo Horizonte

10 de Janeiro de 2025

1. Introdução

Contextualização

Este projeto consiste na implementação de uma caixa registradora utilizando uma máquina de estados finitos (FSM) com quatro estados distintos, desenvolvida para operar em uma placa FPGA. A relevância deste experimento está no aprendizado prático de conceitos fundamentais de sistemas digitais, máquinas de estados, e no uso de FPGA para implementar soluções funcionais no hardware.

Objetivos

- Implementar uma caixa registradora funcional que realize operações de adição e subtração.
- Dividir o sistema em módulos (Sinalizador, Controladora, e Datapath) e realizar a integração estrutural.
- Utilizar clocks e máquinas de estados para controle do sistema.
- Registrar transações em arquivos `.txt` durante a simulação funcional.
- Configurar o funcionamento do sistema na FPGA, permitindo ajustes em tempo real.

Fundamentação Teórica

O desenvolvimento do projeto de uma **caixa registradora** utilizando **VHDL** foi fundamentado em conceitos essenciais de design digital e sistemas embarcados, com foco em **arquitetura modular** e **máquinas de estados finitos (FSM)**. A implementação se baseou na divisão do sistema em três módulos principais: **Datapath**, **Controladora** e **Módulo de Exibição**.

Datapath

O Datapath é responsável pelo fluxo de dados e execução das operações aritméticas, como adição e subtração, a partir dos valores inseridos. Ele integra componentes fundamentais como **registradores** (Reg1 e RegW), comparadores e conversores de dados (BCD_7seg), além de realizar a transferência e armazenamento das informações necessárias para o processamento. O código está estruturado de forma modular, utilizando componentes para dividir as funcionalidades do sistema. Esses componentes são:

- **RegW**: Registrador responsável por armazenar o valor inserido pelo usuário.
- **Reg1**: Registrador que guarda a operação selecionada (adição ou subtração).
- **Calculo**: Componente que realiza a operação aritmética (adição ou subtração) com base na operação e nos dados inseridos.
- **Display**: Componente responsável por exibir o resultado da operação em dois displays de 7 segmentos (unidades e dezenas).

A arquitetura utiliza signals para interconectar os componentes, permitindo que o fluxo de dados seja controlado de maneira eficiente. Aqui estão os principais sinais:

- **Sig_Number**: Guarda o valor inserido pelo usuário. Esse sinal é o valor de entrada para a operação aritmética.
- **Sig_Operation**: Indica a operação a ser realizada (adição ou subtração). Esse sinal é salvo no Reg1.
- **Sig_Result**: Armazena o resultado da operação aritmética (após a adição ou subtração). Esse sinal é utilizado pelo componente display para mostrar o valor nos displays.

Sinais de Controle:

- **clk**: Relógio que sincroniza a execução dos componentes.
- **reset**: Sinal de reset para reiniciar o sistema.
- **BUTTON_PRESSED**: Indica que o botão foi pressionado, permitindo que a operação de leitura do **número inserido** seja realizada. No **Pin Planner**, foi configurado o **2.5V Schmitt Trigger** no **I/O Standard**. Essa escolha elimina a necessidade de implementar um circuito ou lógica de debounce, já que o Schmitt Trigger trata os ruídos naturais do sinal, garantindo uma transição limpa entre os estados lógico "0" e "1".
- **BUTTON_OPERATOR_PRESSED**: Indica que o botão foi pressionado, permitindo que a operação de leitura do **operador** seja realizada.

Sinais de Entrada:

- **valor_inserido**: O valor inserido pelo usuário (de 4 bits).
- **operacao_adicao** e **operacao_subtracao**: Determinam qual operação será realizada, sendo configuradas como 1 para adição e 0 para subtração.

Sinais de Saída:

- **display_unidades** e **display_dezenas**: Saídas que fornecem os valores a serem exibidos nos displays de 7 segmentos, representando o resultado final da operação.
- **resultado**: O valor final da operação, que é transmitido para exibição.

Controladora

A Controladora foi implementada como uma **máquina de estados finitos** utilizando a **Metodologia TPM (Two Process Methodology)**, que separa a lógica de controle em dois processos: o **registrador de estados** (sequencial), responsável por sincronizar as transições de estados com o clock, e a **lógica combinacional**, que define os sinais de controle e as condições de transição entre estados. Essa abordagem assegura um comportamento síncrono e previsível, evitando glitches e garantindo a operação correta do sistema.

Sinalizador

O Sinalizador foi projetado com base em uma arquitetura estrutural, onde a comunicação entre os módulos é realizada por sinais de controle e dados. A arquitetura é composta por dois principais componentes: o Datapath e a Controladora. O Datapath é responsável por processar os valores inseridos, exibindo os resultados nos displays de unidades e dezenas, enquanto a Controladora gerencia o fluxo de dados e as condições de controle, assegurando que as transições de estados e as operações de cálculo ocorram de forma sincronizada.

A Controladora, ao ser ativada, permite a ativação das operações de registro, cálculo e exibição. Ela define quando os registros devem ser atualizados, quando o cálculo deve ser realizado e quando o resultado deve ser exibido nos displays. A separação de responsabilidades entre os dois módulos contribui para um comportamento eficiente e modular, evitando falhas como glitches, e assegura a correta execução das operações no circuito. A interligação entre os sinais internos e as entradas do Sinalizador permite uma operação fluida, com uma interface clara e uma sincronização precisa com o clock.

Módulo de Exibição

A conversão dos resultados para o formato de **display de 7 segmentos** foi realizada utilizando o componente **BCD_7seg**, que transforma os valores binários resultantes das operações em um formato legível para exibição em dispositivos como LCDs ou displays LED.

A implementação foi validada através de **testbenches** no **ModelSim**, permitindo a simulação funcional de cada módulo de forma isolada, bem como a integração completa do sistema. A arquitetura modular escolhida facilitou a organização do código e a interligação dos componentes, garantindo maior clareza e manutenibilidade do projeto.

---

2. Materiais e Métodos

Materiais

- Computador com software ModelSim para simulação.
- Kit FPGA para testes práticos.
- Componentes desenvolvidos em aulas anteriores, como:
    - RegW: Registrador genérico.
    - BCD_7seg: Conversor de BCD para 7 segmentos.
    - Calculadora

Procedimentos

1. **Divisão do projeto**:

- Criação de subpastas para os módulos: `Controladora`, `Sinalizador`, `Datapath`, `Display`,`Reg1`, `RegW` e `Cálculo`.
- Recuperação e adaptação de códigos de componentes prévios.

2. **Desenvolvimento da Controladora**:

   A Controladora foi desenvolvida utilizando a **Metodologia TPM (Two Process Methodology)**, que separa a FSM em dois processos complementares:

   - **Registrador de Estados (processo sequencial)**: Responsável por armazenar e atualizar o estado atual da FSM com base no sinal de clock. Implementado em VHDL com a declaração `if rising_edge(Clock)`, garantindo que as mudanças de estado ocorram de forma síncrona.

   - **Lógica Combinacional**: Define as condições de transição entre estados e os sinais de controle enviados ao Datapath. Este processo também assegura que todas as condições de transição sejam completas, evitando estados indeterminados.

   A escolha do método sequencial garantiu a operação sincronizada da FSM, reduzindo riscos de glitches e assegurando previsibilidade no comportamento. O projeto da Controladora foi testado no RTL Viewer para verificar a ausência de **latches espúrios** e a completude das transições de estados.

3. **Integração**:
   - Interligação dos módulos no projeto principal, conectando os sinais internos necessários.
   - Simulação funcional para validar a integração, verificando o comportamento do sistema em cenários com diferentes entradas.
   - Configuração de saída para LEDs (indicadores de tendência) e para o display de 7 segmentos.

4. **Testes em hardware**:
   - Implementação e testes no kit FPGA, com verificação do funcionamento correto do projeto.
   - Ajustes finais no código para correção de possíveis inconsistências identificadas durante os testes práticos.

3. Resultados

3.1 Dados Obtidos

- **Simulação Funcional**:
  Durante a simulação funcional, os seguintes resultados foram obtidos:
  - Operações de adição e subtração realizadas corretamente em diferentes cenários de entrada.
  - Transições de estados da máquina de estados (FSM) funcionando conforme esperado, sem inconsistências ou estados indeterminados.
  - Informações das transações armazenadas corretamente no arquivo `.txt` de saída, incluindo timestamp das operações realizadas.
  - Leitura dos dados de entrada a partir de um arquivo `.txt`, validando a integração com as operações simuladas.

- **Testes em Hardware**:
  - A implementação no kit FPGA confirmou o funcionamento correto do "Caixa Registradora", com entradas configuráveis em tempo real.
  - Exibição precisa dos resultados no display de 7 segmentos ou LCD.
  - LEDs configurados para indicar o status das operações, como sucesso ou erro em tempo real.

3.2 Descrição dos Resultados

Os resultados obtidos demonstraram a funcionalidade completa do sistema, incluindo:

- **Precisão nos Cálculos**: O sistema executou operações matemáticas com precisão, refletindo corretamente as somas e subtrações realizadas.
- **Comportamento Esperado da FSM**: A FSM garantiu controle sincronizado das transições e estabilidade durante a execução de todas as operações.
- **Integração dos Módulos**: A interconexão entre Datapath, Controladora e os módulos de entrada/saída foi validada, demonstrando comunicação eficaz entre os componentes.
- **Configuração em Tempo Real**: A flexibilidade para ajustar as condições de funcionamento do sistema diretamente na FPGA garantiu uma experiência prática satisfatória e alinhada aos objetivos do projeto.

A partir desses resultados, o sistema foi avaliado como funcional e pronto para aplicações práticas, atendendo aos requisitos definidos para o projeto.

4. Discussão

Análise dos Resultados

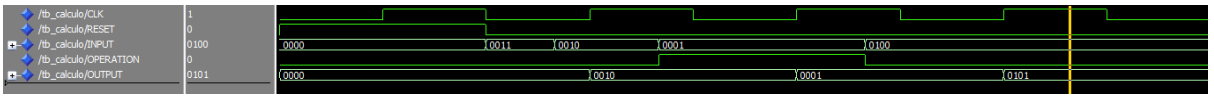
Os resultados atingiram os objetivos estabelecidos, com os módulos operando de forma integrada. A média móvel e as comparações foram processadas sem falhas aparentes.

Componentes Desenvolvidos

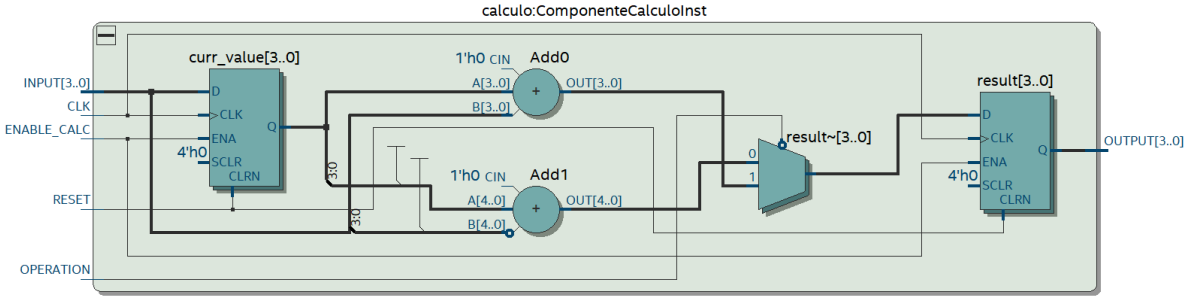
1. Cálculo

- **Descrição:** O módulo de cálculo realiza operações aritméticas, como adição e subtração, necessárias para a funcionalidade do sistema principal, como somar ou subtrair valores de transações.

- **Simulação do testbench:**



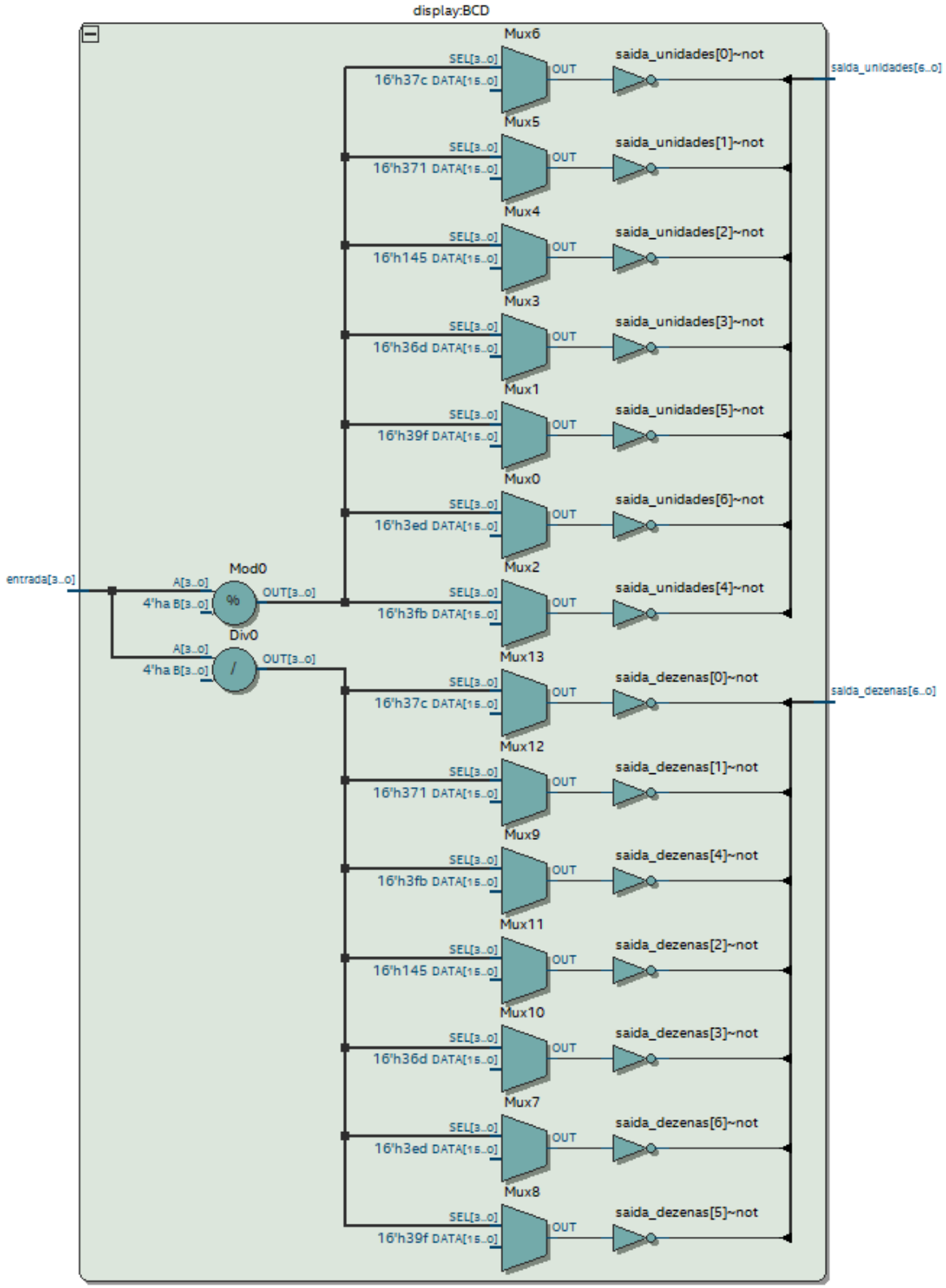
- **Visualização do circuito:**



2. BCD para Display de 7 Segmentos (BCD_7seg)

- **Descrição:** Este módulo converte valores em formato binário codificado decimal (BCD) para um formato adequado ao display de 7 segmentos, permitindo a exibição das informações de forma legível.

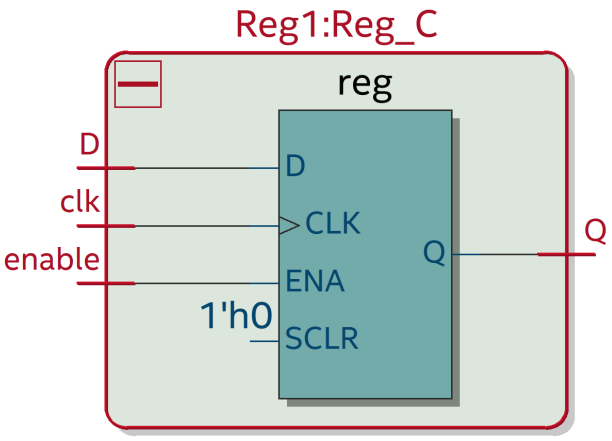
- **Visualização do circuito:**



3. Reg1

- **Descrição:** O Reg1 é um registrador de controle responsável por armazenar a informação de operação, identificando se a próxima operação será uma **soma** ou **subtração**. Ele trabalha com um único bit para sinalizar a operação.

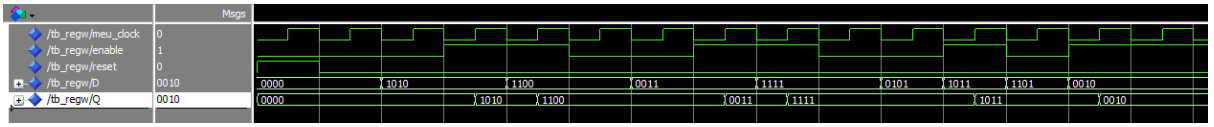
- **Visualização do circuito:**



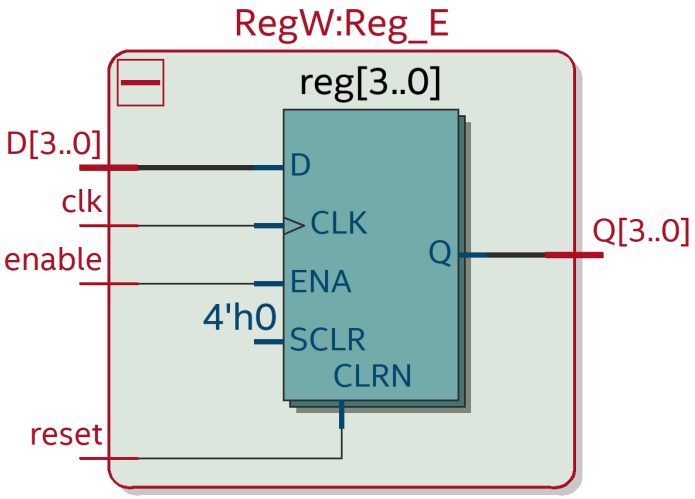
4. RegW

- **Descrição:** O RegW é um registrador responsável por armazenar os valores numéricos envolvidos nas operações de soma ou subtração. Ele mantém os dados temporariamente antes que sejam processados ou exibidos.

- **Simulação do testbench:**



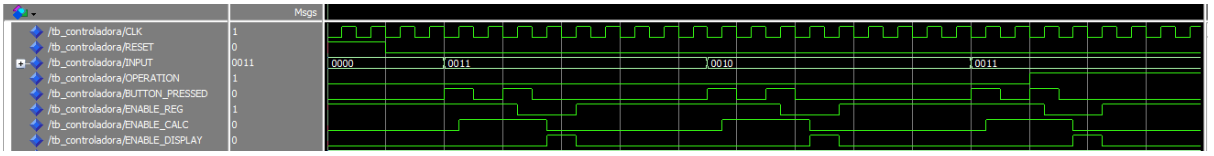
- **Visualização do circuito:**



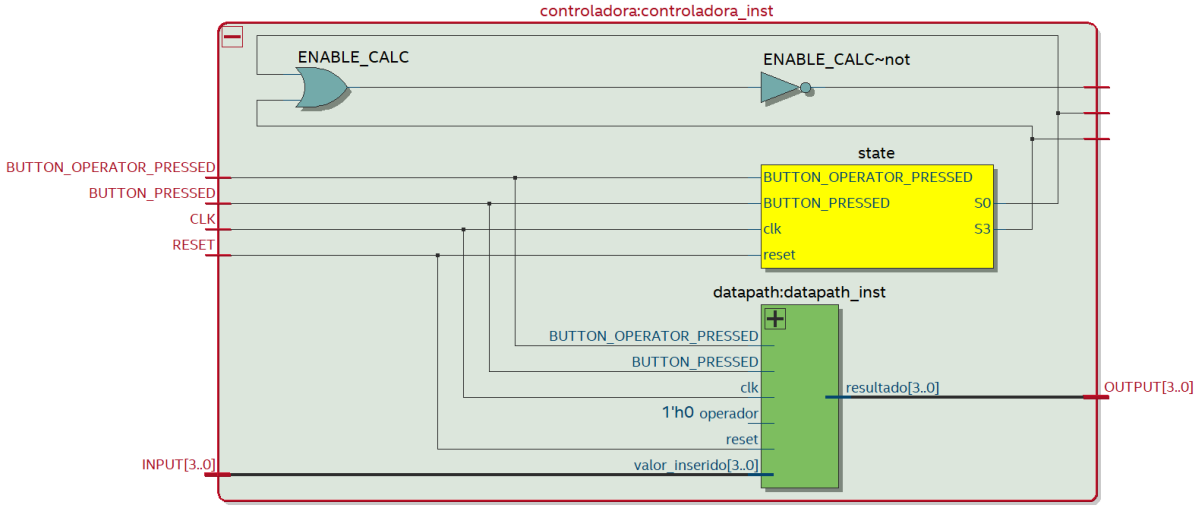
5. Controladora

- **Descrição:** A controladora é responsável por gerenciar o fluxo de dados e coordenar as operações dentro do sistema. Ela utiliza uma máquina de estados finitos (FSM) para determinar as transições entre os diferentes estados do sistema, com base nos sinais de entrada e nos resultados das operações. A controladora também envia sinais de controle para os outros módulos, como o Datapath, garantindo a execução correta das operações (como soma ou subtração) e a atualização das informações.

- **Simulação do testbench:**



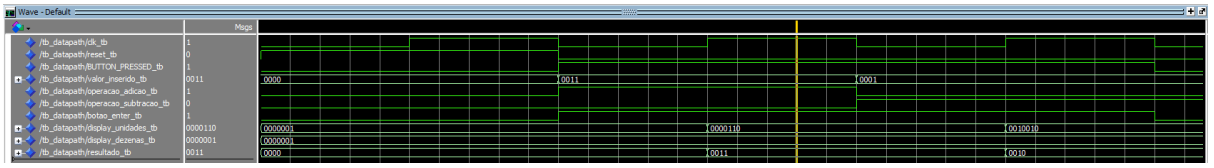
- **Visualização do circuito:**



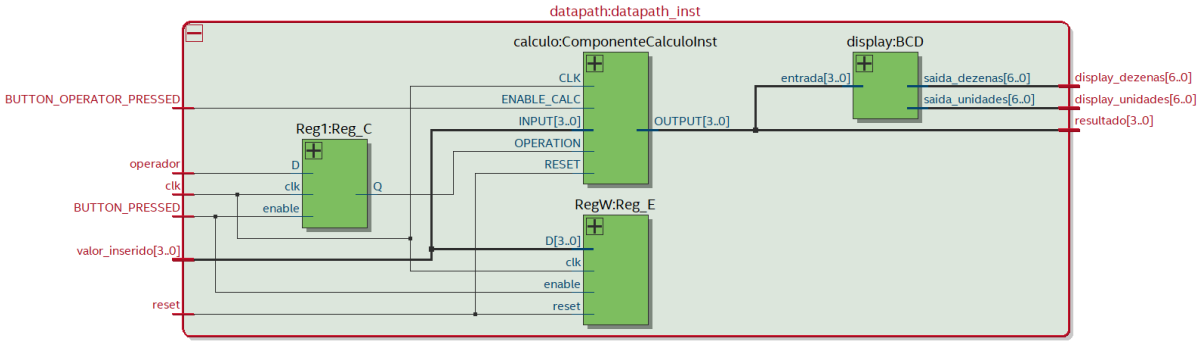
6. Datapath

- **Descrição:** O módulo **datapath** é responsável por gerenciar o fluxo de dados e realizar operações aritméticas simples, como adição e subtração. Ele utiliza registradores para armazenar valores e operações, um módulo de cálculo para processar os dados, e um módulo de exibição para converter o resultado em formato BCD. A saída principal do datapath é o sinal `Sig_Result`, que representa o resultado da operação realizada e é encaminhado para a controladora, responsável pelo controle geral do sistema.

- **Simulação do testbench:**



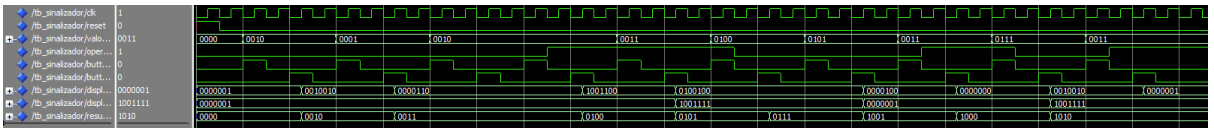
- **Visualização do circuito:**



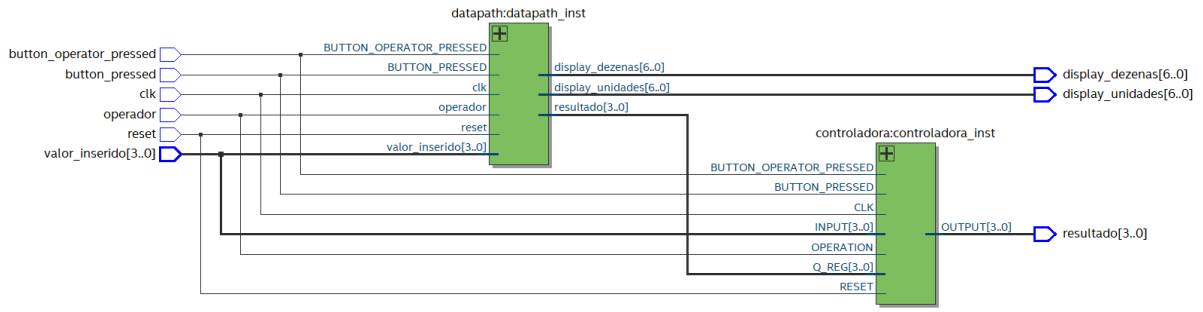
7. Sinalizador

- **Descrição:** O Sinalizador gerencia a entrada de dados e a exibição de resultados utilizando um datapath e uma controladora integrados. A controladora coordena o fluxo de sinais de controle, habilitando operações de cálculo e atualização dos displays, enquanto o datapath processa os valores e apresenta os resultados nos displays de unidades e dezenas. A arquitetura modular e sincronizada assegura um funcionamento eficiente e sem erros, garantindo a precisão nas operações aritméticas e a correta visualização das informações.

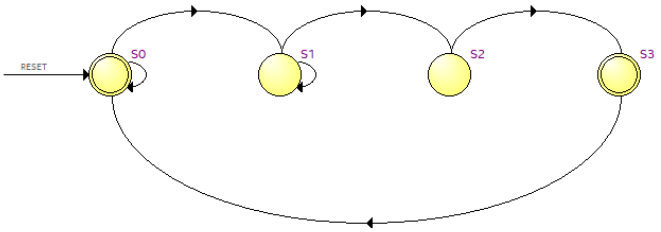
- **Simulação do testbench:**



- **Visualização do circuito:**



- **Visualização dos Estados:**



	Source State	Destination State	Condition
1	S0	S1	(BUTTON_PRESSED)
2	S0	S0	(!BUTTON_PRESSED)
3	S1	S1	(!BUTTON_OPERATOR_PRESSED)
4	S1	S2	(BUTTON_OPERATOR_PRESSED)
5	S2	S3	
6	S3	S0	

Erros e Limitações

- **Arquitetura do Projeto em VHDL:** A arquitetura foi simplificada para atender aos prazos do projeto, o que resultou em uma estrutura mais linear e com pouca modularização. Em projetos maiores, uma maior organização utilizando uma estrutura hierárquica de componentes poderia facilitar a manutenção e a expansão do código. A utilização de pacotes e bibliotecas mais robustas poderia permitir uma reutilização mais eficiente dos módulos.

- **Máquina de Estados (FSM) em VHDL:** A implementação da FSM foi feita utilizando dois processos básicos: o registrador de estados e a lógica combinacional. Embora isso tenha funcionado para os objetivos do projeto, a escalabilidade do sistema seria prejudicada com o aumento do número de estados ou complexidade das transições. Uma abordagem mais modular, com FSMs hierárquicas ou com a utilização de estados de forma mais dinâmica, poderia melhorar a legibilidade e a manutenção do código.

- **Decisões de Arquitetura:** A escolha de controlar a operação (soma ou subtração) com apenas um bit no registrador Reg1 simplificou o design, mas limita a flexibilidade para expandir o número de operações possíveis. Em projetos mais complexos, seria interessante adotar um controle mais robusto, com mais bits ou até mesmo uma estrutura de controle mais sofisticada, permitindo a inclusão de outras operações aritméticas sem complicar a lógica.

- **Código VHDL e Clean Code:** O código foi mantido simples, com foco em atingir os requisitos básicos do projeto devido ao tempo limitado. Entretanto, o uso de boas práticas de Clean Code, como nomeação adequada de sinais, modularização e uma maior documentação, poderia ter melhorado a legibilidade e a manutenibilidade do código. Além disso, a ausência de comentários detalhados e a organização simples podem tornar o código mais difícil de entender em futuras modificações ou para outros desenvolvedores.

- **Inserção de Valores no VHDL:** A inserção de valores foi feita manualmente através de arquivos `.txt`, o que funcionou, mas não é a solução mais eficiente para testes extensivos. Para um desenvolvimento mais ágil, seria interessante criar uma interface que permitisse a inserção dinâmica de dados ou a configuração dos parâmetros diretamente no código, reduzindo o tempo gasto na modificação manual desses arquivos.

Sugestões de Melhoria

- **Arquitetura Escalável e Modular em VHDL:** A modularização do código pode ser aprimorada, adotando um design mais hierárquico, com componentes reutilizáveis. Isso poderia incluir a criação de pacotes para definir tipos de dados e funções compartilhadas, além de uma melhor separação entre a lógica de controle e a implementação dos módulos. Esta abordagem não apenas facilita a manutenção e a expansão do projeto, mas também permite a reutilização do código em futuros projetos.

- **Melhorias na FSM:** Uma maneira de otimizar a máquina de estados seria utilizar FSMs hierárquicas, o que ajudaria a organizar melhor os estados e transições. Em vez de tratar todos os estados em um único processo, seria interessante dividir as transições e comportamentos em submódulos que possam ser facilmente estendidos à medida que o número de estados ou as complexidades aumentam. Isso melhora a clareza e reduz o risco de erros durante a implementação.

- **Otimização e Organização do Código VHDL:** A adoção de práticas de Clean Code, como nomeação clara e significativa para as variáveis, e maior organização no código VHDL, poderia aumentar a legibilidade e tornar o sistema mais modular. Utilizar mais comentários explicativos também seria benéfico, especialmente nas seções críticas como a lógica de controle e transições de estados. Isso permitiria que outros desenvolvedores ou até mesmo os próprios autores pudessem modificar ou estender o sistema de maneira mais eficiente.

- **Expansão das Operações Aritméticas:** A limitação de operações a apenas soma e subtração poderia ser expandida, adicionando mais operações aritméticas, como multiplicação e divisão. Para isso, seria necessário ajustar a FSM para lidar com mais operações, o que poderia ser feito de forma simples com o aumento do número de bits de controle e a implementação de blocos específicos para cada operação aritmética. Isso traria mais flexibilidade ao sistema e o tornaria mais útil em

contextos diversos.

- **Automatização da Inserção de Dados e Parâmetros:** Criar uma interface mais dinâmica e flexível para a inserção de valores no VHDL seria uma melhoria importante. Em vez de modificar arquivos `.txt` manualmente, poderia ser criada uma interface diretamente no código VHDL ou um sistema que aceitasse entradas mais dinâmicas, como comandos ou parâmetros passados via interface serial. Isso tornaria os testes mais rápidos e precisos, permitindo maior controle sobre os cenários de entrada.

- **Aprimoramento dos Testes no Testbench:** Embora os testes no testbench tenham sido adequados para a validação inicial, a cobertura poderia ser expandida. Testes de integração que verifiquem o comportamento do sistema completo, além de simulações com entradas inesperadas ou falhas de sincronização, poderiam aumentar a robustez do sistema. Também seria interessante automatizar os testes, criando um conjunto de testes unitários que verificassem as funcionalidades do sistema sempre que o código fosse alterado.

Essas sugestões de melhoria visam não só otimizar o sistema, mas também garantir que ele seja mais flexível, escalável e sustentável a longo prazo, facilitando tanto a manutenção quanto a possível expansão para outros projetos.

---

5. Conclusão

O desenvolvimento do projeto de caixa registradora utilizando VHDL foi uma experiência significativa, que exigiu a aplicação prática de conceitos fundamentais de design digital, como a implementação de máquinas de estados finitos (FSM), o uso de VHDL para descrever comportamentos sequenciais e combinacionais, e a integração de módulos de maneira eficiente e funcional. Embora o projeto tenha sido realizado dentro de limitações de tempo e complexidade, ele cumpriu seus objetivos principais de demonstrar o controle de operações aritméticas e exibição de resultados através de componentes simples. No entanto, ao longo da execução, foi possível identificar áreas onde o sistema poderia ser expandido e otimizado, como na modularização do código, na estruturação da máquina de estados e na melhoria das práticas de clean code, o que contribuiria para uma maior escalabilidade e manutenção do projeto.

A implementação de uma arquitetura mais modular e a adoção de melhores práticas de codificação poderiam ter proporcionado um código mais claro, reutilizável e eficiente. Além disso, a ampliação da cobertura de testes e a automação da inserção de dados seriam passos importantes para garantir a robustez do sistema em cenários mais complexos. Apesar das limitações, o projeto cumpriu seu papel de aplicar conceitos de VHDL de maneira prática e orientada à solução de problemas reais, sendo um excelente ponto de partida para a realização de melhorias e expansões futuras.

6. Referências

- Perry, D. L. (2002). VHDL: Programming by Example (4th ed.). McGraw-Hill.
  Fornece exemplos práticos de como utilizar VHDL para desenvolver circuitos digitais.

- Bhasker, J. (1999). A VHDL Primer (3rd ed.). Prentice Hall.
  Aborda desde conceitos básicos até avançados de VHDL, sendo uma excelente introdução à linguagem.

- IEEE. (2008). IEEE Standard VHDL Language Reference Manual. IEEE.
  Referência oficial da linguagem VHDL, contendo detalhes sobre sintaxe e semântica.