UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
INE5406 - SISTEMAS DIGITAIS

PROJETO PRÁTICO DE SISTEMAS DIGITAIS:

UNIDADE ARITMÉTICA PARA CÁLCULO DO SENO DE UM ÂNGULO EM PONTO FLUTUANTE

Área:

Sistemas Digitais

Equipe:

Rafael Luiz Cancian
Lápis Preto Castelan
Papel Almaço Chamequinho
Caneta Esferográfica Bic

Florianópolis
Abril, 2013

Equipe:

Rafael Luiz Cancian
Lápis Preto Castelan
Papel Almaço Chamequinho
Caneta Esferográfica Bic

PROJETO PRÁTICO DE SISTEMAS DIGITAIS:

UNIDADE ARITMÉTICA PARA CÁLCULO DO SENO DE UM ÂNGULO EM PONTO FLUTUANTE

Trabalho da disciplina "INE5406 - Sistemas Digitais" apresentado ao Curso de Ciências da Computação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

Professor:Rafael Luiz Cancian, Dr. Eng.

Florianópolis
12/Abril/2013

Lista de Figuras

Sumário

1. Introdução

Este projeto prático de sistemas digitais visa desenvolver um sistema digital síncrono que realize o cálculo do seno de um ângulo (em radianos) com precisão dada. Esse sistema digital será descrito em VHDL, sintetizado, simulado e prototipado em FPGA da Altera. Para a representação de números em ponto flutuantes eerá utilizado o padrão IEEE 754 e para o cálculo do seno será utilizada a expansão em série de Taylor-Mclaurin.

No sistema digital proposto o "usuário" poderá informar a precisão mínima exigida para o cálculo do seno e também poderá informar a quantidade máxima de iterações para o cálculo, já que a expansão em série de Taylo-McLaurin é um cálculo iterativo e pode não alcançar a precisão exigida, dependendo de erros numéricos. O "usuário" pode também passar um número e solicitar o cálculo do seno, quando então esse cálculo é realizado utilizando a expansão para o seno, que é $sin(x) = \sum_{i=1}^{\infty} \frac{x^{(2i-1)}}{(2i-1)!}$, o que pode exigir o uso de operações de subtração, exponenciação, multiplicação e divisão em ponto flutuante. Quando o cálculo termina, quer seja por atender a precisão exigida ou ter atingido a quantidade máxima de iterações, o resultado é disponibilizado e sinalizado.

2. Projeto do Sistema

O projeto do sistema digital inicia com a identificação das entradas e saídas e descrição e captura do comportamento do sistema, o que é realizado nas seções seguintes.

2.1 Identificação das Entradas e Saídas

O sistema digital proposto deve ter a interface apresentada na figura 2.1. Ele possui 3 sinais principais em sua interface, todos de 32 bits por padronização:

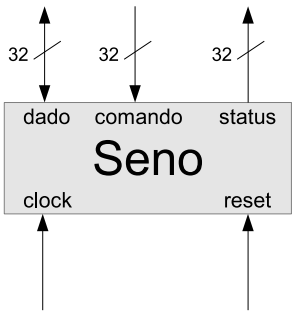

Figura 2.1: *Interface do sistema digital proposto – O sinal de comando informa a ação a adser realizada (definição da precisão, da quantidade máxima de iterações ou o cálculo do seno). O sinal bidirecional de dados permite inserir dados dos comandos ou retornar o resultado do seno, e o sinal de status informa o estado atual do sistema*
Fonte: Própria

(1) O sinal de entrada `comando`, que pemite especificar ao sistema digital o comando a ser realizado, e que pode ser:
   (a) Definir a precisão mínima exigida ($0x00000001$).
   (b) Definir a quantidade máxima de iterações ($0x00000002$).
   (c) Realizar o cálculo do seno de um ângulo ($0x00000004$).

(2) O sinal bidirecional `dado`, que pemite especificar um parâmetro de um comando ao sistema digital, e também é usado para retornar o resultado do cálculo. Esse sinal pode ser:
   (a) A precisão mínima exigida (ε), um número em ponto flutuante, quando o comando $0x00000001$ estiver sendo fornecido (dado de entrada).
   (b) A quantidade máxima de iterações (k), um número inteiro positivo, quando o comando $0x00000002$ estiver sendo fornecido (dado de entrada).
   (c) O ângulo a ter o seno calculado,um número em ponto flutuante, quando o comando $0x00000004$ estiver sendo fornecido (dado de entrada).
   (d) O seno calculado, um número em ponto flutuante, quando o comando $0x00000004$ tiver terminado de ser executado (dado de saída).

(3) O sinal de saída `status`, que pemite ao sistema digital externalizar seu estado atual, e que pode ser:
   (a) Ocioso, sem nenhum resultado disponível ($0x00000000$).
   (b) Ocupado realizando alguma operação interna ($0x00000001$).
   (c) Ocioso, com resultado disponível no sinal `dado` ($0x00000002$).
   (d) Ocioso, sem resultado disponível, devido à ocorrência de algum erro ($0x00000004$).

Além dos sinais descritos, o sistema digital proposto possui ainda um sinal de entrada `clk`, que corresponde ao sinal de relógio para sincronismo e o sinal de entrada `reset`, que corresponde a um sinal de reset assíncrono. Durante o reset são atribuídos valores *default* à precisão mínima exigida (1.10^{-12}) e à quantidade máxima de iterações (30), além de dinifir o valor do sinal de `status` como $0x00000000$.

2.2 Descrição e Captura do Comportamento

O funcionamento básico do sistema pode ser descrito como segue. Após o reset, o sistema fica num estado `ocioso` em que apenas aguarda o recebimento de um comando a ser executado, sem realizar nenhuma ação adicional. Quando um comando for detectado no sinal `comando`, o sistema executa uma das ações a seguir:

(a) Comando "Definir a precisão mínima exigida (0*x*00000001)". Nesse caso, o valor atual do sinal `dado` será interpretado como essa precisão (em ponto flutuante) e será armazenado num registrador correspondente.
(b) Comando "Definir a a quantidade máxima de iterações (0*x*00000002)".Nesse caso, o valor atual do sinal `dado` será interpretado como essa quantidade (em ponto flutuante) e será armazenado num registrador correspondente.
(c) Comando "Realizar o cálculo do seno de um ângulo (0*x*00000004)". Nesse caso, o valor atual do sinal `dado` será interpretado como esse ângulo (em ponto flutuante), será armazenado num registrador correspondente, e o cálculo do seno iniciará.

Após a execução das ações citadas, o sistema volta ao estado `ocioso`. É de responsabilidade da entidade que acionou o sinal `comando`, desativá-lo antes que o sistema volte ao estado `ocioso`, ou ele será reexecutado. Por sua complexidade, o funcionamento do cálculo do seno é descrito mais detalhadamente a seguir.

A realização do cálculo do seno corresponde à execução da equação da expansão em série de Taylor-McLaurin para o seno, dada por $sin(x) = \sum_{i=1}^{\infty} \frac{x^{(2i-1)}}{(2i-1)!}$, e que pode ainda ser reescrita como

$sin(x) = \sum_{\substack{i=1 \\ (i+1)/2 \in \mathbb{N}}}^{\infty} \frac{x^i}{i!}$, ou seja i é impar.

Essa soma não pode ser realizada até o infinito, então precisa ser truncada em algum termo, o que será feito com base na precisão mínima exigida (ε) ou na quantidade máxima de iterações (k), de modo que a equação do seno pode ser reescrita como segue: $sin(x) \approx \sum_{\substack{i=1 \\ (i+1)/2 \in \mathbb{N}}}^{n} \frac{x^i}{i!}$, $\mid n = k \vee \varepsilon < \frac{x^{(n+1)}}{(n+1)!}$

, ou seja, soma de $i = 1$ para todos os i ímpares até n, tal que n tenha atingido a quantidade máxima de iteração k ou então que a precisão ε tenha sido alcançada.

Essa equação pode então ser executada pelo algoritmo computacional 2.1.

```
1  double potencia(double base, unsigned int expoente) {
2      double pot=1;
3      for (int i=1; i<=expoente; i++) {
4          pot*=i;
5      }
6      return pot;
7  }
8
9  unsigned int fatorial(unsigned int n) {
10     fat=1;
11     for(int i=1; i<=n; i++) {
12         fat *= i;
13     }
14     return fat;
15 }
16
17 double seno(double x, double epsilon, unsigned int k) {
18     double soma=0;
19     double termo;
20     unsigned int i=1;
21     do {
22         termo = potencia(x,i) / fatorial(i);
23         soma += termo;
24         i += 2;
25     } while ((i<k) && (termo>epsilon));
26     return soma;
27 }
```

Algoritmo 2.1: *Versão inicial do cálculo do seno de um ângulo seguindo a expansão em série de Taylor-McLaurin*

Contudo, uma breve análise desse algoritmo é suficiente para verificar que, nessa forma, ele é altamente ineficiente e custoso, já que à medida quenovos termos vão sendo calculados, muitas operações já realizadas são repetidas, tanto no cálculo do expoente quanto no cálculo do fatorial. Com uma pequena análise nesse algoritmo é possível reescrevê-lo de forma a evitar tantos recálculos e torná-lo mais eficiente. A segunda versão do algoritmo para o cálculo do seno de um ângulo é apresentada no algortimo 2.2.

```
1  double seno(double x, double epsilon, unsigned int k) {
2      double soma=x;
3      double potencia=x;
4      unsigned int fatorial=1;
5      unsigned int i=3;
6      double termo;
7      do {
8          potencia *= x*x;
9          fatorial *= (i-1)*i;
10         termo = potencia / fatorial;
11         soma += termo;
12         i += 2;
13     } while ((i<k) && (termo>epsilon));
14     return soma;
15 }
```

Algoritmo 2.2: *Segunda versão do cálculo do seno de um ângulo seguindo a expansão em série de Taylor-McLaurin*

Assim, o funcionamento completo do sistema digital proposto pode ser descrito pelo algoritmo 2.3. Nesse algoritmo, podemos verificar os seguintes comportamentos principais: Os valores `default` para a precisão e a quantidade de iteração são definidas na iniciação das variáveis (reset – linhas 2 e 3); O sistema aguarda enquanto nenhum comando é fornecido (linha 5); Quando um comando é dado, o `status` do sistema passa a ser ocupado (linha 6) e, dependendo do comando, um comportamento diferente é executado. O cálculo do seno propriamento dito é realizado nas linhas 17 a 30, e o sistema sinaliza que o resultado está pronto na linha 31. A variável `x` foi substituída por `*dado` nas linhas 18 e 19 apenas para que elas não tenham uma dependência de dados com `x`.

```
1  void sistema_digital_seno(unsigned int comando, double* dado, unsigned int* status) {
2     double x, epsilon=1e-12;                    // inicialização durante o reset
3     unsigned int k=30;
4     while(true) {
5         while(comando==0) ;                     // fica esperando um comando
6         *status = 0x00000001;                   // sistema está ocupado
7         switch (comando) {
8             case 0x00000001:                    // define precisão mínima
9                 epsilon = *dado;                // salva a precisão fornecida
10                *status = 0x00000000;           // sistema volta a estar ocioso
11                break;
12            case 0x00000002:                    // define qtd máxima de iterações
13                k = *dado;                      // salva qtd máxima
14                *status = 0x00000000;           // sistema volta a estar ocioso
15                break;
16            case 0x00000004:                    // solicita cálculo do seno
17                x = *dado;                      // salva ângulo
18                double soma=*dado;              // inicializa com primeiro termo
19                double potencia=*dado;          // faz x^1
20                unsigned int fatorial=1;        // calcula 1!
21                unsigned int i=3;               // começa do 3o termo
22                double termo;
23                 do {
24                    potencia *= x*x;            // acumula x^i
25                    fatorial *= (i-1)*i;        // acumula i!
26                    termo = potencia / fatorial;// calcula termo da série
27                    soma += termo;              // acumula série
28                    i += 2;
29                } while ((i<k) && (termo>epsilon));
30                *dado = soma;                   // disponibiliza resultado
31                *status = 0x00000002;           // sistema ocioso com resultado
32                break;
33        }
34    }
35 }
```

Algoritmo 2.3: *Descrição completa do funcionamento do sistema digital para cálculo do seno*

Visando obter plena confiança no algoritmo que descreve o comportamento do sistema, o mesmo foi implementado em linguagem de programação C (após pequena adaptação para eliminar o laço infinito que representa o funcionamento contínuo do sistema) e um conjunto de testes foi realizado sobre o mesmo. Basicamente, o programa de teste que foi usado é apresentado no algoritmo 2.4.

```
1  void main(void) {
2      unsigned int comando = 0x00000004;
3      double* dado;
4      unsigned int* status;
5      unsigned int i;
6      for (i=0; i<10; i++) {
7          *dado = i*3.141592/10.0;
8          printf("O seno de %f é %f",*dado,sin(*dado));
9          sistema_digital_seno(comando, dado, status);
10         printf(" e o valor calculado foi %f\n",*dado);
11     }
12 }
```

Algoritmo 2.4: *Programa de teste do algoritmo que descreve o comportamento do sistema digital para cálculo do seno*

A saída desse programa de teste mostra que o valor do seno calculado pelo algoritmo é igual ao valor calculado pela biblioteca `math` da linguagem. Portanto, o algoritmo funciona corretamente e pode ser usado como descrição do sistema digital proposto.

Analisando o algoritmo 2.3 podemos verificar se há a necessidade da existência de registradores de dados para armazenar variáveis internas, podemos extrair as operações aritméticas, lógicas, relacionais e de transferência que são necessárias, bem como o fluxo de controle de execução desse comportamento e ainda dependências entre dados que nos permitem identificar as operações que precisam ser sequenciadas e as operações que podem ser realizadas em paralelo.

Portanto, com base nesse algoritmo, podemos identificar os elementos abaixo, que farão parte do projeto do bloco de controle e do bloco operativo desse sistema digital.

I **Registradores de dados**. Os registradores necessários podem ser extraídos observando as variáveis internas. Nesse sistema digital, são necessários pelo menos os seguintes registradores:

(1) **x**: 32 bits, ponto flutuante.
(2) **epsilon**: 32 bits, ponto flutuante.
(3) **k**: 32 bits, inteiro sem sinal.
(4) **soma**: 32 bits, ponto flutuante.
(5) **potencia**: 32 bits, ponto flutuante.
(6) **fatorial**: 32 bits, ponto flutuante.
(7) **termo**: 32 bits, ponto flutuante.
(8) **i**: 32 bits, inteiro sem sinal.

Os registradores de dados identificadores serão alocados ao bloco operativo. Ressalta-se que o sinal de saída `status` também corresponde a uma variável interna e recebe atribuições de valores. Contudo, ele não foi listado como registrador de dados pois a variável `status` no algoritmo demonstra se comportar como uma variável de controle, e não de dados. Assim, deve ser associado ao bloco de controle, e não ser asoociada a um registrador de dados no bloco operativo.

II **Operações**. As operações aritméticas, lógicas, relacionais e de transferência podem ser extraídas observando diretamente as operações realizadas pelo algoritmo e também seus operandos. Nesse sistema digital são necessárias as seguintes operações:

(1) **Comparação com zero**: Necessária para aexecutar a funcionalidade da linha 5.
(2) **Atribuição** (transferência de dados): Todos os registradores terão valores atribuídos em certos momentos (linhas 2, 3, 9,, 13, 18, 19, 20, 22, 24, 25, 26, 27), o que significa que a carga desses registradores precisa ser controlada.
(3) **Multiplicação em ponto flutuante**: Necessária para executar a funcionalidade da linha 24.
(4) **Subtração inteira sem sinal**: Necessária para executar a funcionalidade da linha 25.
(5) **Multiplicação inteira sem sinal**: Necessária para executar a funcionalidade da linha 25.
(6) **Divisão em ponto flutuante**: Necessária para executar a funcionalidade da linha 26. Nessa mesma linha, implicitamente há mais uma operação a ser realizada, que á conversão de `fatorial` do tipo `unsgned int` para o tipo `double`.
(7) **Conversão Inteiro para Ponto Flutuante**: Necessária para a conversão implícita da linha 26, ou pode ser eliminada se tanto `fatorial` quanto `i` forem variáveis em ponto flutuante, caso em que a `Subtração inteira sem sinal` não será mais necessária, mas torna-se necessária a operação `Subtração em ponto flutuante`.

(8) **Adição em ponto flutuante**: Necessária para executar a funcionalidade da linha 27.
(9) **Adição inteira**: Necessária para executar a funcionalidade da linha 28, ou desnecessária caso `i` seja uma variável em ponto flutuante.
(10) **Comparação de menor entre números inteiros sem sinal**: Necessária para executar a funcionalidade da linha 29.
(11) **Comparação de maior entre números em ponto flutuante**: Necessária para executar a funcionalidade da linha 29.

As operações listadas, após as devidas decisões de projeto sobre compartilhamento ou não de recusos, serão associadas a elementos de hardware no bloco operativo.

III **Fluxo de Controle**. O fluxo de controle pode ser extraído a partir das próprias estruturas de controle de fluxo do algoritmo (como `if-the-else`, `switch-case`, `for`, `do-while`, sub-rotinas, etc), além do controle de início de término de algumas operações que podem levar de um pulso de clock, como `Divisão em ponto flutuante`, por exemplo. Assim, o fluxo de controle principal do sistema consiste em:

(1) **Inicialização variáveis**: Necessária em praticamente todos os sistemas digitais e explícita nas linhas 2 e 3, e poderá corresponder ao controle do estado de reset.
(2) **Espera pelo comando**: Necessário para o controle da linha 5, em que o sistema fica "parado" até que um comando tenha sido enviado.
(3) **Desvio para comportamento do comando**: Necessário para o controle da linha 7, em que o fluxo de execução é desviado dependendo do comando recebido.
(4) **Controle de final de laço**: Necessário para o controle da linha 29, em que o fluxo de execução pode voltar para a linha 23 ou continuar, conforme a condição especificada.
(5) **Desvios incondicionais**: Necessários para os controles das linhas 11, 15 e 32, que retornam o fluxo de execução para a linha 5 assim que o respectivo comando terminou.

O fluxo de controle é função do bloco de controle, que pode ser implementado como uma FSM ou outra alternativa (microcódigo, ROM, etc). Ressalta-se que o fluxo de controle pode requerer sinais de status provindos do bloco operativo que forneçam informações sobre a "direção" desse fluxo.

IV **Dependência entre Dados**. A dependência entre dados exige uma análise mais cuidadosa do algoritmo. Em geral, a natureza sequencial de um algoritmo já ajuda a identificar quais dados dependem de outros, devido à ordem de execução dos comandos. Entretanto, alguns deles podem ser paralelizados pois não há dependência real entre eles. Então, assumimos que as operações devem ser realizadas sequencialmente conforme especificado no algoritmo, com exceção das operações seguintes, que foram identificadas como não possuindo dependência de dados entre si.

(1) **Linhas 2 e 3**: A inicialização das variáveis `epsilon` e `k` não possui dependências.
(2) **Linhas 9 e 10**: A atribuição de valores para `epsilon` e `status` não possui dependências.
(3) **Linhas 13 e 14**: A atribuição de valores para `k` e `status` não possui dependências.
(4) **Linhas 17, 18, 19, 20 e 21**: A atribuição de valores para todas essas variáves não possui dependências, já que `x` no algoritmo 2.2 foi substiuída por `*dado` na versão final no algoritmo 2.3.
(5) **Linhas 24 e 25**: O cálculo e atribuição valores para `potencia` e `fatorial` não possuem dependências.
(6) **Linhas 27 e 28**: O cálculo e atribuição valores para `soma` e `i` não possuem dependências.

Com todos esses elementos extraídos diretamente a partir do algoritmo que descreve o comportamento do sistema digital, podemos capturar o comportamento do sistema digital em uma máquina de estados de alto nível (FSMD). A figura 2.2 apresenta a FSMD do sistema digital proposto.

2.3 Projeto do Bloco Operativo

A partir dos registradores de dados e das operações identificados com o algoritmo do comportamento do sistema, iniciamos o projeto do bloco operativo. Diferentes decisões de projeto podem levar a uma implementação monociclo, multiciclo ou com pipeline, po exemplo. Uma primeira etapa pode ser isolar todas as opções de atribuição a cada registrador de dados identificado e derivar o circuito associado.

Nesse caso, abaixo são apresentados os circuitos para atribuição de cada registrador identificado com o

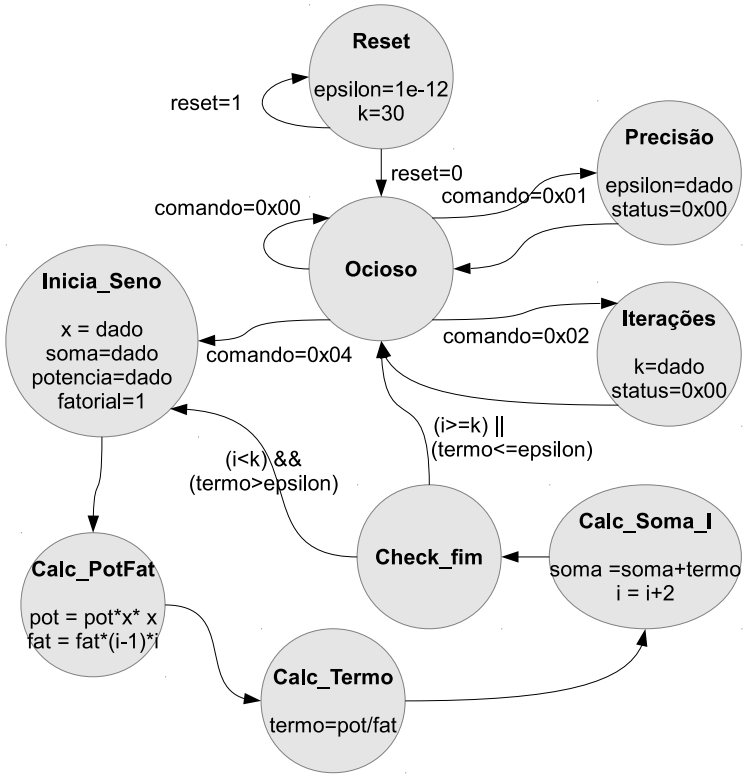

Figura 2.2: *FSMD do sistema digital – O comportamento do sistema digital, previamente representado em algoritmo, foi capturado numa FSM de alto nível, que já considera os registradores, operações, fluxo de controle e dependências identificados. Fonte: Própria*

algoritmo. Nas figuras apresentadas, os sinais de controle são representados em azul e os sinais de estado (retorno ao bloco de controle) são apresentados em vermelho.

(1) **x = dado;** (linha 17). Figura 2.3

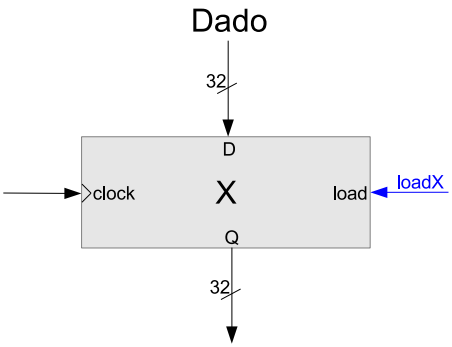

Figura 2.3: *Circuito para atribuição do registardor x*

(2) **epsilon = dado;** (linha 9). Figura 2.4

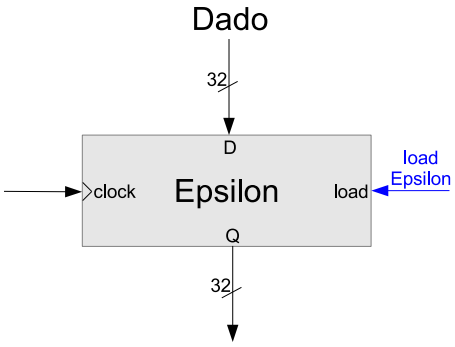

Figura 2.4: *Circuito para atribuição do registardor* `epsilon`

(3) **k = dado;** (linha 13). Figura 2.5

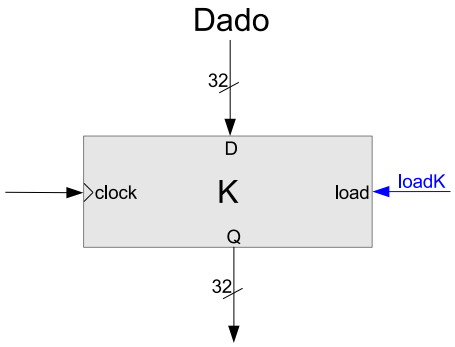

Figura 2.5: *Circuito para atribuição do registardor* `k`

(4) **soma = dado;** (linha 18) ou **soma = soma + termo;** (linha 27). Figura 2.6

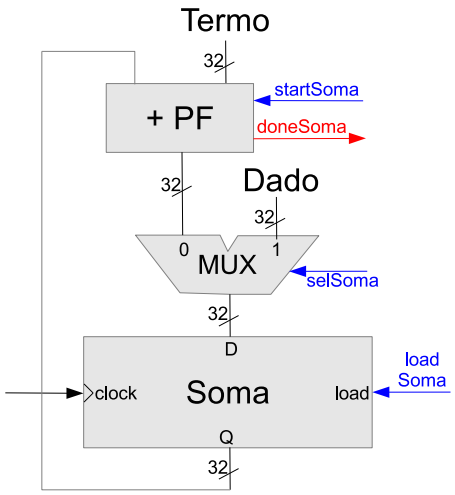

Figura 2.6: *Circuito para atribuição do registardor* `soma`

(5) **potencia = dado;** (linha 19) ou **potencia = potencia * x * x;** (linha 24). Figura 2.7

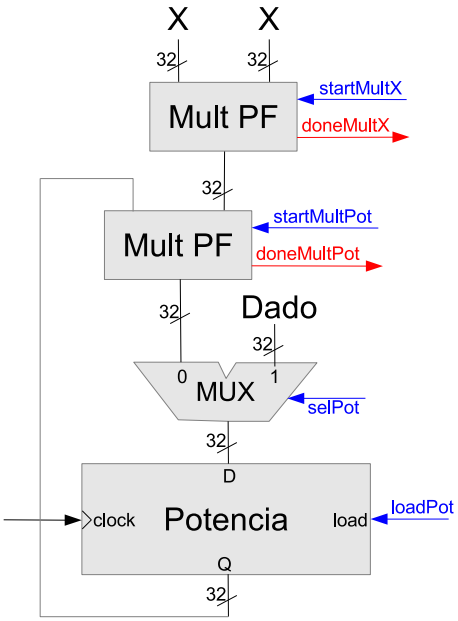

Figura 2.7: *Circuito para atribuição do registardor* `potencia`

(6) **fatorial = 1;** (linha 20) ou **fatorial = fatorial * (i-1) * i;** (linha 25). Figura 2.8

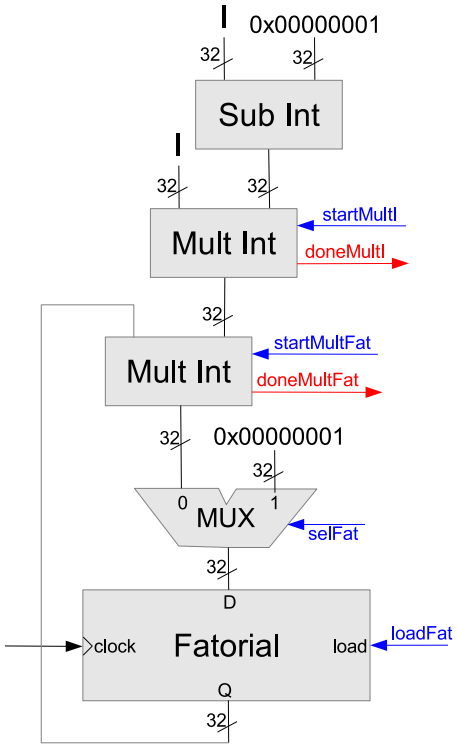

Figura 2.8: *Circuito para atribuição do registardor* `fatorial`

(7) **termo = potencia / fatorial;** (linha 26). Figura 2.9

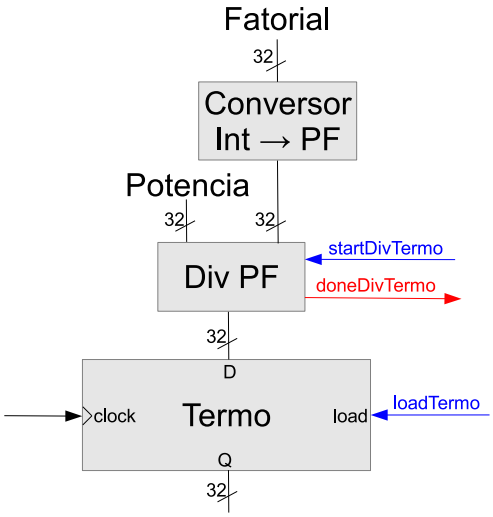

Figura 2.9: *Circuito para atribuição do registardor* `termo`

(8) **i = 3;** (linha 21) ou **i = i + 2;** (linha 28). Figura 2.10

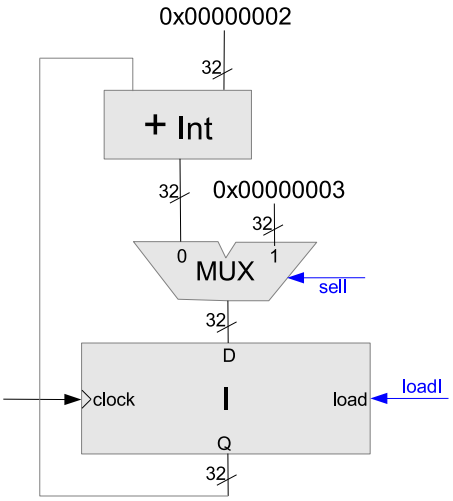

Figura 2.10: *Circuito para atribuição do registardor* `i`

Além disso precisamos dos circuitos que serão usados para o controle do fluxo de execução. Com base no fluxo de controle previamente identificado, verifica-se que o estados que serão usado em dois controles de fluxo precisam ser gerados:

(1) **comando==0** (Espera pelo comando – linha 17).
(2) **(i<k) && (termo>epsilon)** (Controle de final de laço – linha 9).

Esses circuitos precisam ser unidos para formar o bloco operativo, e é preciso decidir se alguns recursos idênticos serão replicados e poderão operar paralelamente, ou se eles corresponderão a uma única instância que precisará ser escolada em diferentes pontos do tempo para realizar as diferentes operações às quais estão associados. Se decidirmos por uma versão com menos recursos de hardware, o projeto do bloco operativo pode ser aquele apresentado na figura 2.11. Analisando essa figura, percebe-se que dois multiplicadores inteiros foram unificados e que dois multiplicadores em ponto flutuante também foram unificados. Com isso, as operações que usavam essas multiplicações (cálculo da `potencia` e do `fatorial`) precisam ser realizadas em diferentes instantes no tempo, o que será realizado pelo bloco de controle.

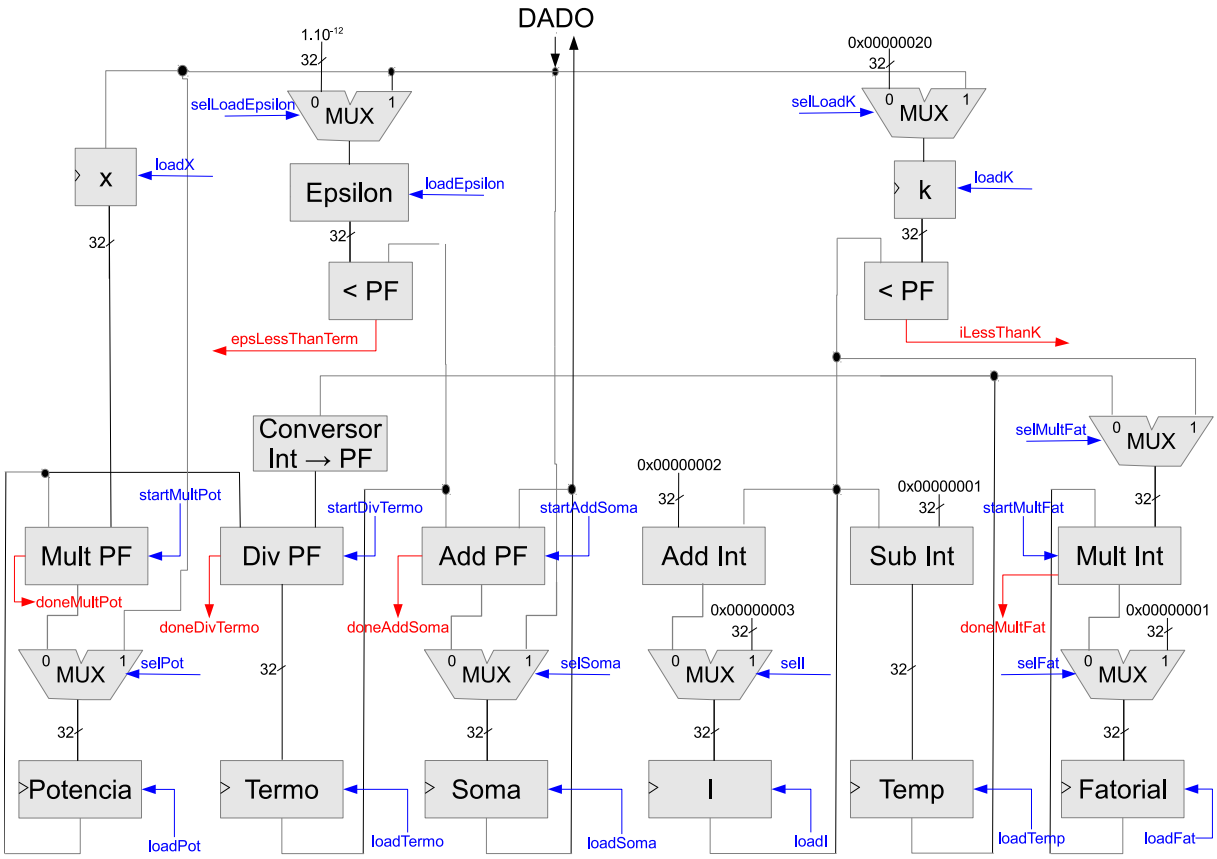

Figura 2.11: *Bloco Operativo do sistema digital proposto – Todas as operações sobre dados e também a geração de estados para o controle foram consideradas e optou-se por uma versão com recursos compartilhados (menor custo de hardware). Fonte: Própria*

Com base no projeto do bloco operativo, podemos listar os sinais de controle (vindos do futuro bloco de controle) e os sinais de status (que serão enviados ao bloco de controle). São sinais de controle:

(1) `loadX`: Controla a carga do registrador `X` (0:Mantém; 1:Carrega).
(2) `loadEpsilon`: Controla a carga do registrador `Epsilon` (0:Mantém; 1:Carrega).
(3) `loadK`: Controla a carga do registrador `K` (0:Mantém; 1:Carrega).
(4) `loadPot`: Controla a carga do registrador `Potencia` (0:Mantém; 1:Carrega).
(5) `loadTermo`: Controla a carga do registrador `Termo` (0:Mantém; 1:Carrega).
(6) `loadSoma`: Controla a carga do registrador `Soma` (0:Mantém; 1:Carrega).
(7) `loadI`: Controla a carga do registrador `I` (0:Mantém; 1:Carrega).
(8) `loadTemp`: Controla a carga do registrador `Temp` (0:Mantém; 1:Carrega).
(9) `loadFat`: Controla a carga do registrador `Fatorial` (0:Mantém; 1:Carrega).
(10) `selLoadEpsilon`: Controla o dado a ser carregado em `Epsilon` (0:cte 1e-12; 1:dado).
(11) `selLoadK`: Controla o dado a ser carregado em `K` (0:cte 32; 1:dado).
(12) `selMultFat`: Conrola o dado a ser multiplicado para cálculo do `Fatorial` (0:temp(i-1); 1:i).
(13) `selPot`: Controla o dado a ser carregado em `Potencia` (0:Multiplicação; 1:dado).
(14) `selSoma`: Controla o dado a ser carregado em `Soma` (0:Soma; 1:dado).
(15) `selI`: Controla o dado a ser carregado em `I` (0:soma; 1:dado).
(16) `selFat`: Controla o dado a ser carregado em `Fatorial` (0:Multiplicação; 1:cte 1).
(17) `startMultPot`: Comando o início da multiplicação em ponto flutuante para cálculo da `Potencia`.
(18) `startDivTermo`: Comando o início da divisão em ponto flutuante para cálculo do `Termo`.
(19) `startAddSoma`: Comando o início da soma em ponto flutuante para cálculo da `Soma`.
(20) `startMultFat`: Comando o início da multiplicação inteira para cálculo do `Fatorial`.

São sinais de status:

1. epsLessThanTerm: Indica que o registrador `Epsilon` tem valor (em ponto flutuante) menor que `Termo`.
2. iLessThanK: Indica que o registrador `I` é menor que `K`.
3. doneMultPot: Indica que a multiplicação para cálculo da `Potencia` já terminou.
4. doneDivTermo: Indica que a divisão para cálculo do `Termo` já terminou.
5. doneAddSoma: Indica que a soma para cálculo da `Soma` já terminou.
6. doneMultFat: Indica que a multiplicação inteira para cálculo do `Fatrorial` já terminou.

2.4 Projeto do Bloco de Controle

Com a definição do bloco operativo, o algoritmo e/ou a FSMD que capturam o computamento do sistema precisam ser refinados para refletr essa estrutura e as decisões de projeto tomadas, bem como considerar os nomes dos sinais que foram estabelecidos. Assim, inicialmente o algoritmo foi reescrito, considerando as restrições de recursos do bloco operativo, os controles das operações que levam mais de um pulso de clock e o agrupamento das operações que não possuem dependências de dados. A versão final do algoritmo que descreve o comportamento do sistema é representada pelo algoritmo 2.5.

```
1  void sistema_digital_seno(unsigned int comando, double* dado, unsigned int* status) {
2     double x;
3     // comandos na mesma linha não possuem dependência de dados e são realizados paparelalemente
4     double epsilon=1e-12;  unsigned int k=30;
5     while(true) {
6         while(comando==0) ;
7         *status = 0x00000001;
8         switch (comando) {
9             case 0x00000001:
10                epsilon = *dado;  *status = 0x00000000;
11                break;
12            case 0x00000002:
13                k = *dado;  *status = 0x00000000;
14                break;
15            case 0x00000004:
16                x = *dado; double soma=potencia=*dado; unsigned int fatorial=1; unsigned int i=3;
17                double termo, unsigned int temp;
18                 do {
19                    potencia = potencia*x; temp = (i-1); fatorial = fatorial*i;
20                    while(!done(*PF) && !done(*Int)) ;                    // aguarda as operações
21                    potencia = potencia*x; fatorial = fatorial*temp;
22                    while(!done(*PF) && !done(*Int)) ;                    // aguarda as operações
23                    termo = potencia / fatorial;
24                    while(!done(/PF)) ;                                   // aguarda as operações
25                    soma = soma + termo; i += 2;
26                    while(!done(+PF)) ;                                   // aguarda as operações
27                } while ((i<k) && (epsilon<termo));
28                *dado = soma; *status = 0x00000002;
29                break;
30        }
31    }
32 }
```

Algoritmo 2.5: *Descrição completa do funcionamento do sistema digital para cálculo do seno*

Com base nesse novo algoritmo reescrevemos a FSMD que captura o comportamento do sistema, considerando as aletrações devidas ao projeto do bloco operativo. A versão final da FSMD é apresentrada na figura 2.12.

Por fim, projetou-se a FSM que considera o projeto do bloco operativo e os nomes dados aos sinais de controle e de estados, e descreve o funcionamento do bloco de controle, e que é apresentada na figura 2.13.

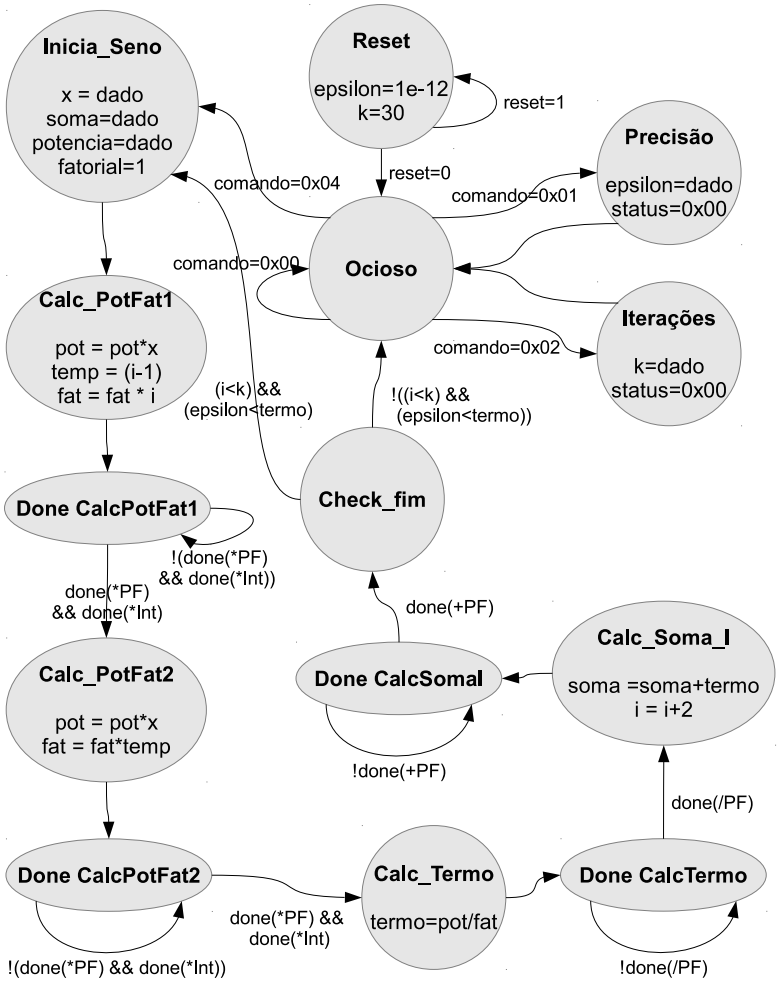

Figura 2.12: *FSMD final do sistema digital – O comportamento final do sistema digital, previamente representado em algoritmo, foi novamente capturado numa FSM de alto nível, já considerando o proejto do bloco operativo.*
Fonte: Própria

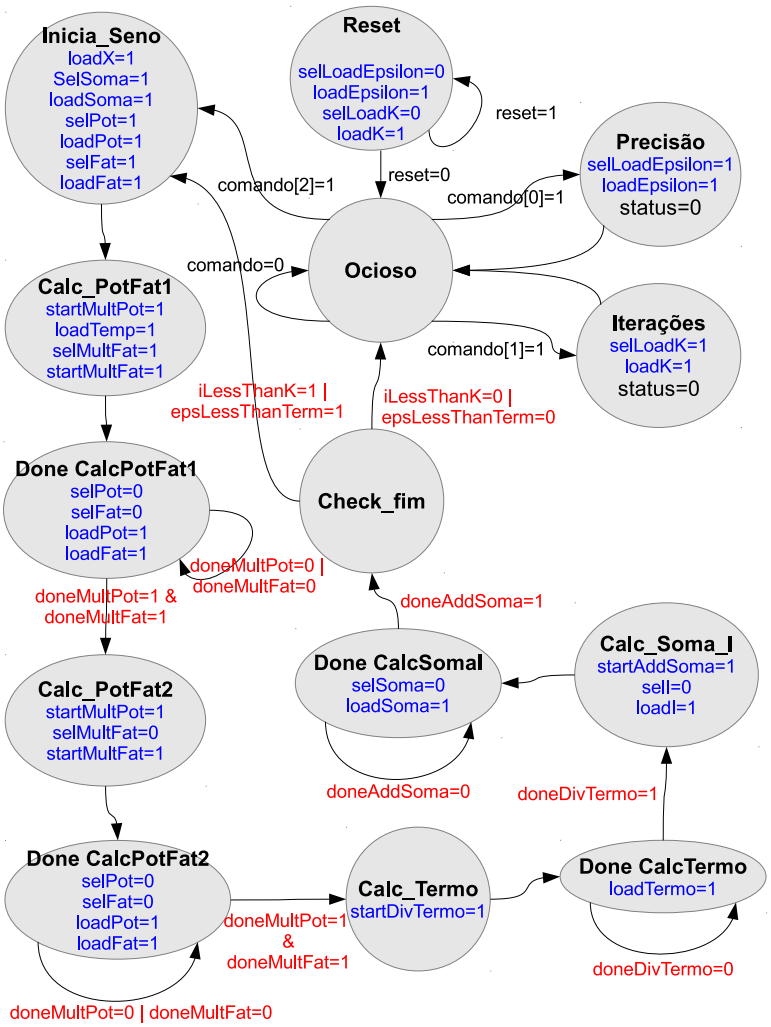

Figura 2.13: *Bloco de controle, projetado com o uma FSM – O controle do sistema é representado por esta FSM, que recebe sinais de cotnrole externos e sinais de status do bloco operativo, e gera saídas de controle e sinais de comando para o bloco operativo.*
Fonte: Própria

2.5 Projeto da Unidade Aritmética para Cálculo do Seno

Com os projetos tanto do bloco operativoquanto do bloco de controle prontos, pode-se realizar o projeto do sistema digital completo, ou seja, da Unidade Aritmética para Cálculo do Seno. Basicamente, o sistema digital será composto de um único bloco de controle e de um único bloco operativo, de modo que o projeto completo implica apenas integrar ambos os projetos anteriores. A estrutura da Unidade Aritmética para Cálculo do Seno é apresentada na figura 2.14

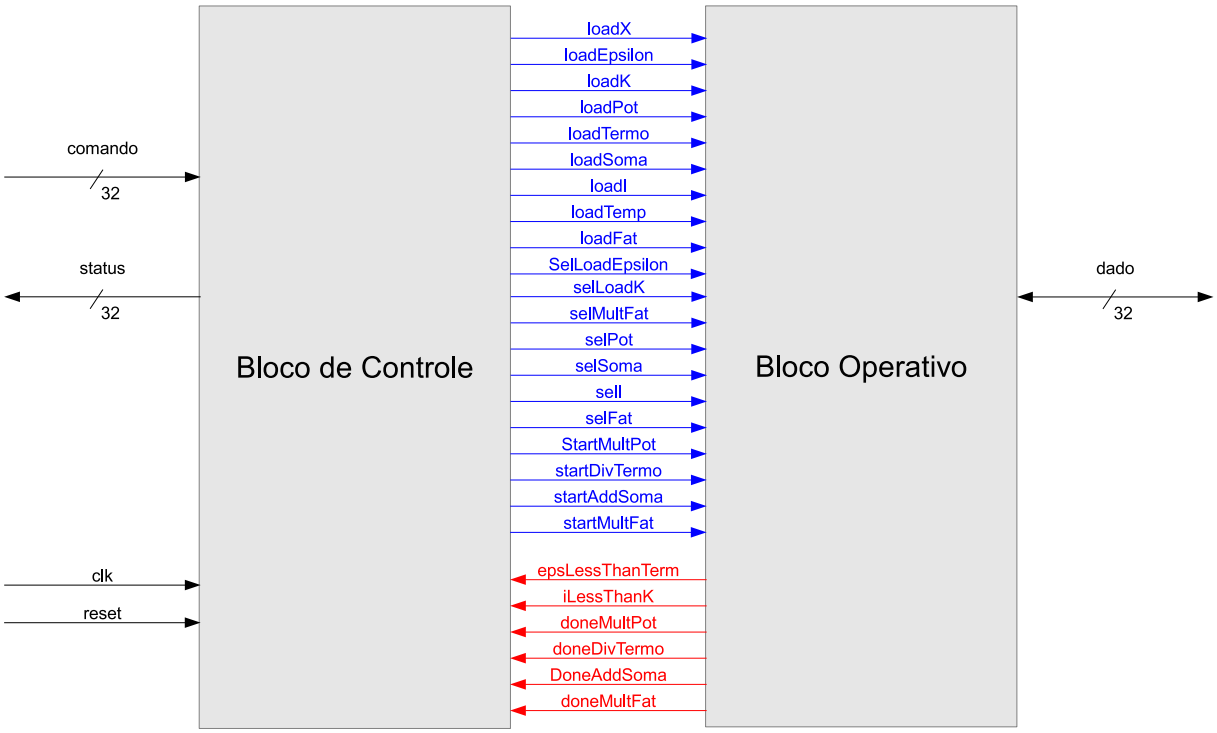

Figura 2.14: *Bloco de controle, projetado com o uma FSM – O controle do sistema é representado por esta FSM, que recebe sinais de cotnrole externos e sinais de status do bloco operativo, e gera saídas de controle e sinais de comando para o bloco operativo.*
Fonte: Própria

3. Desenvolmento

todo todo todo todo ...

3.1 Desenvolvimento do Bloco Operativo

todo todo todo todo ...

3.1.1 Multiplexador 2x1 (MUX)

todo todo todo todo ...

3.1.2 Registrador (x, epsilon, k, ...)

todo todo todo todo ...

3.1.3 Somador Inteiro (Add Int)

todo todo todo todo ...

3.1.4 Subtrator Inteiro (Sub Int)

todo todo todo todo ...

3.1.5 Multiplicador Inteiro (Mult Int)

todo todo todo todo ...

3.1.6 Somador em Ponto Flutuante (Add PF)

todo todo todo todo ...

3.1.7 Multiplicador em Ponto Flutuante (Mult PF)

todo todo todo todo ...

3.1.8 Divisor em Ponto Flutuante (Div PF)

todo todo todo todo ...

3.1.9 Comparador de Menor em Ponto Flutuante (< PF)

todo todo todo todo ...

3.2 Desenvolvimento do Bloco de Controle

todo todo todo todo ...

3.3 Desenvolvimento da Unidade Aritmética para Cálculo do Seno

todo todo todo todo ...

4. Testes e Validação

todo todo todo todo ...

4.1 Validação do Bloco Operativo

todo todo todo todo ...

4.1.1 Multiplexador 2x1 (MUX)

todo todo todo todo ...

4.1.2 Registrador (x, epsilon, k, ...)

todo todo todo todo ...

4.1.3 Somador Inteiro (Add Int)

todo todo todo todo ...

4.1.4 Subtrator Inteiro (Sub Int)

todo todo todo todo ...

4.1.5 Multiplicador Inteiro (Mult Int)

todo todo todo todo ...

4.1.6 Somador em Ponto Flutuante (Add PF)

todo todo todo todo ...

4.1.7 Multiplicador em Ponto Flutuante (Mult PF)

todo todo todo todo ...

4.1.8 Divisor em Ponto Flutuante (Div PF)

todo todo todo todo ...

4.1.9 Comparador de Menor em Ponto Flutuante (< PF)

todo todo todo todo ...

4.2 Validação do Bloco de Controle

todo todo todo todo ...

4.3 Validação da Unidade Aritmética para Cálculo do Seno

todo todo todo todo ...

5. Conclusões

todo todo todo todo ...

Referências Bibliográficas