Multiplicação Através de Somas – Modelo PC-PO

Alunos: Anderson Bottega da Silva, Juliano Felipe Prass da Silva, Maycon de Queiroz Oliveira.

# Introdução

Este trabalho teve como objetivo desenvolver um multiplicador através de somas com o modelo PC-PO (Parte Controle – Parte Operativa). O multiplicador implementado é de 8 bits utilizando somadores de 16 bits com os números em complemento de dois. Os somadores de 16 bits são CSAs (Carry Select Adders) que utilizam somadores CSAs de 8 bits internamente, que, por sua vez, utilizam somadores CLAs (Carry Look-Ahead Adders) de 4 bits.

## Modelo PC-PO (Parte Controle – Parte Operativa)

A divisão de um sistema digital pode ser descrita em duas partes e são a cooperação entre dois blocos:

Parte Operativa: É todo o caminho dos dados que estão na CPU através do barramento, oriundos dos registradores e posteriormente de volta aos próprios registradores.

Parte Controle: Ativa a sequência de operações na Parte Operativa.

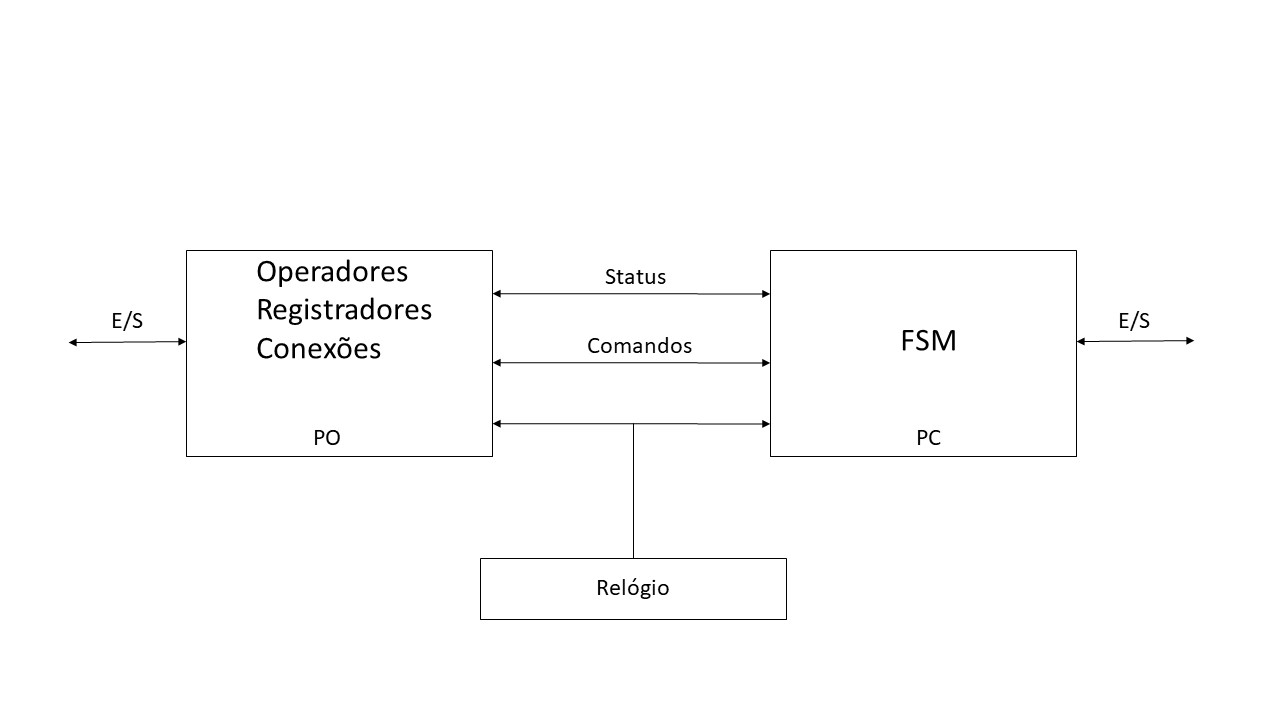


Figura 1.1: Modelo PC-PO

# Desenvolvimento

Considerando que os sinais de entrada são 8 bits de tamanho, o resultado será de até 16 bits para multiplicações. Assim, fez-se um “expansor” para que as entradas de 8 bits se tornem de 16 bits. Para o caso sinalizado, multiplica-se o bit mais significativo, seja este ‘1’ ou ‘0’. A figura 2.1 mostra o componente para realizar a expansão de entradas em complemento de dois.

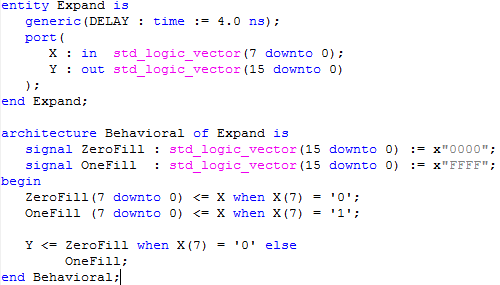


Figure 2.1 Componente para transformar de 8 para 16 bits em complemento de dois.

Depois de expandidos os sinais de entrada, deve-se atribuir os sinais “Multiplicador” e “Multiplicando”, como otimização no número de ciclos necessários para computar a multiplicação, escolhe-se o a entrada mínima absoluta como multiplicador (Se entradas são “-2” e “-3”, o componente retorna “2”) e a máxima como multiplicando. As figuras 2.2 e 2.3 mostram os componentes para a seleção do mínimo absoluto e o máximo, respectivamente.

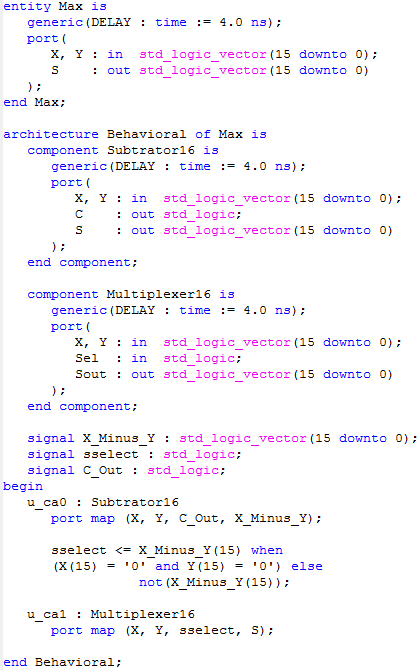
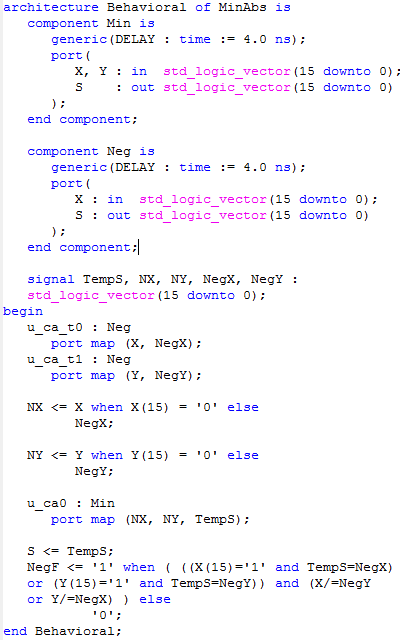


Figure 2.2 Componente mínimo absoluto. As entradas e saídas são iguais ao Max, exceto por uma flag de saída quando o multiplicador é negativo.

Figure . Componente máximo.

O componente “Subtrator16” usado em ambos os casos é uma especialização do somador CSA16 (Mostrado mais detalhadamente mais a frente), onde a entrada “Y” é passada pela porta lógica “Not” e o Carry In (Do CSA16) é forçado em “1”, isto é, soma de “X” com o complemento de dois da segunda entrada. O componente “Neg” funciona analogamente, mas força “Y” como “0” na entrada do CSA16, ou seja, faz a negação do número (e.g. 2 para -2). O “Multiplexer16” é um Multiplexador de 16 bits, onde o seletor em zero seleciona “X” e em um, “Y”.

Depois de selecionados os fatores da multiplicação, utilizam-se dois “Multiplexer16” para fazer a inicialização dos dados, um para iniciar o acumulador do resultado em zero e o outro para carregar o valor do multiplicador no acumulador do multiplicador. A inicialização é feita com o sinal “init” ativado, caso este esteja desativado (Depois do segundo pulso de clock), o multiplexador carrega o resultado da operação anterior no acumulador.

Com o auxílio de dois registradores, os dados dos sinais acumuladores são carregados em variáveis de entrada dos somadores. O primeiro somador soma o que foi acumulado até então com o multiplicando. O segundo, decrementa o multiplicador em um (Soma com uma constante igual á -1). Os resultados podem então ser mapeados para os sinais acumuladores (nos multiplexadores) e carregados nos registradores. Por fim, um sinal extra é mantido como a negação do multiplicando acumulado (Será repassado para a saída caso o multiplicador seja negativo).

O Registrador (de 16 bits) apresenta uma porta para carregar algum dado e entrada de clock. Após 8 ns da descida do Clock, a entrada é mapeada para a saída. A figura 2.4 mostra o componente para 1 bit (São acoplados 16 para 16 bits).

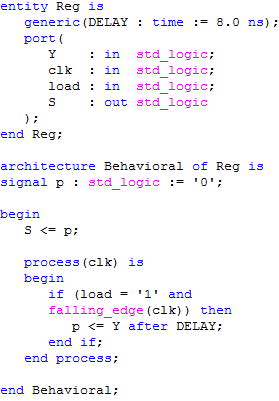


Figure 2.4 Componente do registrador de 1 bit.

As figuras 2.5 e 2.6 mostram os sinais internos e o mapeamento dos componentes internos da arquitetura, respectivamente.

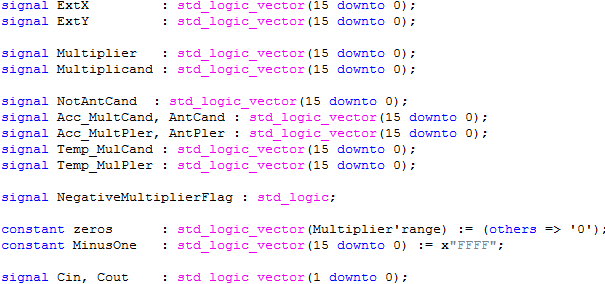


Figure 2.5 Sinais internos do Multiplicador.

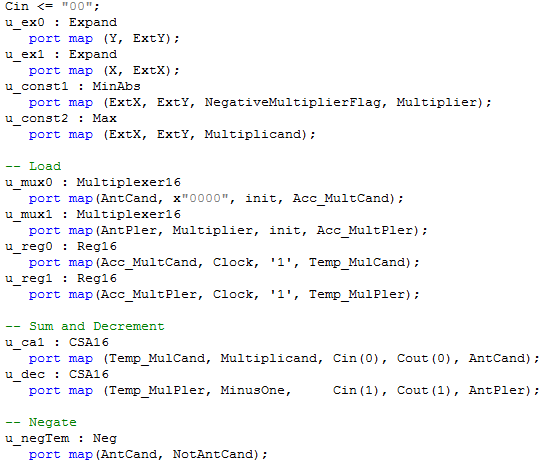


Figure 2.6 Mapeamento dos componentes internos do Multiplicador.

## Especificação do somador CSA16

Na parte mais alta, o CSA16 utiliza três CSAs de 8 bits. Os 8 bits menos significativos são passados para um CSA8 com o Carry In do primeiro componente. Os bits restantes são duplicados para os outros dois CSAs, um com Carry In forçado em zero, outro em 1. O Carry out do primeiro somador de 8 bits é usado como seletor em um Multiplexador de 8 bits. A soma, consequentemente, converge para a porta de saída depois do tempo de um somador CSA8 (As partes são calculadas em paralelo) mais o tempo de seleção do Multiplexador (4 ns). A figura 2.7 mostra esta parte do somador.

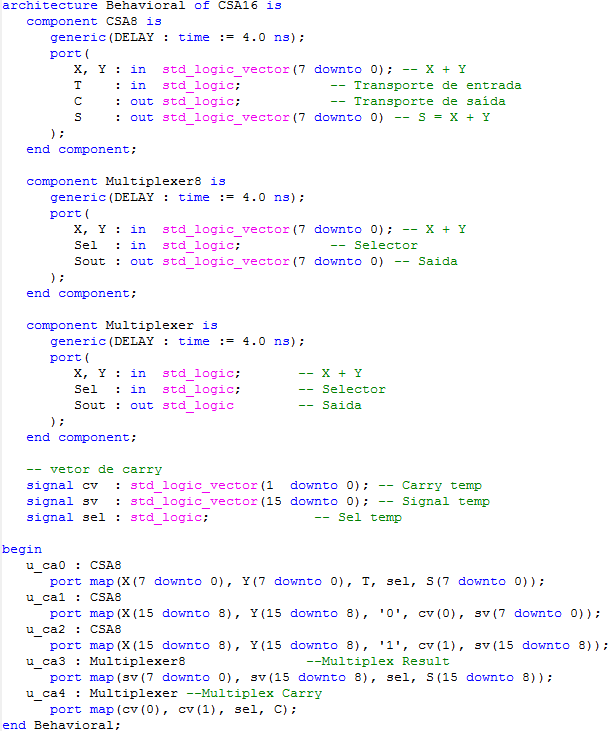


Figure 2.7 Arquitetura do componente CSA16.

O somador CSA8 funciona de forma análoga, mas ao invés de utilizar “CSA4” bits internamente, utiliza CLAs de 4 bits. A figura 2.8 ilustra a arquitetura do primeiro.

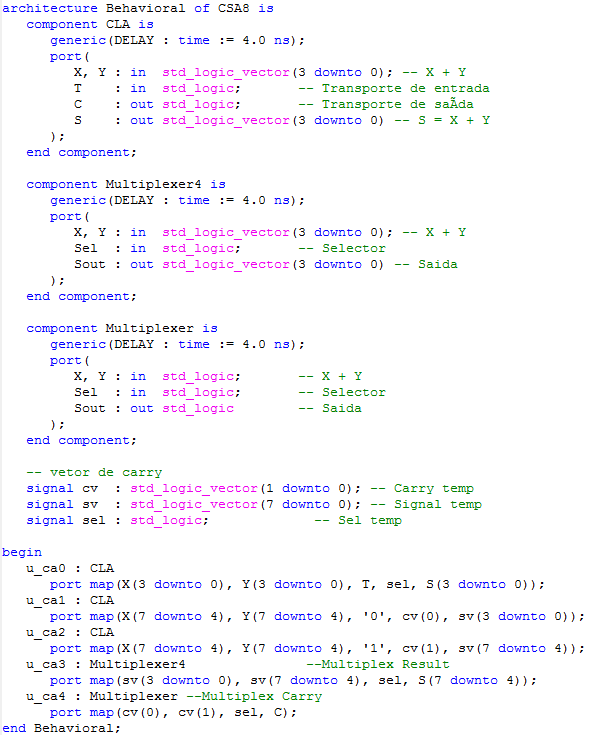


Figure 2.8 Arquitetura do componente CSA8.

O somador de 4 bits usa somadores completos simples (Que utiliza um meio somador internamente) e fórmulas para determinar antecipadamente o Carry In dos somadores completos intermediários em tempo de estabilização O(log2 n). As fórmulas booleanas utilizadas são conseguidas a partir da expansão das expressões booleanas de um somador completo. São elas:

Si = Xi xor Yi xor Ci, com Ci como Carry In; -> Soma i

Cout i = (Xi . Yi) + (Ci . (Xi + Yi)); -> Carry Out de i ou Carry In de i+1

Para efeitos de simplificação, considera-se:

Gi = (Xi . Yi) -> Gerador de Carry

Pi = (Xi + Yi) -> Propagador de Carry

Assim, pode-se reescrever a expressão para o Carry In i+1 como:

C i+1 = Gi + (Pi . Ci)

Dessa maneira, podemos explicitar as fórmulas para os “Carry In”s dos bits de 1 à 3 (Carry In 0 é dado como entrada do componente):

C1 = G0 + (P0 . C0)

C2 = G1 + (P1 . C1)

= G1 + (P1 . (G0 + P0 . C0))

= G1 + (P1 . G0) + (P1 . P0 . C0)

C3 = G2 + (P2 . G1) + (P2 . P1 . G0) + (P2 . P1 . P0 . C0)

Nota-se que expandir a fórmula para mais bits (e.g. 8), torna a estratégia menos atrativa, devido ao número de portas lógicas necessárias a partir da expansão de C3. Por esse motivo, utilizou-se a estratégia do Carry Select Adder para as outras etapas. A figura 2.9 mostra a arquitetura em VHDL do CLA.

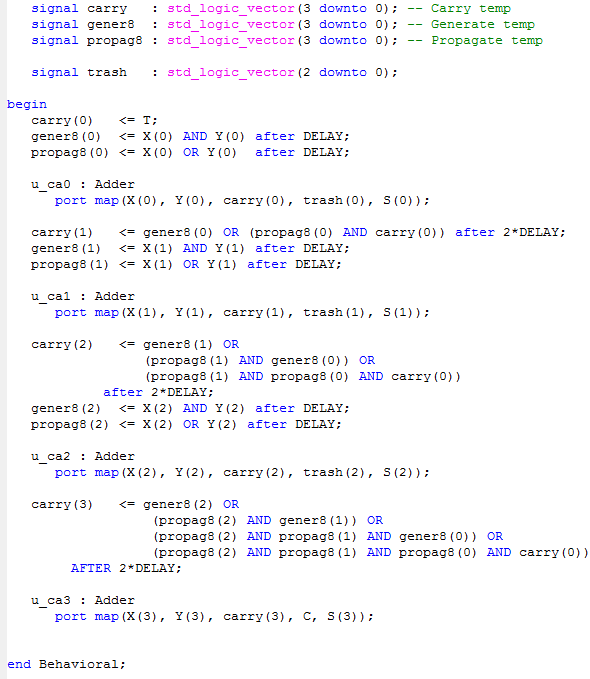


Figure 2.9 CLA de 4 bits.

## Controle do sinal de saída e TestBench utilizado

Após 1 ciclo de inicialização (Entrada “init” no componente “MultiplicadorSomador8”), são necessários “m” ciclos para o resultado convergir, onde “m” é o multiplicador, ou seja, ao multiplicar os números “5” e “-2”, levam-se 1+2 (Multiplicador é 2, já que o menor número absoluto é “-2“) ciclos para ter-se o resultado na saída. Caso multiplique-se por 0, o tempo é de apenas um ciclo. Devido ao sinal auxiliar “NotAntCand” (Figura 2.5), o tempo não muda para multiplicadores negativos.

O controle da saída é feito com um processo sensível ao sinal de clock. A figura 2.10 mostra esse trecho do código VHDL. A figura 2.11 mostra um exemplo de testbench para testar o Multiplicador.

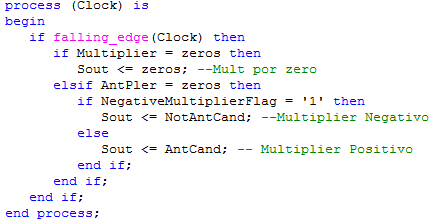


Figure 2.10 Controle de saída de resultados

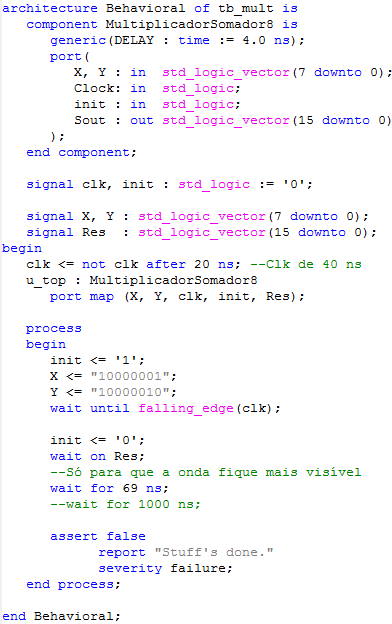


Figure 2.11 Exemplo de TestBench para o MultiplicadorSomador8.

## Diferenciação na multiplicação de não sinalizados

A expansão dos não sinalizados (Componente U\_Expand no projeto) muda no sentido que não há o sinal “OneFill”, isto é, a saída Y sempre recebe “ZeroFill”, que tem os 8 bits menos significativos mudados para os da entrada.

## Período de clock e tempo dos componentes

A expansão dos não sinalizados (Componente U\_Expand no projeto) muda no sentido que não há o sinal “OneFill”, isto é, a saída Y sempre recebe “ZeroFill”, que tem os 8 bits menos significativos mudados para os da entrada.

# Funcionamento

O multiplicador de 8 bits recebe duas entradas e escolhe o menor valor absoluto entre elas, então considera este valor como n, o outro valor é somado n vezes. Se o valor de n for negativo, o resultado é invertido. O resultado é armazenado em 16 bits

# Conclusão