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.

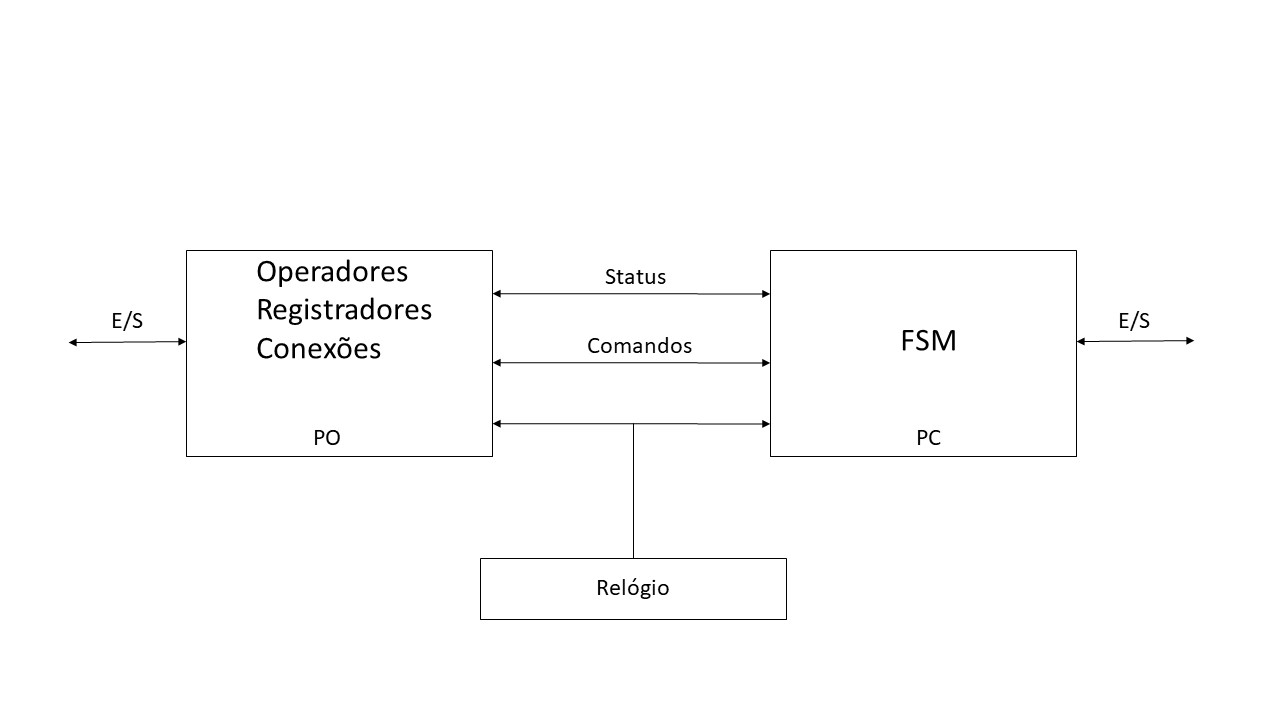


Figura 1.1: Modelo PC-PO

# 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 (*MSB, most significant bit*), seja este ‘1’ ou ‘0’. A figura 2.1 mostra o componente para realizar a expansão de entradas em complemento de dois.

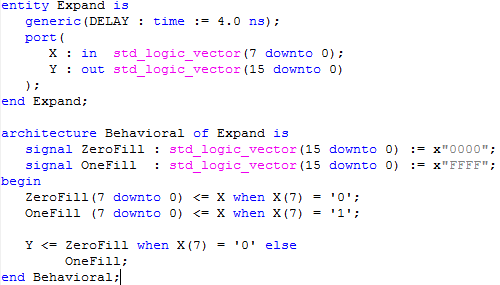


Figure 2.1 Componente para transformar de 8 para 16 bits em complemento de dois.

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 (A máxima absoluta, mas com a saída mantendo o sinal; ex.: entre 2 e -3, componente retorna -3).

Dentro do “MinAbs”, faz-se o módulo das duas entradas (Se o *MSB* de alguma entrada for “1”, atribui-se o valor negado). Depois, mapeia-se as entradas, em módulo, para o componente mínimo. Este último seleciona o mínimo entre dois valores não sinalizados através da subtração do primeiro pelo segundo, caso o resultado negativo, o primeiro é o mínimo, caso contrário, positivo. Retorna-se o valor em módulo e uma flag que indica se o valor retornado era negativo ou não.

O “Máx” utiliza o componente anterior como auxílio para a escolha da entrada. O máximo faz essencialmente o oposto do “MinAbs”, isto é, retorna o máximo em módulo. A diferença é que o primeiro retorna os valores ainda em complemento de dois, não sendo necessário posterior conversão.

O mínimo absoluto é utilizado como multiplicador (Para reduzir a quantia necessária de somas para o resultado final), e o máximo para o multiplicando. As figuras 2.2 e 2.3 mostram os componentes para a seleção do mínimo absoluto e o máximo, respectivamente.

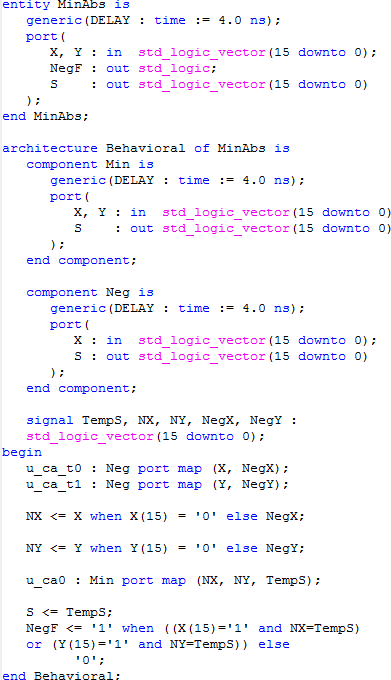
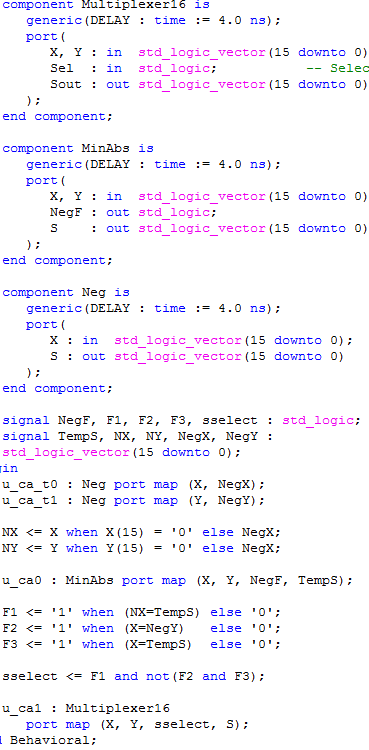


Figure 2.2 Componente mínimo absoluto. As entradas e saídas são iguais ao Max, exceto por uma flag de saída quando o multiplicador é negativo.

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).

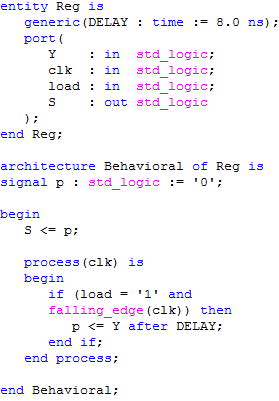


Figure 2.4 Componente do registrador de 1 bit.

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

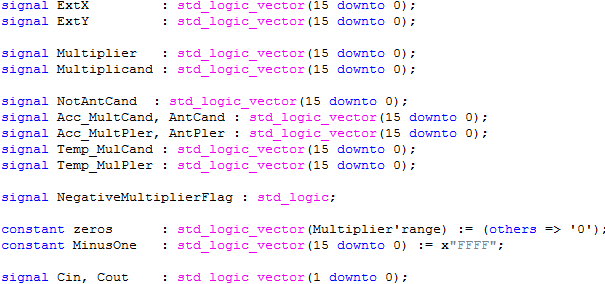


Figure 2.5 Sinais internos do Multiplicador.

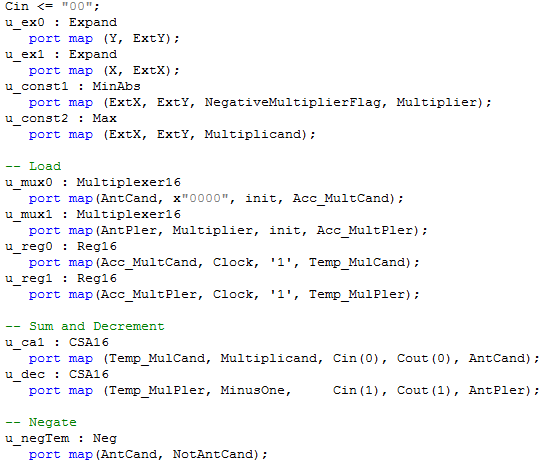


Figure 2.6 Mapeamento dos componentes internos do Multiplicador.

## 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.

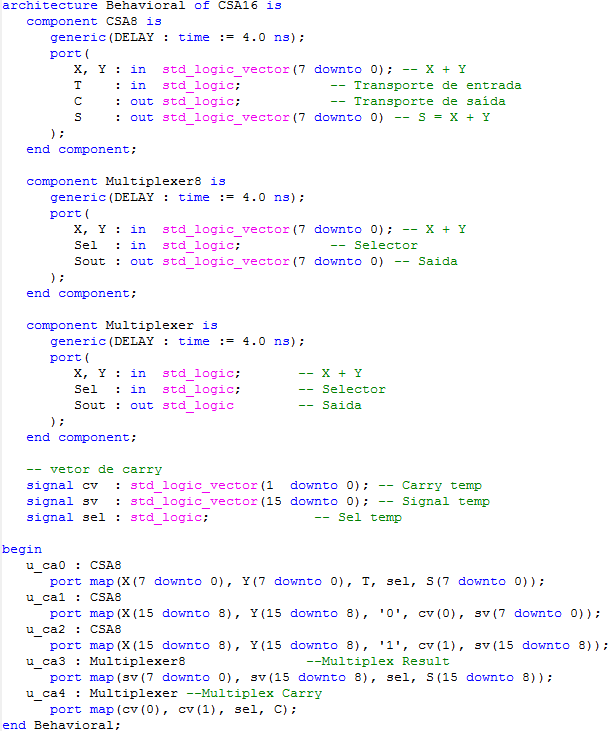


Figure 2.7 Arquitetura do componente CSA16.

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.

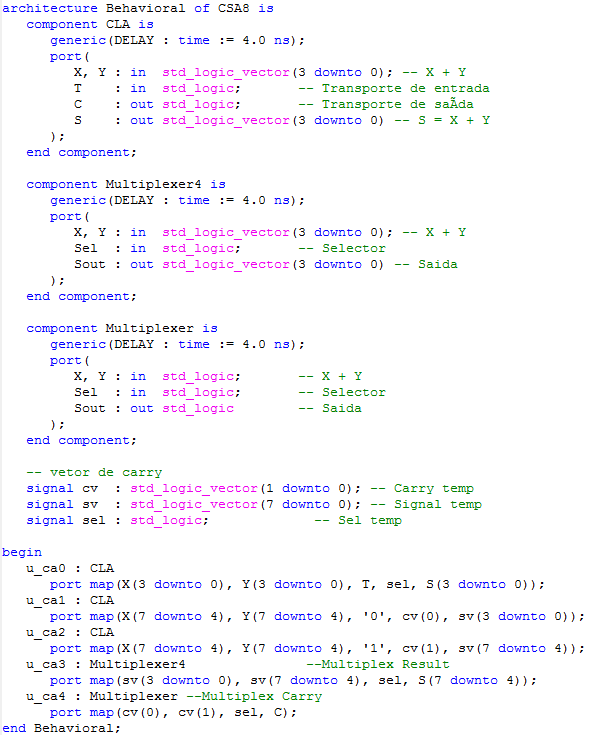


Figure 2.8 Arquitetura do componente CSA8.

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.

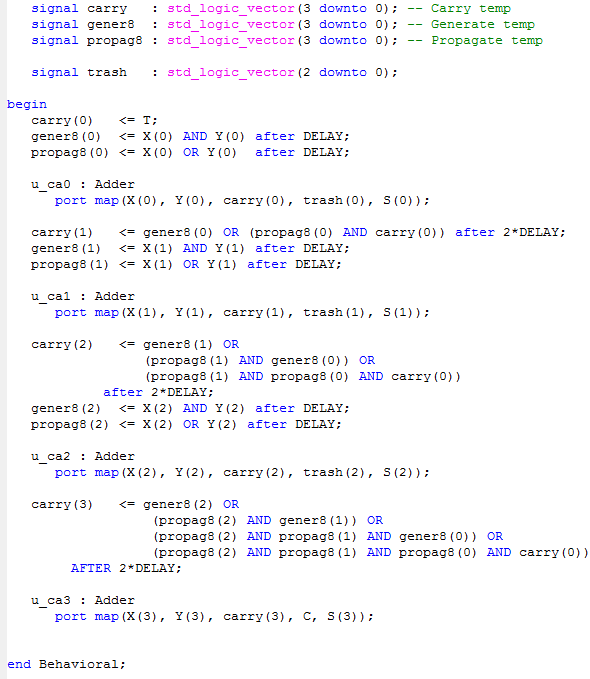


Figure 2.9 CLA de 4 bits.

## 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.

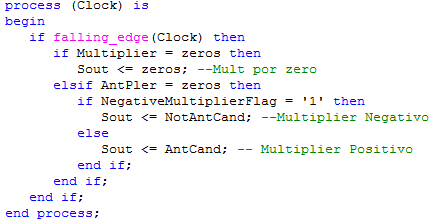


Figure 2.10 Controle de saída de resultados

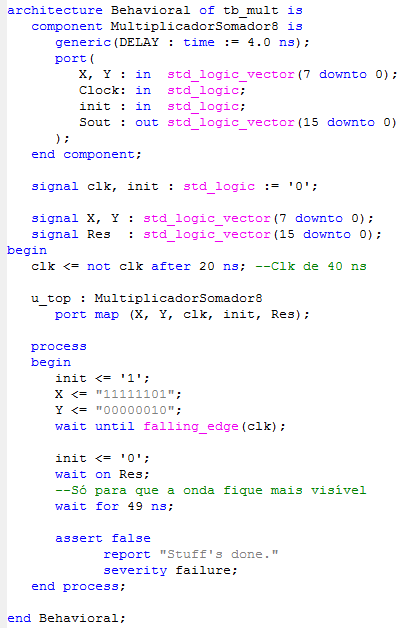


Figure 2.11 Exemplo de TestBench para o MultiplicadorSomador8.

## Diferenciação na multiplicação de não sinalizados

Para realizar multiplicações não-sinalizadas, tem-se que modificar levemente o funcionamento do MultiplicadorSomador8 e de seus componentes internos “Expand” e “Max”. Os outros, apesar de considerarem valores negativos, não precisam ser substituídos devido à expansão de sinal. Por exemplo, ao utilizar o componente “MinAbs”, não haverá diferença, pois, o bit mais significativo nunca será um (e.g. o componente não o considerará negativo), uma vez que a expansão de números não sinalizados sempre os preenche com zeros.

O componente nomeado “U\_Max” faz a seleção do valor máximo entre as duas entradas de forma muito semelhante ao “Min”, descrito anteriormente: Realiza-se a subtração “X-Y”, se o *MSB* do resultado for um, seleciona-se Y, se não, X. A figura 2.12 mostra tal componente.

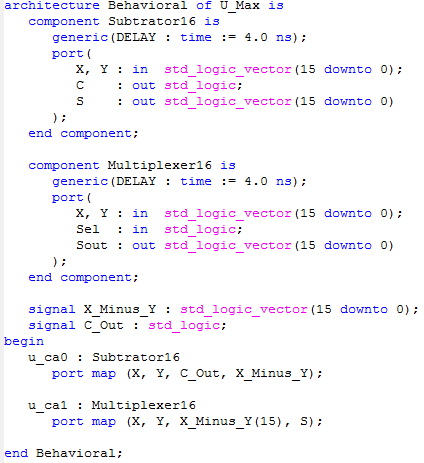


Figure 2.12 U\_Max.

A expansão do sinal pode ser simplificada para uma simples inclusão de zeros ao lado do *MSB*. A figura 2.13 mostra tal inclusão.

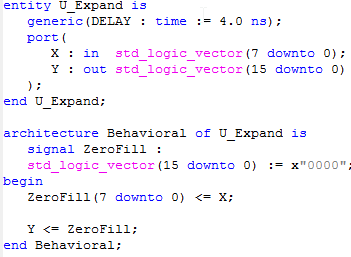


Figure 2.13 U\_Expand

O Multiplicador não sinalizado (alcunhado de “U\_MultiplicadorSomador8”) tem o mapeamento de portas e os sinais de controle levemente alterados. Ao invés de utilizar “Expand” e “Max”, substitui por “U\_Expand” e “U\_Max”, respectivamente e exclui a necessidade de manter um sinal acumulado negado adicional. A estrutura de controle de saída também pode ser simplificada. O testbench só precisa ser alterado no componente que utiliza para passar as entradas, de “MultiplicadorSomador8” para “U\_MultiplicadorSomador8”. As figuras 2.14 e 2.15 mostram o mapeamento e o controle, respectivamente.

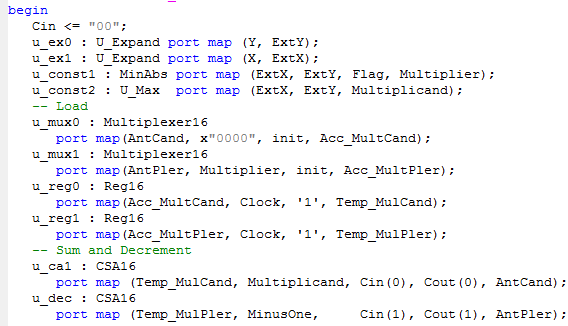


Figure 2.14 Mapeamento do "U\_MultiplicadorSomador8".

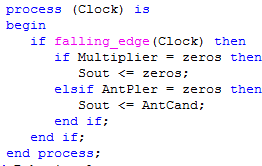


Figure 2.15 Controle de saída do "U\_MultiplicadorSomador8".

# 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.

O somador CSA16 leva 20ns para convergir à um resultado, uma vez que o CSA8 leva 16 ns e o Multiplexador interno leva 4 ns para realizar a seleção. O CSA8, por sua vez, leva tal tempo pois o CLA leva 12ns e o multiplexador leva mais 4 ns. Os registradores levam 8 ns após a descida do Clock para carregar o resultado na saída.

Considerando os tempos, o Clock utilizado foi com um período de 40 ns, iniciando em zero (20 ns em zero, depois 20 ns em um, e assim sucessivamente). O multiplicador não sinalizado pode ser operado com um clock de 32 ns de período, devido à ausência de um negador de acumulador (Necessário no sinalizado para realizar multiplicação com multiplicadores negativos). As figuras 3.1 a 3.3 mostram testes de tempo com os somadores de 4 a 16 bits, em ordem.

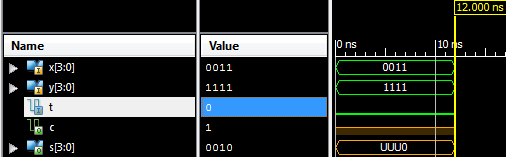


Figure 3.1 Teste com CLA 4 bits.

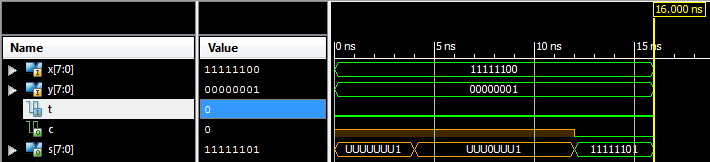


Figure . Teste com CSA8 bits.

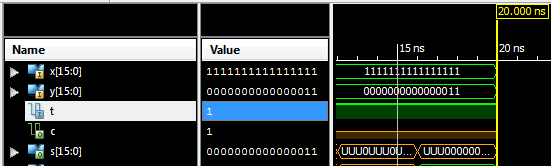


Figure . Teste com CSA 16 bits.

# Conclusão

Há casos onde a estratégia implementada pode ser melhor que o Algoritmo de Multiplicação de Booth, por exemplo. Enquanto o alg. De Booth leva y (Sendo este o número de bits do multiplicador) ciclos de clock + 1 de inicialização, o multiplicador somador leva z (Sendo este o módulo do multiplicador) ciclos + 1 de inicialização. Quando se multiplica -3 \* 2 (8 bits) usando Booth, por exemplo, ter-se-ia o resultado em 8 + 1 ciclos de clock; enquanto no implementado ter-se-ia o mesmo resultado em 2 + 1 ciclos. A figura 4.1 mostra tal exemplo.

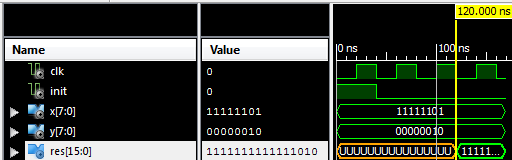


Figure 4.1 Multiplicação entre -3 e utilizando o "MultiplicadorSomador8".

Consequentemente, quando se tem um multiplicador cujo módulo é maior que o número de bits utilizado para representa-lo, o Multiplicador de Booth supera o somador. Como exemplo, considere a multiplicação sinalizada -126\*-122 com 8 bits, enquanto Booth calcularia em 9 ciclos, o somador calcularia em 122 ciclos (|-122| + 1). A figura 4.2 mostra este exemplo.

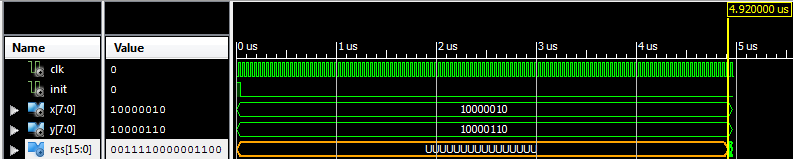


Figure 4.2 Multiplicação entre -126 e -122 utilizando o "MultiplicadorSomador8"

Outra nuance é que Booth apenas considera inteiros sinalizados, então caso fosse multiplicar 128\*2 com 8 bits, necessitaria de 9 bits para que a multiplicação não fosse interpretada como -128\*2, o que ocuparia mais recursos, um ciclo a mais de clock quando comparado com o mesmo algoritmo sinalizado de mesmo tamanho e um somador de tamanho equivalente (O de Booth utiliza somadores do mesmo número de bits que os operandos).

Obviamente, ainda seria melhor que o não sinalizado “Multiplicador Somador”, pois este precisaria alterar os componentes “Max” e “Expand”, como discutido na seção “[Diferenciação na multiplicação de não sinalizados](#_Diferenciação_na_multiplicação)”, e ainda possui a característica de precisar de somadores maiores que os operandos da multiplicação e a grande quantia de ciclos com multiplicadores grandes.

# Anexos

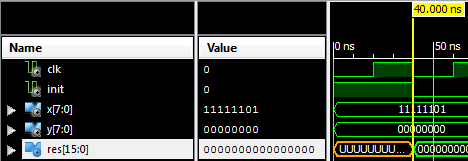


Figure 5.1 Multiplicação sinalizada entre -3 e 0. Resultado em 1 ciclo.

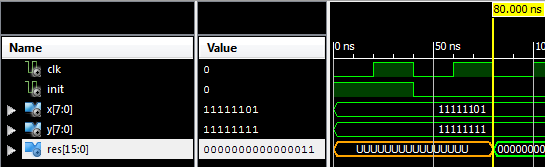


Figure 5.2 Multiplicação sinalizada entre -3 e -1. Resultado em 2 ciclos (Init + |-1|).

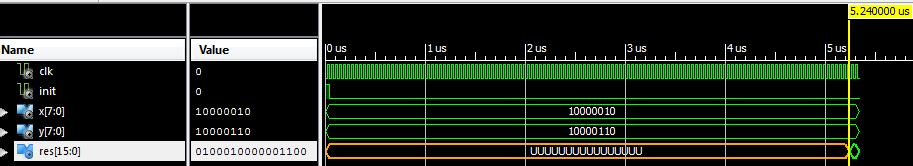


Figure 5.3 Multiplicação não sinalizada entre 130 e 134. Resultado em 131 ciclos (Init + |130|).

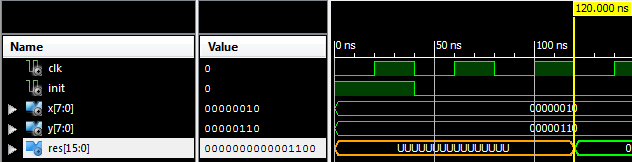


Figure 5.4 Multiplicação não sinalizada entre 2 e 3. Resultado em 3 ciclos (Init + |2|).