Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Curso de Engenharia de Telecomunicações – TGT

Programa de Educação Tutorial – PET
Grupo PET-Tele

Tutorial

Implementação de Filtro Digital em EPLD

(Versão: A2016M05D19)

Autores: Gustavo Araujo Machado (2016)

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Maio / 2016

Sumário

1 Introdução

No ano de 2015 o Grupo PET-Tele possuiu integrantes que cursavam os quinto e sexto períodos do curso de graduação em Engenharia de Telecomunicações. Nestes períodos são cursadas as disciplinas de Técnicas Digitais e Processamento Digital de Sinais. O tutor ministra atualmente (TET, 2015) as disciplinas de Circuitos Digitais e Processamento Digital de Sinais e já realizou diversas implementações em *hardware* de projeto de sistemas digitais. Conhecendo melhor os conteúdos apresentados nestas disciplinas, os alunos do Grupo demonstraram interesse em aplicar os conceitos em projetos práticos.

Sistemas digitais podem ser implementados em Dispositivos Lógicos Programáveis (*Programmable Logic Devices* - PLD). PLDs são dispositivos de função indefinida que precisam ser carregados com uma programação que definirá seu funcionamento como *hardware*. Por este motivo, diferem de portas lógicas simples que possuem funções bem definidas. PLDs são largamente utilizados em sistemas digitais devido a facilidade de reprogramação e, consequentemente, enorme campo de aplicação.

Filtros digitais são sistemas digitais construídos a partir de circuitos combinacionais que respondem de determinada maneira a um sinal aplicado em sua entrada. Filtros digitais podem ser classificados como: Filtros de resposta ao impulso infinita (*infinite Impulse Response* - IIR) cuja resposta ao estímulo é infinita, ou seja, aplicada uma entrada do tipo impulsiva sua resposta de propagará infinitamente. Isso se dá pelo uso de recursividade (realimentação) no sistema. Filtros de resposta ao impulso finita (*Finite Impulse Response* - FIR) cuja resposta ao estímulo é finita, ou seja, após determinado período de tempo a resposta será nula. Isso se dá pelo não uso de recursividade (realimentação) no sistema, pelo uso de coeficientes do sistema que permitam esse tipo de resposta e pela associação em cascata de sistemas do tipo IIR.

O projeto consiste em implementar um Filtro Digital IIR de segunda ordem, com entrada serial e barramento de saída de 8 *bits*, definido por uma equação de diferenças cujos coeficientes definem seu funcionamento. A equação de diferenças que define o sistema é
$y[n] = (-a1)y[n-1] + (-a2)y[n-2] + (b0)x[n] + (b1)x[n-1] + (b2)x[n-2]$
e o sistema pode ser caracterizado pelo esquema a seguir, em sua Forma Direta I (FDI).

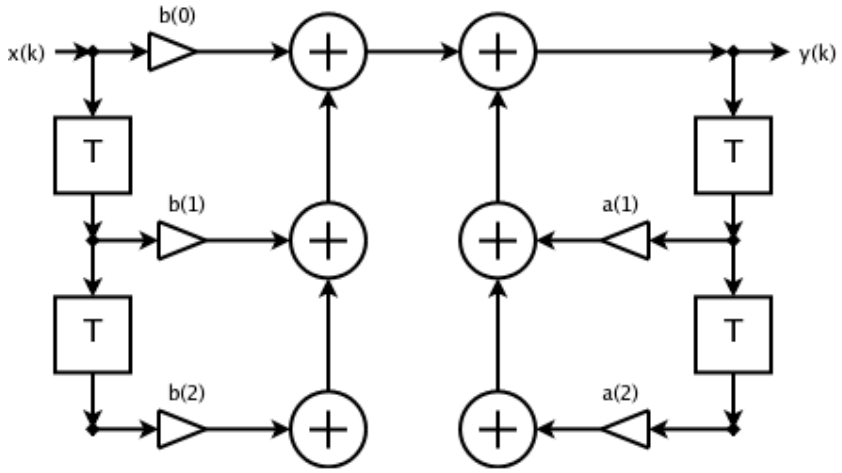

Figura 1: Representação do sistema em sua Forma Direta I

Nos capítulos seguintes são apresentados os subsistemas implementados para a construção do sistema principal, o filtro.

2 Ambiente de software utilizado

Existem diversos softwares disponíveis no mercado para design de circuito em chip, no entanto, o Grupo PET-Tele busca o uso de ferramentas gratuitas acessíveis aos alunos de graduação e demais membros da comunidade acadêmica. Por este motivo, optamos por utilizar o software Altera QuartusII Version 9.1 Web Edition. Vale destacar que o software descrito não é uma ferramenta didática, mas uma plataforma de projeto e simulação de sistemas digitais, além de gravação em chip dos projetos criados. O QuartusII possibilita a criação de projetos hierárquicos em design gráfico e descrição textual de hardware. Para nosso projetos utilizamos o design gráfico, ou arquivos de diagramas em blocos (*Block Diagram Files* - BDF) para a construção dos circuitos. Os arquivos BDF realizam a construçãoo do *hardware* pela interpretaçãoo de circuitos construídos por meio de símbolos (portas lógicas, trilhas, *flip-flops* e demais componentes de circuitos digitais).

3 Implementação

3.1 Registradores

Em sistemas digitais, muitas vezes nossas aplicações não são simplesmente instantâneas, ou seja, nosso sistema não responde imediatamente ao estímulo aplicado. Para conseguirmos dinamizar a operação do sistema, necessitamos de componentes de memória capazes de armazenar informações, sejam pré estabelecidas no projeto, sejam informações de entrada.Para aplicações desse tipo, temos os registradores. Registradores são unidades de memória capazes de armazenar alguma quantidade de *bits* e portanto, podem ser construídos com uso de *flip-flops* ou *latches*.

3.1.1 Registrador SISO - *Single Input - Single Output*

Registradores SISO são componentes de memória com carga serial, ou seja, carregam um *bit* por vez. internamente, ocorre o deslocamento da informação entre os *flip-flops* que o constituem a cada pulso de *clock*. No caso da ativação da porta assíncrona *clear* ocorrer, todos os *flip-flops* têm sua saída forçada para nivel baixo (zero). De modo geral, o sistema completo tem sua saída correspondente a saída do último flip-flop da cadeia sequêncial interna. Foram utilizados *flip-flops* tipo D em cascata e síncronos. A entrada e a saída do dispositivo são seriais. O circuito projetado foi:

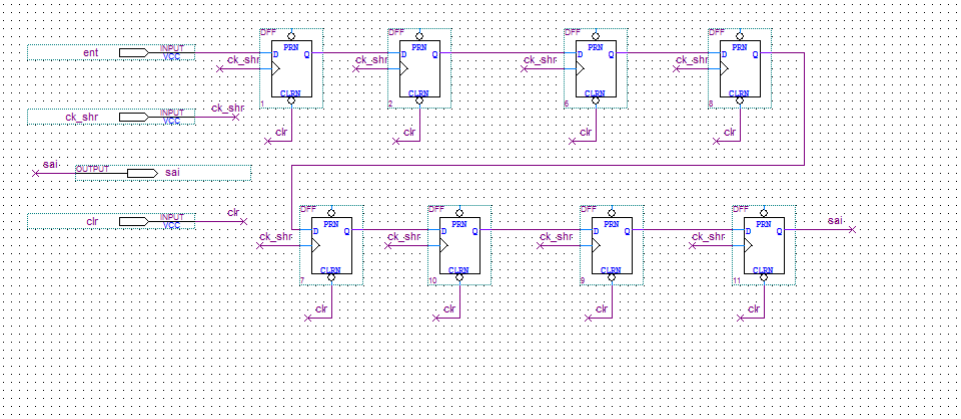

Figura 2: Circuito do Registrador de entrada e saída seriais

O símbolo criado para o circuito foi:

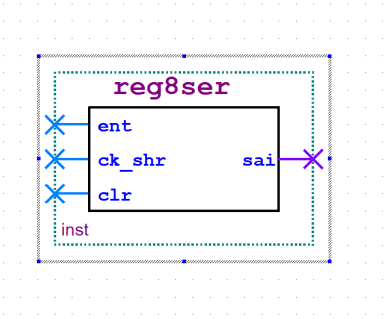

Figura 3: Encapsulamento do circuito Registrador de entradas e saídas seriais

3.1.2 Contador em anel

O contador em anel é um dispositivo criado a partir de um registrador de deslocamento (shift register), porém com a característica de repetir a sequência gravada em um intervalo fixo. O contador em anel pode ser utilizado como gerador de sinal de controle de outros componentes do sistema. O contador abaixo é iniciado com zeros em todos seus registradores, exceto o primeiro da cascata. Como nosso sistema trabalha com uma palavra binária de 8 *bits*, o sinal se desloca dentro do dispositivo no número de pulsos até que a saída em nível alto seja lido na saída do último *flip-flop* da cascata. O comando de *clear* é o sinal que inicia o contador e o *clock* é o comando que controla a transferência dos sinais entre os *flip-flop*s.

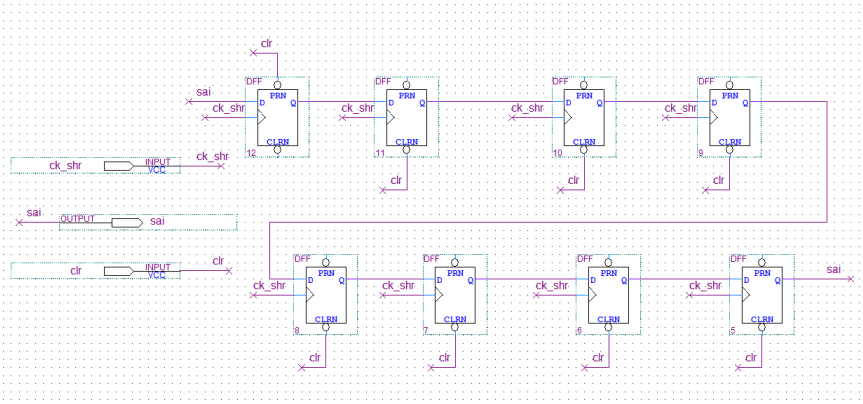

Figura 4: Circuito Contador em anel

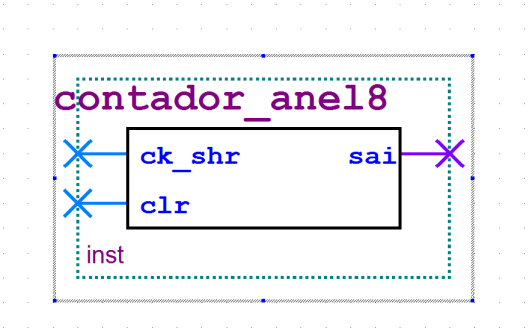

Figura 5: Encapsulamento do Circuito contador em anel

3.1.3 Registrador MIMO - *Multiple Input - Multiple Output*

O registradores MIMO são componentes de memória com carga múltipla, ou seja, são capazes de armazenar vários *bits* em apenas um pulso de *clock*. Muito utilizados pela velocidade e capacidade de armazenamento de informação, são úteis em sistemas que demamdam velocidade para descarga de muitos *bits* de uma vez. Sua construção é mais simples que os demais registradores, pois seus *flip-flops* são independentes, porém síncronos.

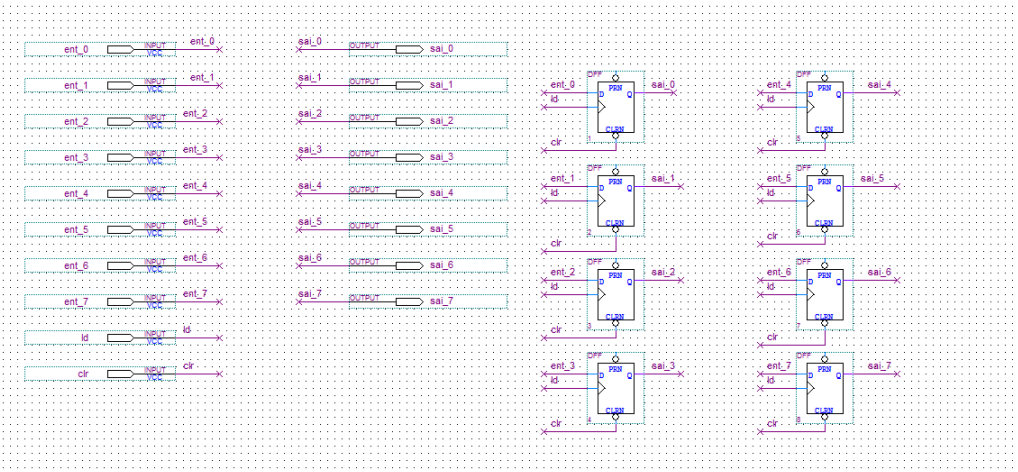

Figura 6: Circuito Registrador de entradas e saídas paralelas

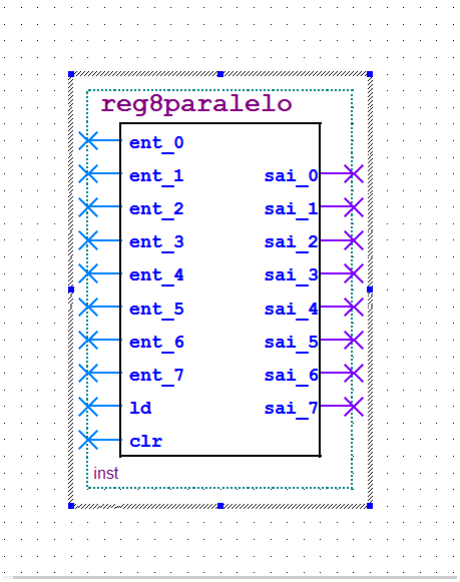

Figura 7: Encapsulamento do ciruito Registrador de entradas e saídas paralelas.

3.1.4 Registrador MISO - *Multiple Input - Single Output*

O circuito proposto consiste em um registrador capaz de receber um barramento de *bits* e dispor na saída apenas um *bit*. Os *bits* saem a partir do LSB. O deslocamento interno dos *bits* são feitos seqeuencialmente pela associação em cascata de *flip-flops*. Para aumentar a eficiência do circuito, é suposto um dispositivo multiplexador na entrada de cada *flip-flop*, para poder selecionar entre a entrada paralela e a saída do *flip-flop* anterior da cascata. O componente é controlado pelo *clock*, pelo comando de *clear* e pelo comando de carga. O *clock* controla a transição de cada *flip-flop* do registrador. O comando de *clear* força todas as saídas para zero. O comando de carga faz com que seja carregada uma palavra binária de 8 *bits*.

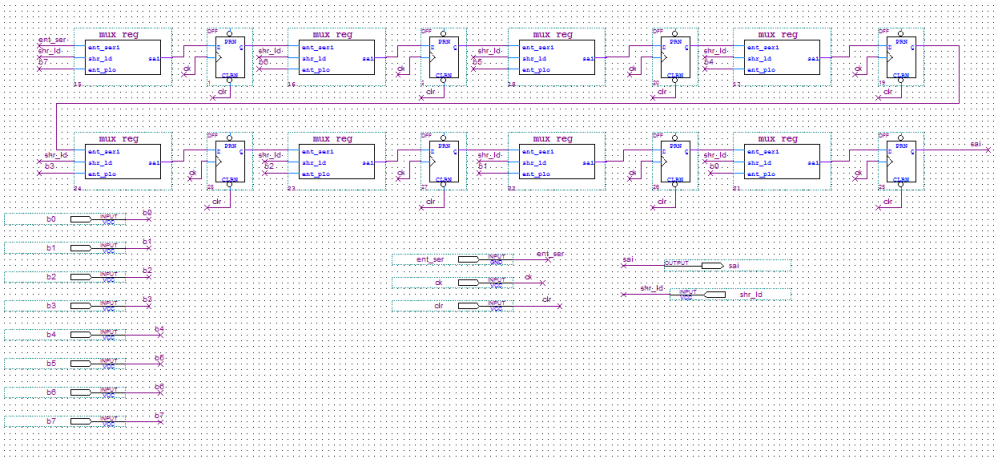

Figura 8: Circuito Registrador de Entradas paralelas e saída serial.

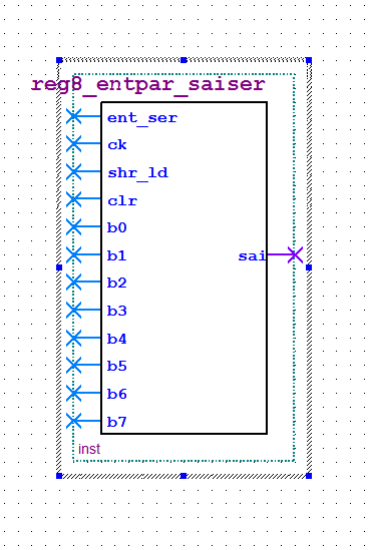

Figura 9: Encapsulamento do circuito Registrador de Entradas paralelas e saída serial.

3.2 Complementador a dois

Podemos construir a tabela verdade que representa o processo de complemento. Supondo uma entrada **E**, um comando que sinalize o inicio da operação de complemento **C2**, um sinal interno sinalizando que o primeiro *bit* 1 já foi encontrado "Achou antes"A_{i-1}, um sinal interno sinalizando que um *bit* 1 acaba de ser encontrado "Achou aqui"A_i, e a saída S.

c2	A_{i-1}	E	A_i	S
0	0	0	X	0
0	0	1	X	1
0	1	0	X	0
0	1	1	X	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Através da tabela verdade descrita acima, podemos escrever uma expressão booleana que representa o circuito que pode ser implementado para realizar o processo de complemento a dois.A expressão segue abaixo:
$S = c2.A_{i-1}.\overline{E} + \overline{c2}.E + \overline{A_{i-1}}.E$
e
$Ai = A_{i-1} + E$

Assim, cada bloco do sistema complementador foram construídos da seguinte forma:

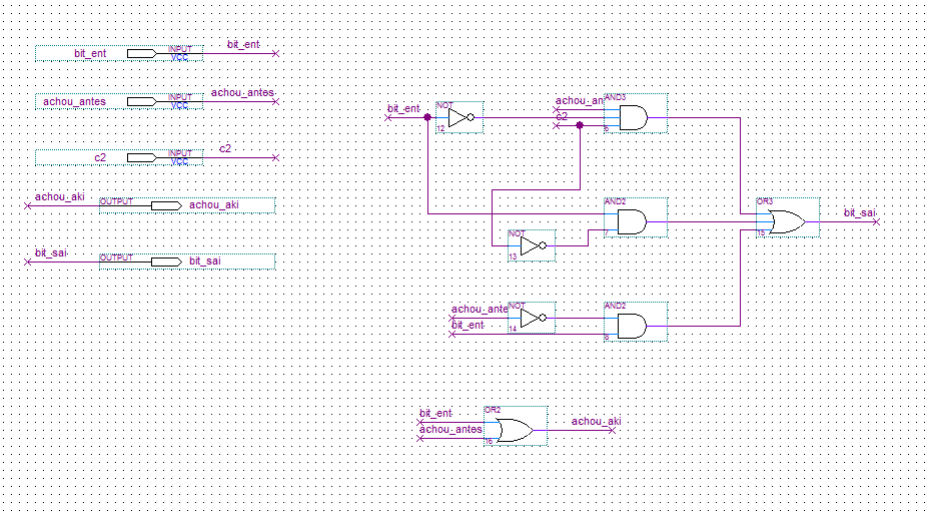

Figura 10: Circuito do bloco individual Complementador a dois.

Foi criado o símbolo que o representa para ser utilizado nos níveis mais altos do projeto. Com este símbolo, construímos a sequência de estágios do complementador

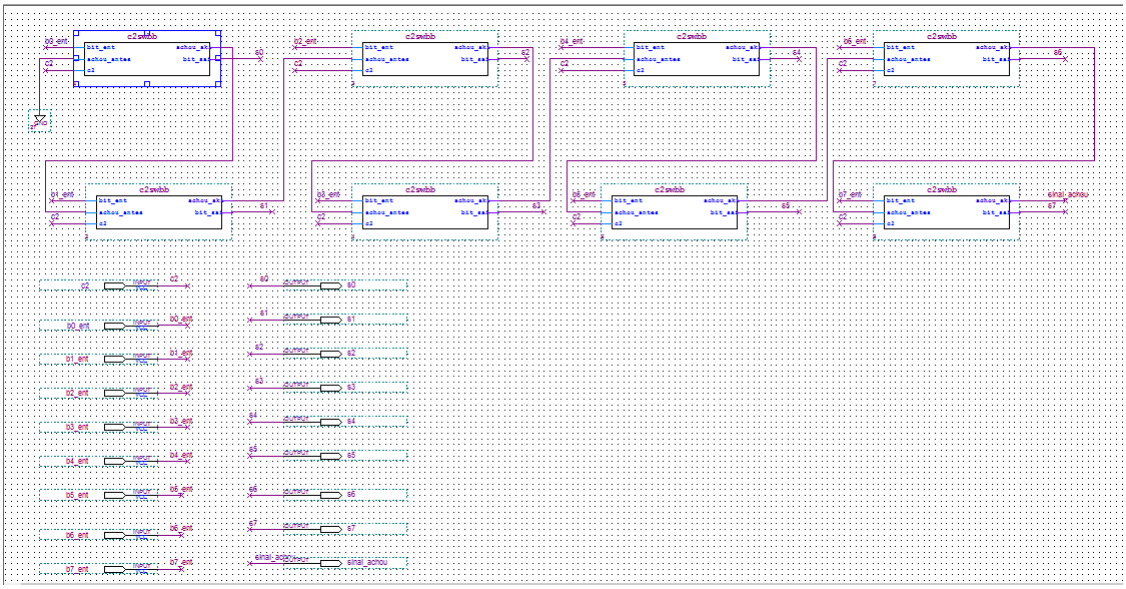

Figura 11: Circuito Complementador a dois.

Finalmente, o símbolo para o sistema complementador a dois.

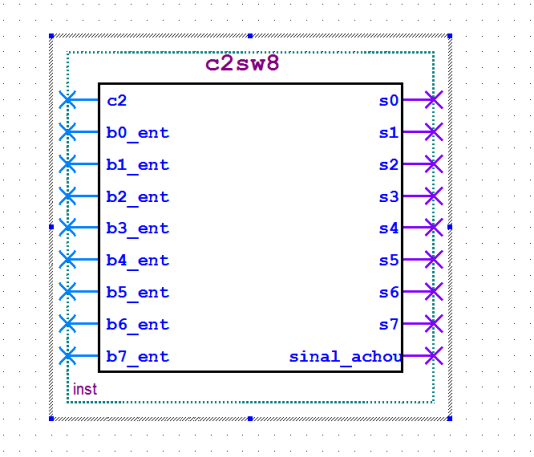

Figura 12: Encapsulamento do circuito Complementador a dois.

3.3 Somador

Para somar duas sequências de *bits* de mesmo tamanho, precisamos realizar a soma um a um, do *bit* menos significativo (*Less significant bit* - LSB) ao *bit* mais significativo (*Most significant bit* - MSB)de cada sequência levando em consideração o *carry* da soma anterior. Com base nisso, conhecendo a

operação básica da soma entre dois *bits*, podemos montar a tabela verdade que melhor representa a operação que queremos representar. Supondo duas entradas A e B, temos para cada combinação da entrada as possíveis combinações de saída S:

c_in	**A**	**B**	**S**	**c_out**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Construindo o mapa de Karnaugh para a saída S e o *carry* resultante *carry*_out, podemos encontrar as funções booleanas que melhor representam o problema proposto.

***carry*_in/AB**	**00**	**01**	**11**	**10**
0	0	1	0	1
1	1	0	1	0

$$S = (A \bigoplus B) \bigoplus c\_in$$
$$c\_out = BC + AC + AB$$

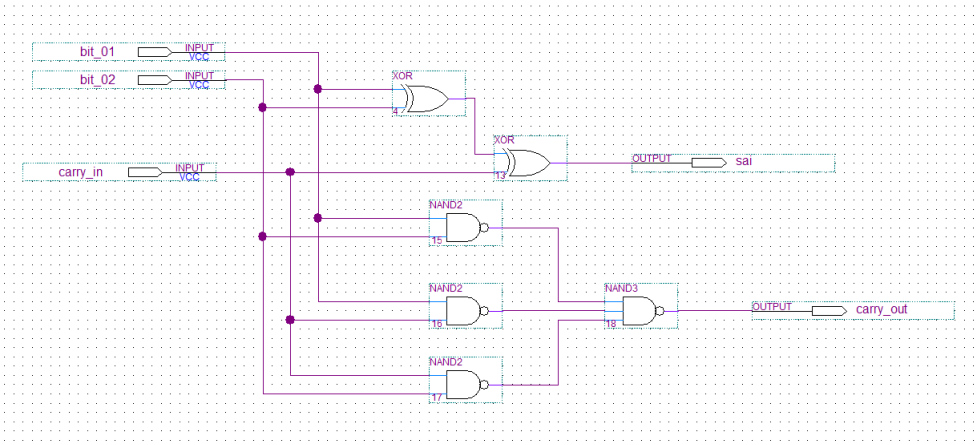

Figura 13: Circuito do bloco individual Somador.

Foi criado o símbolo que o representa para ser utilizado nos níveis mais altos do projeto. Com este símbolo, construímos a sequência de estágios do somador.

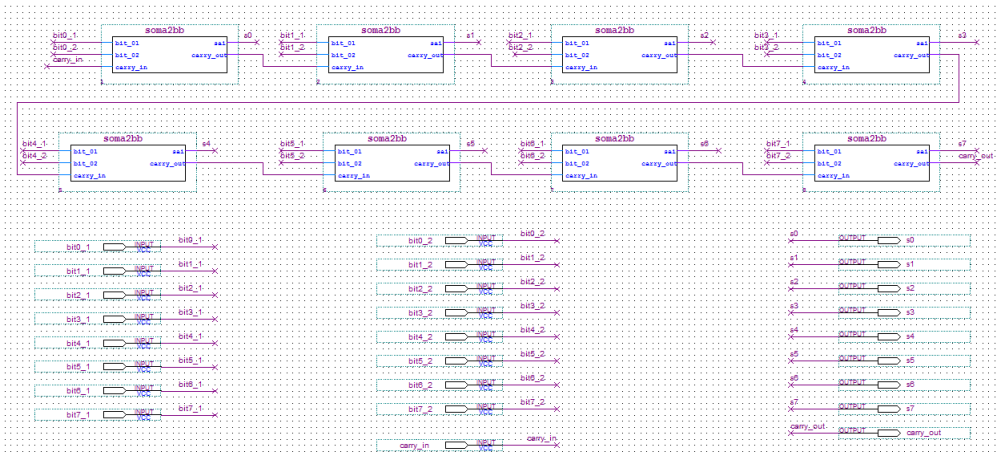

Figura 14: Circuito Somador.

Finalmente, o símbolo para o sistema somador.

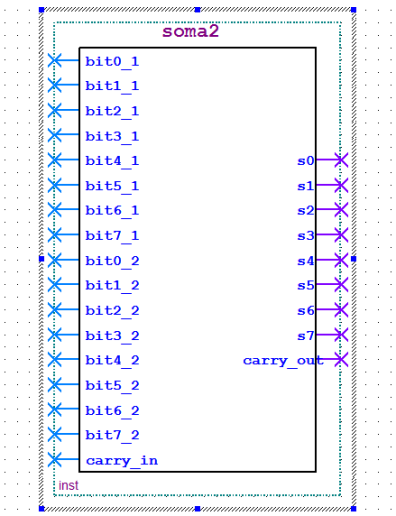

Figura 15: Encapsulamento do circuito Somador.

3.4 Saturador

Quando realizamos a soma de dois números binários existe a possibilidade de o resultado precisa ser representado por mais dígitos que os utilizados na

soma. O problema se agrava quando realizamos uma subtração entre dois números binários com sinal e o resultado não condiz com o sinal do valor de maior magnitude. Chamamos esse tipo de acontecimento de *overflow* na soma. Para evitar este tipo de situação, dado que um valor ultrapassa o valor máximo representável para uma sequência binária, podemos realizar o procedimento que chamamos de saturar o valor, ou seja, levaremos ao máximo representável. Como trabalhamos com números binários com sinal, devemos levar em consideração os casos possíveis de resultados de somas. Por este motivo, foi criado o sistema para saturar o resultado da soma, caso ocorra o *overflow*. Para isso, utilizamos os dígitos MSB de cada palavra binária somada e comparamos com seu resultado.

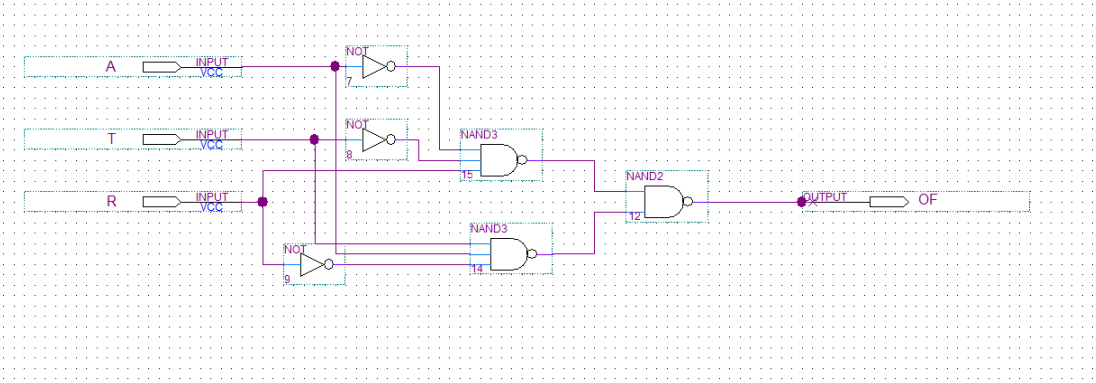

Figura 16: Circuito Detector de *overflow*.

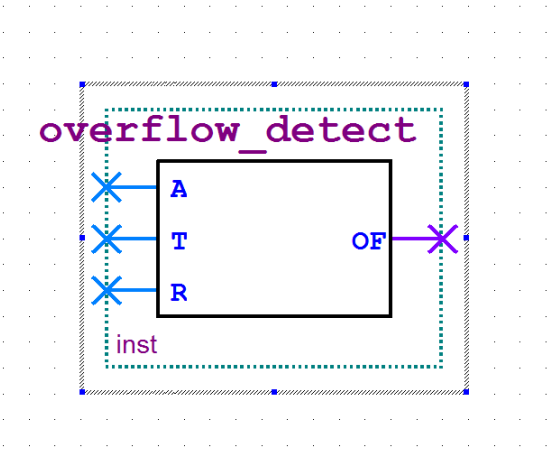

Figura 17: Encapsulamento do circuito Detector de *overflow*.

Detectado o *overflow* na soma, o dispositivo detector gera um sinal que aciona imediatamete o sistema saturador. O Saturador é composta por seletores de 2 entradas, 1 saída e 1 chave seletora os quais chamamos de mux_2x1_bb. é selecionada a saída da soma caso o *overflow* não ocorra, ou selecionado o valor do MSB da palavra binária caso o *overflow* ocorra. Além disso, caso o *overflow* ocorra, significa que o MSB da palavra binária está errada e então é invertida utilizando-se uma porta lógica inversora.

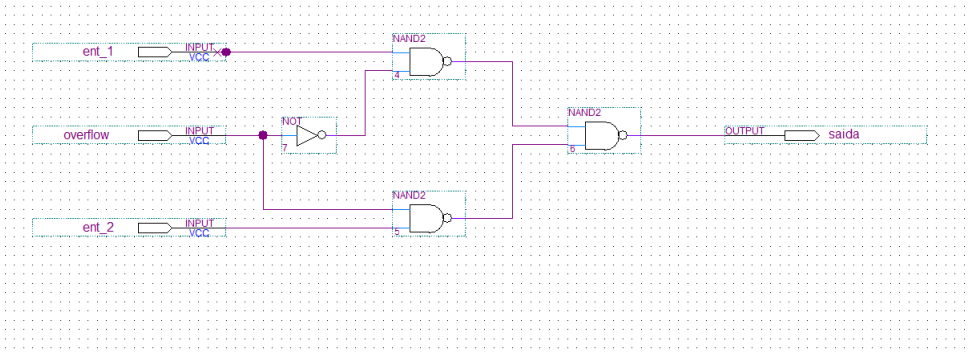

Figura 18: Circuito Seletor 2 entradas para 1 saída.

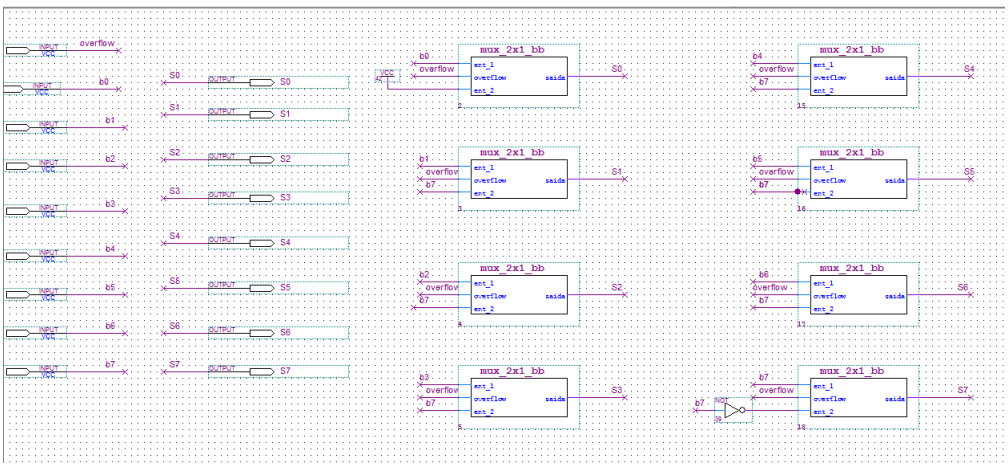

Figura 19: Circuito Saturador.

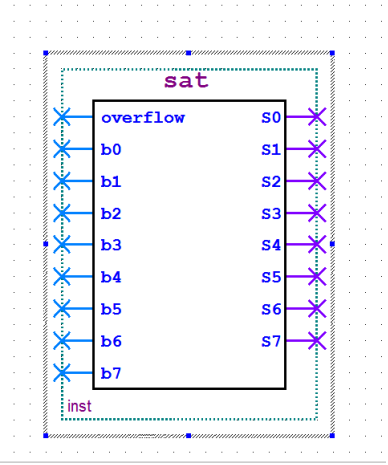

Figura 20: Encapsulamento do circuito Saturador.

3.5 Multiplicador

Um sistema multiplicador ou escalador, como o nome já diz, multiplica o valor de sua entrada por uma constante. No projeto foi utilizado um multiplicador por 2. Multiplicar por dois é o mesmo que deslocar a palavra binária em direção ao seu MSB (para a direita) acrescentando zero em seu LSB e descartando o antigo MSB. Criamos o multiplicador acionado por um comando externo, para caso a multiplicação não seja necessária a entrada é repetida na saída. Abaixo o sistema multiplicador:

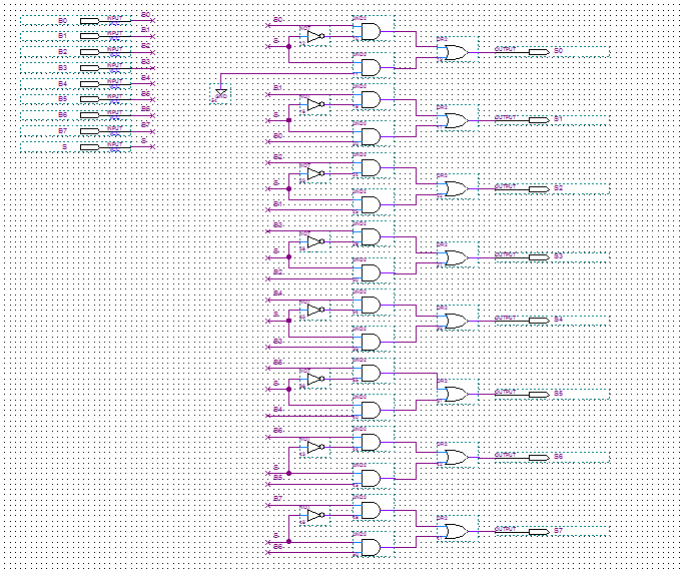

Figura 21: Circuito multiplicador

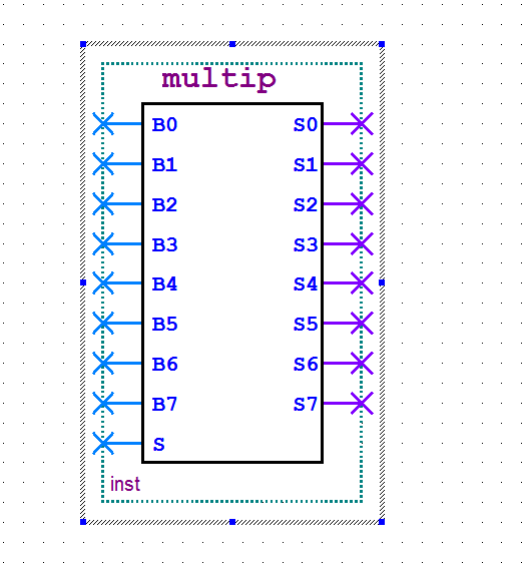

Figura 22: Encapsulamento do circuito Multiplicador.

3.6 Memória RAM estática

Criamos a memória RAM com o uso do recurso de implementação em linguagem de descrição de *hardware*, o VHDL (***V****ery* ***H****igh Speed Integrated Circuit* ***H****ardware* ***D****escription* ***L****anguage*). Fizemos o endereçamento com os

5 *bits* que representam, respectivamente, os valores dos coeficientes a2, a1, b2, b1 e b0 somados. Por exemplo, a sequência 00011 representa o caso b1 + b0. A saída é um barramento de dados de 8 *bits*. Utilizamos a biblioteca padrão para representação de *bits* em VHDL *ieee.std_logic_1164.all*.

```
-- ---------- ---------- ---------- ---------- ----------
--LIBRARY ieee;
--USE ieee.std_logic_1164.all;
-- ---------- ---------- ---------- ---------- ----------
--
-- ---------- ---------- ---------- ---------- ----------
ENTITY rom_norm IS
  PORT( address: in  bit_VECTOR(4 downto 0);
        data   : OUT bit_VECTOR(7 downto 0)
      );
END ENTITY rom_norm;
-- ---------- ---------- ---------- ---------- ----------
--
-- ---------- ---------- ---------- ---------- ----------
ARCHITECTURE arch_rom_table OF rom_norm IS
--
BEGin
--
  data <= B"00000000" WHEN address = B"00000" ELSE
          B"00000001" WHEN address = B"00001" ELSE
          B"00000011" WHEN address = B"00010" ELSE
          B"00000100" WHEN address = B"00011" ELSE
          B"00000001" WHEN address = B"00100" ELSE
          B"00000010" WHEN address = B"00101" ELSE
          B"00000100" WHEN address = B"00110" ELSE
          B"00000101" WHEN address = B"00111" ELSE
          B"11010101" WHEN address = B"01000" ELSE
          B"11010110" WHEN address = B"01001" ELSE
          B"11011000" WHEN address = B"01010" ELSE
          B"11011001" WHEN address = B"01011" ELSE
          B"11010110" WHEN address = B"01100" ELSE
          B"11010111" WHEN address = B"01101" ELSE
          B"11011001" WHEN address = B"01110" ELSE
          B"11011010" WHEN address = B"01111" ELSE
          B"01100101" WHEN address = B"10000" ELSE
          B"01100110" WHEN address = B"10001" ELSE
```

```
            B"01101000" WHEN address = B"10010" ELSE
            B"01101001" WHEN address = B"10011" ELSE
            B"01100110" WHEN address = B"10100" ELSE
            B"01100111" WHEN address = B"10101" ELSE
            B"01101001" WHEN address = B"10110" ELSE
            B"01101010" WHEN address = B"10111" ELSE
            B"00111010" WHEN address = B"11000" ELSE
            B"00111011" WHEN address = B"11001" ELSE
            B"00111101" WHEN address = B"11010" ELSE
            B"00111110" WHEN address = B"11011" ELSE
            B"00111011" WHEN address = B"11100" ELSE
            B"00111100" WHEN address = B"11101" ELSE
            B"00111110" WHEN address = B"11110" ELSE
            B"00111111" WHEN address = B"11111";
--
END ARCHITECTURE arch_rom_table;
-- ---------- ---------- ---------- ---------- ----------
```

3.7 Filtro Digital

O filtro foi construído com o uso dos módulos citados anteriormente. No sistema implementado utilizados as seguintes entradas e seus respectivos nomes:

- uma entrada de dados serial - entrada,
- o *clock* do sistema - *clock*,
- o controle de *clear*, para iniciar o sistema com condições iniciais nulas - clr,
- a saída em um barramento de 8 *bits* - A[7..0].

A dinâmica do sistema funciona da seguinte maneira:

- Os *bits* que representam os coeficientes são obtidos das saídas dos atrasadores (reg8ser), e encaminhados à memória rom.
- O barramento de saída da memória rom é encaminhado à entrada do bloco complementador.
- O barramento de saída do complementador a dois é somado ao barramento de saída do registrador de entradas e saídas paralelas.

- O resultado da soma é verificado e na ocorrência de *overflow* o sinal é saturado, caso contrário apenas repetido.

- O barramento de saída do saturador é encaminhado à entrada do multiplicador.

- O barramento de saída do multiplicador é encaminhado ao registrador de entradas paralelas e saídas seriais (estes alimentarão os coeficientes a2 e a1) e para o registrador de entradas e saídas paralelas, que alimentará o somador e será lido nos pinos de saída do chip.

- Os dois contadores em anel utilizados funcionam como geradores de sinal de controle. Um para o bloco complementador a dois e outro para a carga do registrador de entradas paralelas e saída serial.

Como trabalhamos com palavras binárias que representam números com sinal, caso o MSB e seu valor imediatamente anterior não sejam iguais a informação sobre o sinal do número se perde. Para evitar esse problema, criamos um módulo para comparar o *bit* mais significativo da palavra binária com seu imediatamente anterior e comparar com a ocorrência de *overflow*. Temos então a seguinte situação:

- A multiplicação não deve ocorrer se o MSB e o imediatamente anterior forem diferentes.

- A multiplicação não deve ocorrer se o sinal de *overflow* da soma estiver ativo.

- A multiplicação deve ocorrer se o sinal de complemento a dois estiver ativo.

Condiderando B7 o *bit* mais significativo da palavra binária na entrada do multiplicador e B6 o *bit* imediatamente anterior, OF o sinal de ativação do bloco saturador alertando ocorrencia de *overflow* e C2 o comando para ativação do bloco complementador a dois escrevemos a seguinte expressão booleana:
multi = $(B7 \odot B6).(C2.\overline{OF})$.
Temos então o circuito:

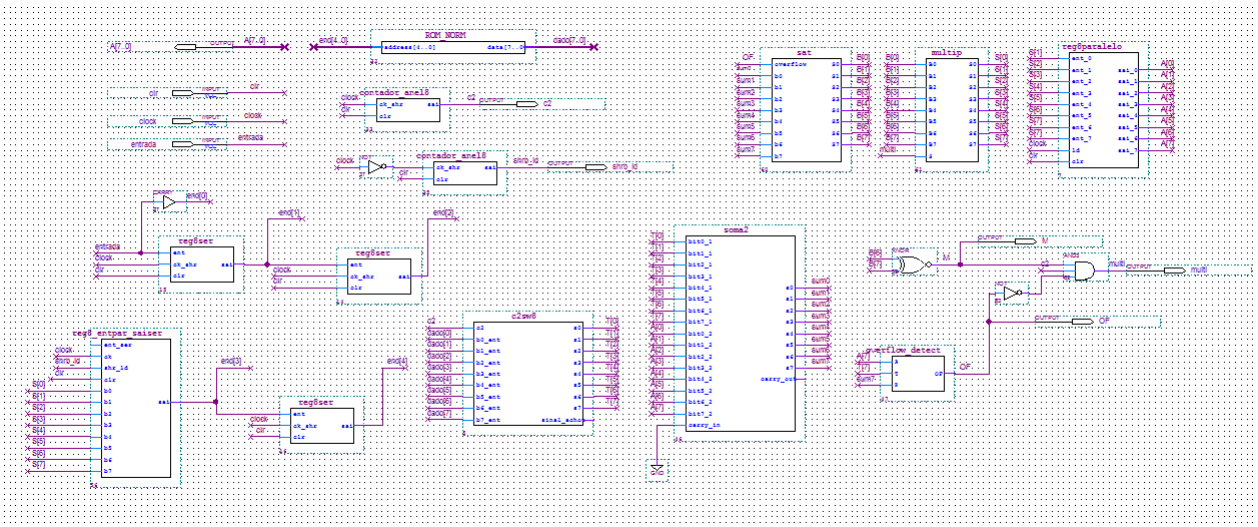

Figura 23: Filtro completo