Ministério da Educação
Universidade Federal de Itajubá

ELTD15 – Laboratório de Sistemas Digitais – Metodologia

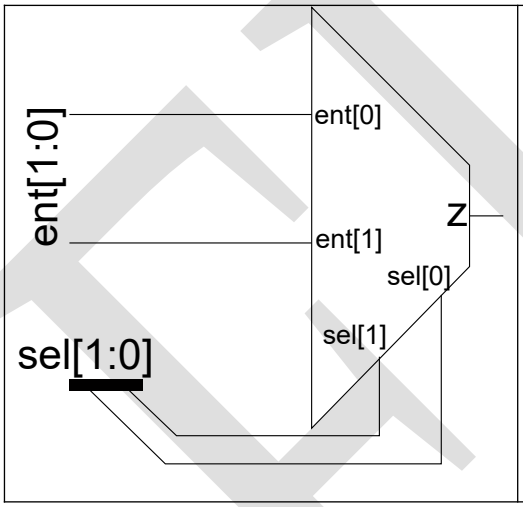
1– Objetivos

- Utilizar a linguagem de descrição de *hardware* Verilog e descrição estrutural;
- Entender a abordagem ascendente e descendente;
- Desenvolver módulo *top* de um multiplicador baseado no algoritmo soma e desloca com descrição estrutural;
- Entender como adicionar vários *modules* diferentes a uma hierarquia *Top no Quartus.*

Utilizar a FPGA MAX10 10M50DAF484C7G

2 - **Mux 16x1 através de 5 Mux 4x1 (abordagem ascendente)**

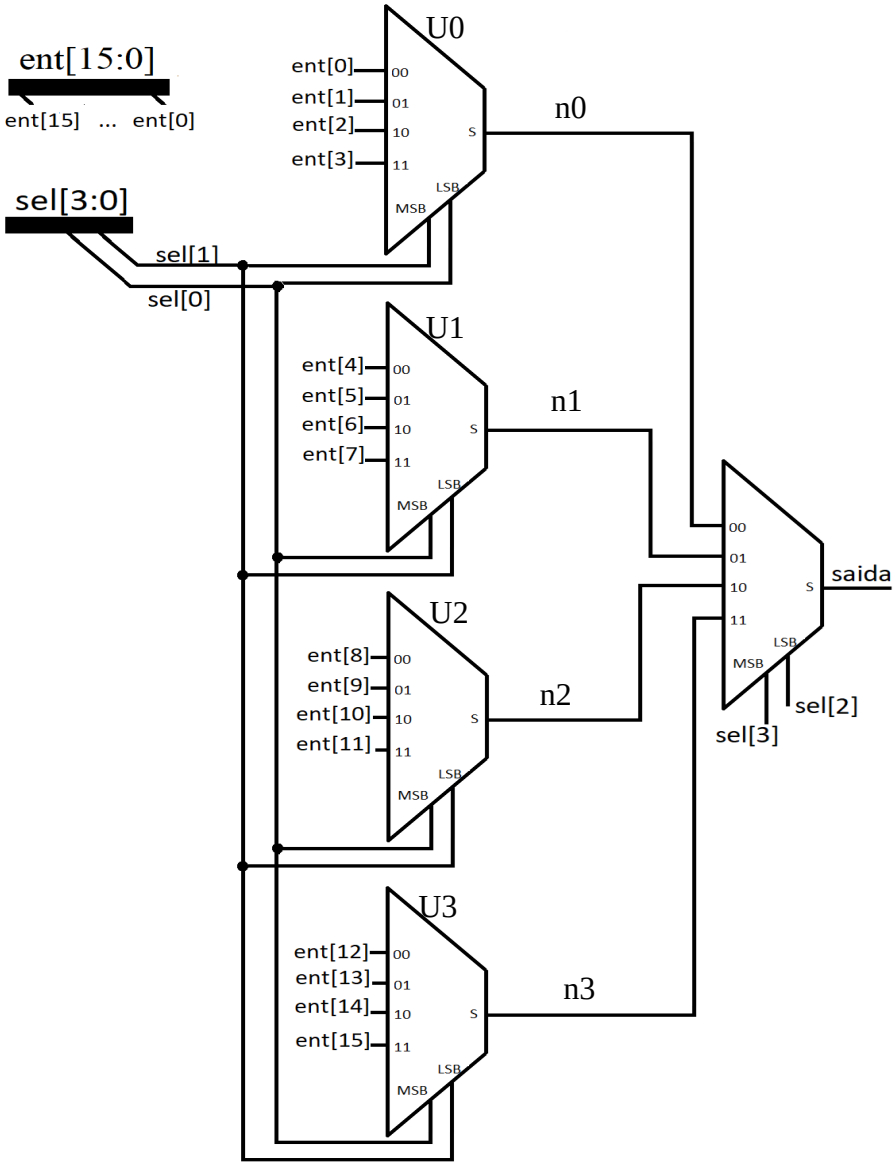
2.1 - Crie o projeto de um Mux parametrizado com a descrição a seguir. O tamanho padrão desse MUX está definido em 2.




```
module mux #(parameter MUX_SIZE = 2
)(
    input [MUX_SIZE-1:0] ent,
    input [$clog2(MUX_SIZE)-1:0] sel,
    output out
);
    assign out = ent[sel];

endmodule
```

2.2 - Compile o projeto.

2.3 – Crie um novo projeto (em pasta diferente do projeto anterior) com nome **mux16x1.** Utilizando o módulo previamente descrito (**mux**), faça a descrição de um Mux 16X1 utilizando descrição estrutural. O circuito esquemático é apresentado abaixo, bem como o seu código estrutural. Os sinais de entrada são fornecidos através de dois vetores: **ent[15:0]** que representa o sinal de entrada e **sel[3:0]** que representa o sinal de seleção. As interconexões entre os multiplexadores é realizada por fios (*wires*). Lembre-se de, ao instanciar o mux parametrizado, passar o novo parâmetro ***MUX_SIZE = 4***, caso contrário serão instanciados mux de 2x1.



- Arquivo mux16x1.v:

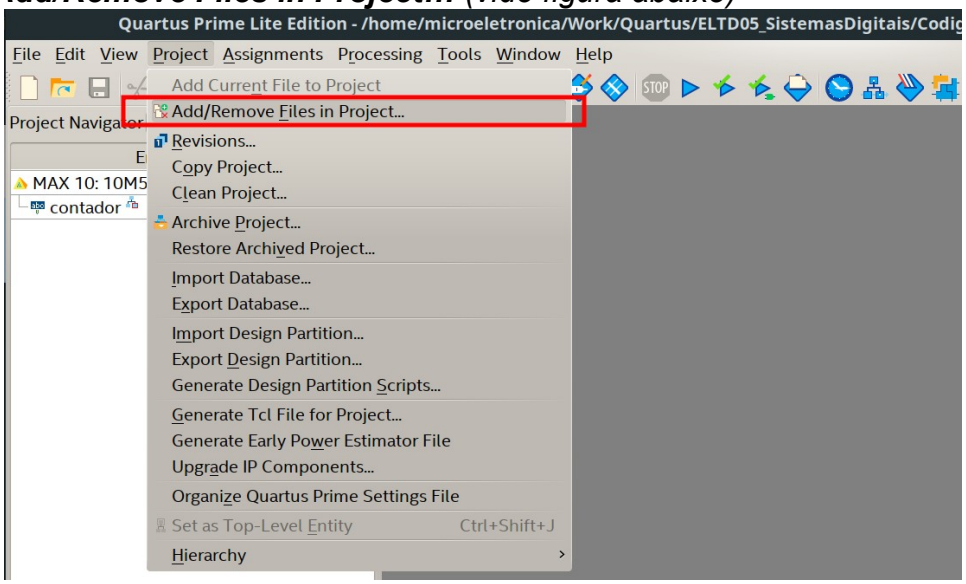
```
module mux16x1 (saida, ent, sel);
    input [3:0] sel;
    input [15:0] ent;
    output saida;
    parameter MUX_SIZE=4;
    wire n0, n1, n2, n3;

    mux #(.MUX_SIZE(MUX_SIZE)) U0
        (ent[3:0], sel[1:0], n0);
    mux #(.MUX_SIZE(MUX_SIZE)) U1
        (ent[7:4], sel[1:0], n1);
    mux #(.MUX_SIZE(MUX_SIZE)) U2
        (ent[11:8], sel[1:0], n2);
    mux #(.MUX_SIZE(MUX_SIZE)) U3
        (ent[15:12], sel[1:0], n3);
    mux #(.MUX_SIZE(MUX_SIZE)) U4
        ({n3, n2, n1, n0}, sel[3:2], saida);

endmodule
```

Para deixar claro, como o multiplexador **mux** foi parametrizado, a criação de um mux de 16x1 baseado neste módulo, seria simplemente:

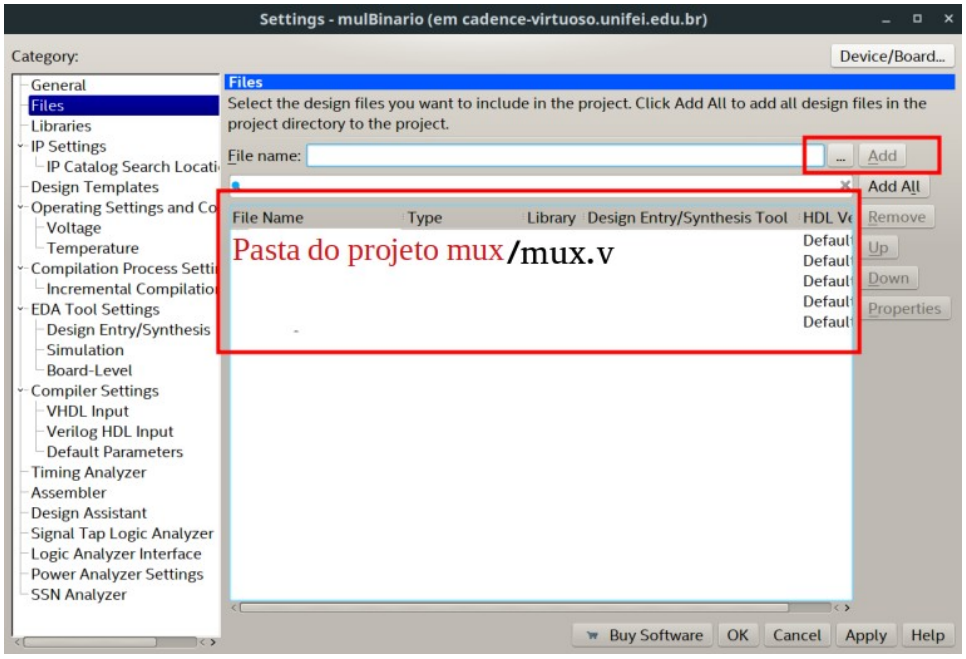
```
module mux16x1 (saida, ent, sel);
    input [3:0] sel;
    input [15:0] ent;
    output saida;
    parameter MUX_SIZE=16;

  mux #(.MUX_SIZE(MUX_SIZE)) U0
       (ent, sel, saida);

endmodule
```

No entanto, como o trabalho necessitará de descrição estrutural, fizemos a criação do Mux de 16 canais com 5 mux de 4 canais para pratica a descrição estrutural. Implemente e simule, também, o multiplexador acima e veja que os dois funcionam corretamente.

2.4 – Antes de compilar o projeto, lembre-se que o Quartus não sabe onde está implementado o module ***mux*** que é utilizado no projeto do mux16x1. Precisamos informá-lo onde está o arquivo.
Vá em ***Project, Add/Remove Files in Project…*** *(vide figura abaixo)*



e **adicione** o *verilog* que implementa o ***mux*** (vide figura abaixo).



Compile o novo projeto e verifique se a síntese RTL é similar ao circuito esquemático dado anteriormente.

3- Multiplicador *Shift* and *Add* (abordagem descendente)

A figura abaixo representa o diagrama de blocos de um microssistema (**Multiplicador**).
(Obs.: mais detalhes deste multiplicador e como seu bloco de controle funciona serão dados no próximo laboratório)
Atente que, cada caixa no diagrama representa um módulo específico, a ser implementado posteriormente. Existem, então, 4 módulos (**ACC**, **CONTROL**, **Counter** e **Adder**) que deverão ser instanciados e interconectados, através de **descrição estrutural**, no módulo *top* **Multiplicador**.

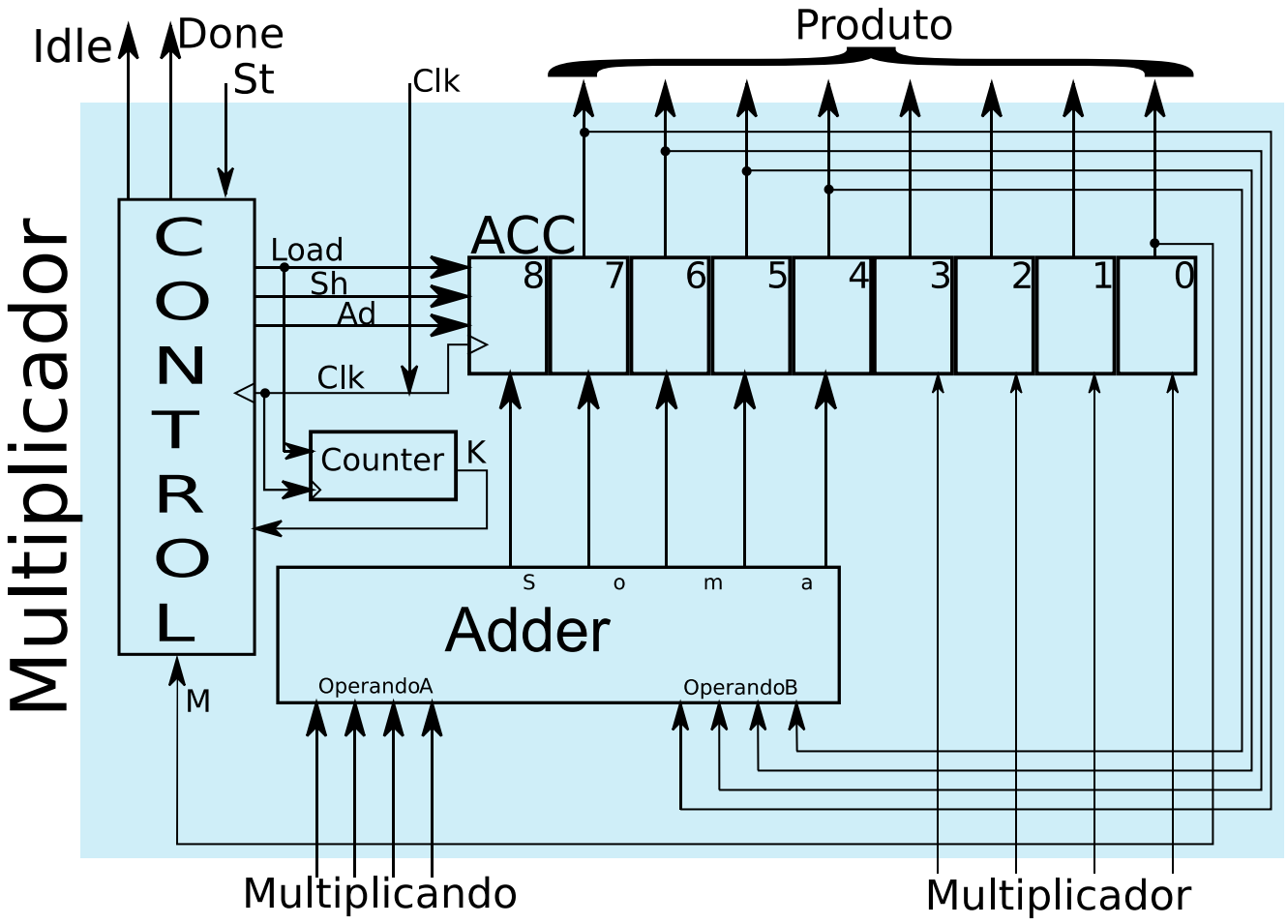
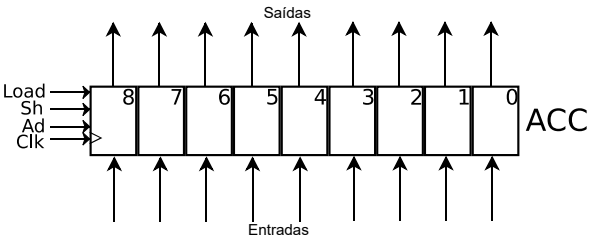
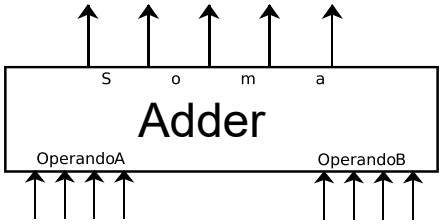


Diagrama de Blocos **Multiplicador**
Obs.: Se achar necessário, adicionar uma linha de Reset.

Um arquivo zip, chamado **Multiplicador.zip**, foi disponibilizado no site (Materiais → Lab) com a "caixa preta" de cada um dos blocos já implementada (no próximo lab vocês irão implementar o funcionamento dos mesmos). Crie um projeto para cada um destes blocos, em sua devida pasta.
Abaixo é possível visualizar os blocos, separadamente, para maior clareza de seus *ports*. Mantenha as nomenclaturas **ACC**, **CONTROL**, **Counter** e **Adder** para cada um deles. Mantenha também os nomes dos *ports* como indicados nas figuras.
Não é necessário, por ora, implementarmos o comportamento/lógica de cada bloco. É necessário, apenas, para sermos capazes de compilar o módulo *top* **Multiplicador**, que cada um dos blocos tenha seus ***ports* declarados e definidos** e que estes *ports* sejam iguais aos módulos instanciados no módulo *top* **Multiplicador**.

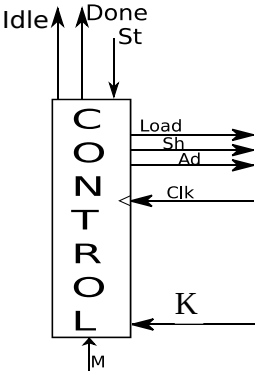


*Acumulador - **ACC***



*Somador - **Adder***

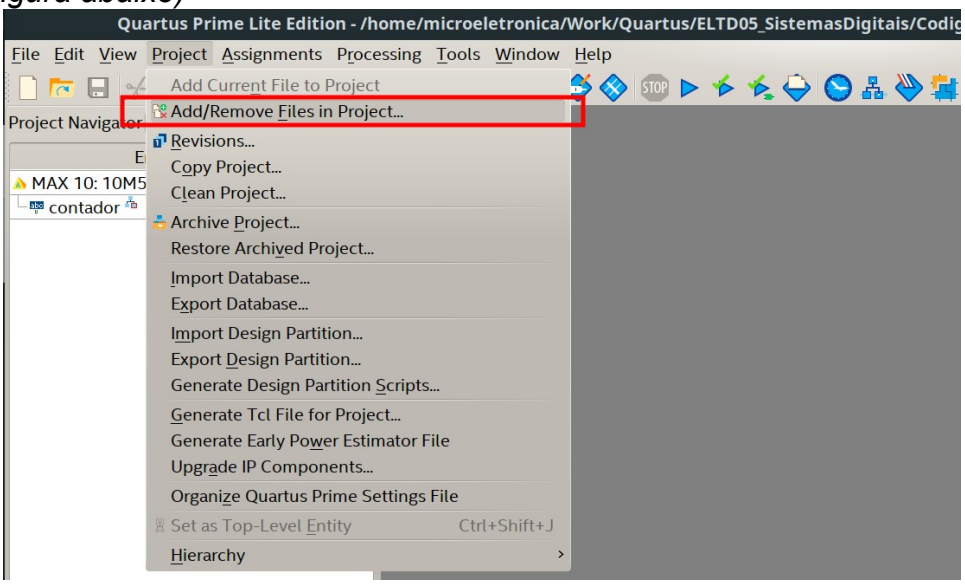
*Máquina de Estados de Controle - **CONTROL***



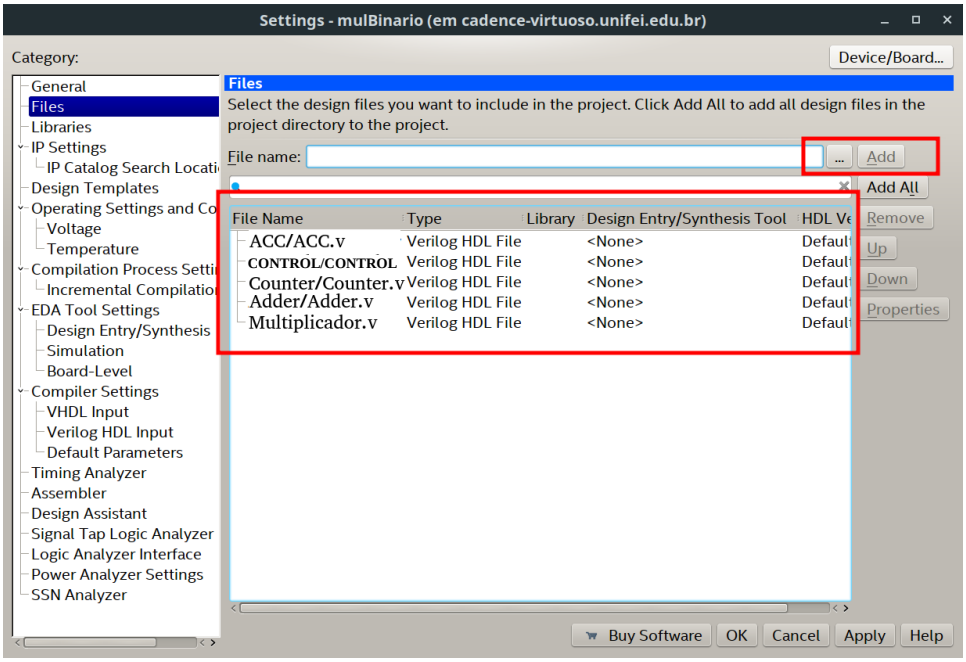
*Contador - **Counter***

3.1) Utilizando a abordagem descendente, descreva o arquivo *verilog* que representará o módulo *top* do **Multiplicador** (lembre-se que este projeto será construído na pasta raiz do arquivo zip fornecido). Compile o projeto do módulo *top* **Multiplicador**. Perceba que haverá erros na compilação. Eles se darão devido a estarmos instanciando módulos (**ACC**, **CONTROL**, **Counter** e **Adder**) que não foram previamente implementados e/ou informados onde estão suas implementações.

3.3) Com o projeto *top* **Multiplicador** aberto no Quartus, vá em ***Project, Add/Remove Files in Project…*** *(vide figura abaixo)*



e **adicione** todos os *verilogs* que implementam os blocos presentes no módulo *top* **Multiplicador** (vide figura abaixo).



3.4) Recompile o projeto Multiplicador. Se você não cometeu erros nos passos anteriores, a síntese deverá ocorrer normalmente já que, agora, o Quartus sabe onde estão as implementações dos módulos instanciados.
No entanto, como nenhum comportamento/circuito foi implementado nos *verilogs* dos blocos (**ACC**, **CONTROL**, **Counter** e **Adder**), caso simulássemos o circuito, não veríamos nada, pois só existe a "caixinha" multiplicador, com "caixinhas" dos blocos dentro. Porém, as "caixinhas" não têm implementação.

Faremos a implementação dos comportamento/circuitos dos blocos no próximo laboratório.

Este tipo de design exemplifica uma abordagem de projeto **descendente**, onde vamos descendo o nível de abstração – partimos de uma diagrama de blocos, implementamos o módulo top **Multiplicador** contendo seus subcircuitos (**ACC**, **CONTROL**, **Counter** e **Adder**) como "caixas pretas". No lab seguinte, implementaremos os circuitos/comportamentos dos blocos.

Copie e salve em algo perene para poder usar estes projetos no laboratório seguinte.