Manual de uso da biblioteca de componentes

Instruções para instalar a Biblioteca de Componentes no Active-HDL do aluno.

0) Fazer o download do arquivo "Biblioteca_de_ComponentesV4.zip" que se encontra na ferramenta "Arquivos" na pasta "Projeto" do site da disciplina;

1) Descompactar esse arquivo diretamente na pasta de seu computador onde se encontra instalado o seu Active-HDL. Normalmente essa pasta se encontra em: C:\ProgramFile\Aldec\Active-HDL 7.2SE\Vlib\

2) Abrir o programa Active-HDL sem abrir nenhuma projeto;

3) Selecionar o ícone de Bibliotecas (ícone = conjunto de livros). Nesse momento aparecerá uma barra adicional de tarefas onde existe o ícone de "Attach Library". Clique nesse ícone.

4) Localize na pasta "Biblioteca_de_Componentes" que você acabou descompactar no diretório Vlib. Navegue nessa pasta um nível abaixo e localize o arquivo "Biblioteca_De_Componentes.Lib". Dê um duplo "click" nesse arquivo.

Se estes passos forem executados corretamente a sua biblioteca de componentes estará instalada no Active-HDL e em todo novo projeto criado.
Para adicioná-la aos projetos já existentes basta fazer o passo 3 com o projeto aberto.

Descrição de alguns componentes da biblitoeca.

```
entity absminmaxsomsub is
  generic(
      NumeroBits : integer  := 8;
      Tsom :              time          := 5 ns;
      Tabs :              time          := 2 ns;
      Tsub :              time          := 5 ns;
      Tmm :               time          := 3 ns
  );
  port(
      C : in std_logic_vector(2 downto 0);
      a : in std_logic_vector(NumeroBits - 1 downto 0);
      b : in std_logic_vector(NumeroBits - 1 downto 0);
      f : out std_logic_vector(NumeroBits - 1 downto 0)
  );
end absminmaxsomsub;
```

NumeroBits: Quantidade de bits que a unidade irá utilizar.
Tsom: Tempo para uma operação de soma.
Tabs: Tempo para uma operação de absoluto.
Tsub: Tempo para uma operação de subtração.
Tmm: Tempo para uma operação de mínimo e máximo.

C: Operação que irá rodar na unidade.
000 – Valor zero na saída.
001 – Soma dos operandos “a” e “b”.
010 – Valor absoluto de “b”.
011 – Subtração dos operandos “b” e “a” (b - a).
100 – Valor absoluto de “a”.
101 – Subtração dos operandos “a” e “b” (a - b).
110 – Valor mínimo de “a” e “b”.
111 – Valor máximo de “a” e “b”.
a: Operando “a” na função escolhida com “NumeroBits” de bits.
b: Operando “a” na função escolhida com “NumeroBits” de bits.
f: Operando “a” na função escolhida com “NumeroBits” de bits.

```
entity And_NE is
  generic(
      NE: integer := 4
  );
  port(
      Sp : in std_logic_vector(NE - 1  downto 0);
      P : out std_logic
  );
end And_NE;
```

NE: Quantidade de bits que a unidade irá utilizar.
Sp: Operando com “NE” bits.
P: O AND lógico de todos bits do operando de entrada. Caso a entrada estiver com todos os bits “1” a saída é “1”, caso tenha pelo menos um “0”, é “0”.

```
entity contador is
  generic(
      NB: integer := 8;
      Tsetup: time := 2 ns;
      Tcarga: time := 5 ns;
      Tcount: time := 3 ns
  );
  port(
      C : in std_logic;
      LC : in std_logic;
      R : in std_logic;
      UD : in std_logic;
      D : in std_logic_vector(NB - 1 downto 0);
      U : out std_logic;
      Z : out std_logic;
      Q : out std_logic_vector(NB - 1 downto 0)
  );
end contador;
```

NB: Quantidade de bits que a unidade irá utilizar.
Tsetup: Tempo de setup do registrador do contador.
Tcarga: Tempo de load do registrador do contador
Tcount: Tempo de uma contagem.

C: Clock do Contador
LC:
Caso “0” carrega o valor em “D” no contador.
Caso “1” incrementa/decrementa o valor do contador.
R: Se “1” preenche o contador com “0”.
UD:
Caso “1” incrementa o valor do contador.
Caso “0” decrementa o valor do contador.
D: Valor a ser carregado no contador.
U:
Igual a “1” quando o valor do contador é igual a tudo “1”.
Igual a “0”, caso contrário.
Z:
Igual a “1” quando o valor do contador é igual a tudo “0”.
Igual a “0”, caso contrário.
Q: Valor do contador.

```
entity Decodificador is
  port(
      In0 : in std_logic_vector(2 downto 0);
      Sai : out std_logic_vector(7 downto 0)
  );
end Decodificador;
```

In0: Entrada do decodificador de 3 bits.
Sai: A saída é em função da seguinte tabela
Sai = "00000001" quando In0 = “000”
Sai = "00000010" quando In0 = “001”
Sai = "00000100" quando In0 = "010"
Sai = "00001000" quando In0 = "011"
Sai = "00010000" quando In0 = "100"
Sai = "00100000" quando In0 = "101"
Sai = "01000000" quando In0 = "110"
Sai = "10000000" quando In0 = "111"
Sai = "XXXXXXXX" quando In0 é outro valor que não seja os acima.

```
entity DualRegFile is
  generic(
      NBend: integer := 4;
      NBdado: integer := 8;
      Tread: time := 5 ns;
      Twrite: time := 5 ns
  );
  port(
      clk : in std_logic;
      we : in std_logic;
      dadoina : in std_logic_vector(NBdado - 1 downto 0);
      enda : in std_logic_vector(NBend - 1 downto 0);
      endb : in std_logic_vector(NBend - 1 downto 0);
      dadoouta : out std_logic_vector(NBdado - 1 downto 0);
      dadooutb : out std_logic_vector(NBdado - 1 downto 0)
  );
end DualRegFile;
```

NBend: Quantidade de bits de endereço.
NBdado: Quantidade de bits de dados.
Tread: Tempo de leitura no banco.
Twrite: Tempo de escrita no banco.

clk: Clock do banco de registrador
we: Habilita a escrita no banco
dadoina: Dado de entrada a ser escrito no endereço “a”.
enda: Endereço do valor a ser escrito no banco ou lido.
endb: Endereço de outro valor a ser lido no banco.
dadooutа: Dado armazenado no endereço “a”.
dadooutb: Dado armazenado no endereço “b”.

```
entity maisum is
  generic(
      NB: integer := 8;
      Tprop: time := 2 ns
  );
  port(
      I : in std_logic_vector(NB - 1 downto 0);
      O : out std_logic_vector(NB - 1 downto 0)
  );
end maisum;
```

NB: Quantidade de bits dos dados.
Tprop: Tempo de propagação da soma.

I: Valor de entrada.
O: Valor de entrada somado de 1.

```
entity multiplexador is
  generic(
      NumeroBits:integer:=8;
      Tsel : time  := 2 ns;
      Tdata : time  := 1 ns
  );
  port(
      S : in std_logic;
      I0 : in std_logic_vector(NumeroBits - 1 downto 0);
      I1 : in std_logic_vector(NumeroBits - 1 downto 0);
      O : out std_logic_vector(NumeroBits - 1 downto 0)
  );
end multiplexador;
```

NumeroBits: Quantidades de bits da entrada e saída do multiplexador.
Tsel: Tempo do seleção do multiplexador.
Tdata: Tempo de transferência de dados.

S: Sinal de seleção do multiplexador
I0: Entrada 0 com "NumeroBits" de bits para ser selecionada.
I1: Entrada 1 com "NumeroBits" de bits para ser selecionada.
O:
Caso S seja igual a '0', então a saída é o valor de I0.
Caso S seja igual a '1', então a saída é o valor de I1.

```
entity Ou_NS is
  generic(
      NP: integer := 4
  );
  port(
      Ss : in std_logic_vector(NP - 1 downto 0);
      S : out std_logic
  );
end Ou_NS;
```

NP: Quantidade de bits de entrada.

Ss: Operando com "NP" bits.
S: O OR lógico de todos bits do operando de entrada. Caso a entrada estiver com todos os bits "0" a saída é "0", caso tenha pelo menos um "1", é "1".

```
entity Ram is
  generic(
      BE : integer := 8;
      BP : integer := 16;
      NA: string := "mram.txt";
      Tz : time := 2 ns;
      Twrite : time := 5 ns;
      Tsetup : time := 2 ns;
      Tread : time  := 5 ns
  );
  port(
      Clock : in std_logic;
      rw : in std_logic;
      dadoEntrada : in std_logic_vector(BP - 1 downto 0);
      ender : in std_logic_vector(BE - 1 downto 0);
      pronto : out std_logic;
      dadoSaida : out std_logic_vector(BP - 1 downto 0)
  );
end Ram;
```

BE: Quantidade de bits do endereço da memória.
BP: Quantidade de bits de dados da memória.
NA: Nome do arquivo que contém o valor inicial da memória.
Tz: Não tem utilidade nessa versão da memória.
Twrite: Tempo de escrita da memória.
Tsetup: Tempo de setup da memória.
Tread: Tempo de leitura da memória.

Clock: Clock da memória.
rw: Caso o valor seja 1 a operação no ciclo é escrita, caso seja 0 a operação no ciclo é leitura.
dadoEntrada: Valor para ser carregado na memória.
ender: Endereço de leitura ou escrita na memória.
pronto: Sinal que avisa se o valor está pronto para ser lido no ciclo de leitura ou se foi escrito no ciclo de escrita.
dadoSaida: Valor que foi lido pela memória.

O arquivo “mram.txt” precisa estar no mesmo local que os .vhd e precisa seguir o seguinte formato.
00 8 -- Endereço inicial e número de palavras a serem carregadas.
0F0F 1A1A 2B2B 3C3C 4D4D 5E5E 6060 7878 -- Conteúdo das oito palavras a carregar.
EE N
XXX….XXX XXX….XXX XXX….XXX …. XXX….XXX XXX….XXX XXX….XXX
EE = Endereço inicial
N = Quantidade de valores XXX…XXX no arquivo
XXX...XXX = valor a ser colocado na memória em hexadecimal.

```
entity registrador is
  generic(
      NumeroBits : INTEGER := 8;
      Tprop : time := 5 ns;
      Tsetup : time := 2 ns
  );
  port(
      C : in std_logic;
      R : in std_logic;
      S : in std_logic;
      D : in std_logic_vector(NumeroBits - 1 downto 0);
      Q : out std_logic_vector(NumeroBits - 1 downto 0)
  );
end registrador;
```

NumeroBits: Quantidade de bits que o registrador armazena.
Tprop: Tempo de propagação do registrador.
Tsetup: Tempo de setup do registrador.

C: Clock do registrador.
R: Caso R=’1’, então o registrador é zerado de forma assíncrona.
S: Caso S=’1’, então o registrador é preenchido de ‘1’ de forma assíncrona.
D: Entrada a ser armazenada no registrador.
Q: Valor contido no registrador.

```
entity ROM is
  generic(
      BE : integer := 8;
      BP : integer := 16;
      Tacesso : time := 5 ns;
      NA: string  := "mrom.txt"
  );
  port(
      ender : in std_logic_vector(BE - 1 downto 0);
      dado : out std_logic_vector(BP - 1 downto 0)
  );
end ROM;
```

BE: Quantidade de bits do endereço da memória.
BP: Quantidade de bits de dados da memória.
Tacesso: Tempo de leitura da memória.
NA: Nome do arquivo que contém o valor inicial da memória.

ender: Endereço de leitura ou escrita na memória.
dado: Valor que foi lido pela memória.

O arquivo “mrom.txt” segue o mesmo formato que o “mram.txt”.
mal.

```
entity ULA is
  generic(
      NB : integer := 8;
      Tsom : time := 5 ns;
      Tsub : time := 5 ns;
      Ttrans : time := 5 ns;
      Tgate : time := 1 ns
  );
  port(
      Veum : in std_logic;
      A : in std_logic_vector(NB - 1 downto 0);
      B : in std_logic_vector(NB - 1 downto 0);
      cUla : in std_logic_vector(2 downto 0);
      Sinal : out std_logic;
      Vaum : out std_logic;
      Zero : out std_logic;
      C : out std_logic_vector(NB - 1 downto 0)
  );
end ULA;
```

NB: Quantidade de bits da ULA.
Tsom: Tempo para uma operação de soma.
Tsub: Tempo para uma operação de subtração.
Ttrans: Tempo para uma operação de copia.
Tgate: Tempo de operação lógica.

Veum: Sinal de “Vem-um”,adicionado a soma.
A: Valor de Entrada “A”.
B: Valor de Entrada “B”.
cUla: Seleciona a operação de saída.
000 - (A + Veum)
001 - (A + B + Veum)
010 - (B + Veum)
011 - (A - B + Veum)
100 - (A and B)
101 - (A or B)
110 - (A xor B)
111 - (not A)
Sinal: Sinal do operando de saída em complemento de 2.
Vaum: Sinal de “Vai-um” para concatenar com o próxima ULA.
Zero: Igual a ‘1’ quando o valor de saída é igual a zero.
C: Valor de saída da operação selecionada.