# Laboratório sobre Implementação de Sistemas Digitais com VHDL
# Multiplicação por somas sucessivas

Prática: Implementação estrutural de multiplicação por somas de produtos

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc, CAD Foundation e Ferramentas XSTools e Plataforma XS40/XST-1 da Xess, Inc.

## Parte I – Introdução e Objetivos

Laboratórios anteriores introduziram conceitos básicos sobre a utilização do ambiente de simulação Active-HDL, e investigaram o fluxo de projeto completo a partir de uma descrição inicial em linguagem de descrição de hardware (em inglês, *Hardware Description Language*, ou *HDL*), desde a captura do projeto até a sua implementação em hardware.

Este Laboratório tem os seguintes objetivos específicos:

- Implementar um multiplicador por somas sucessivas em VHDL.
- Implementar uma máquina de estados simples para o controle do multiplicador.
- Realizar o test_bench para a simulação da máquina de controle do multiplicador.
- Realizar o download do multiplicador no hardware para verificar o correto funcionamento do sistema.

As **Parte II**, **III** e **IV**, deste laboratório devem ser executadas no ambiente Active-VHDL, enquanto que a **Parte V** será realizada mediante emprego do sistema Foundation.

## Parte II – Implementação do Bloco de Dados do Módulo Multiplicador

A Figura 1 ilustra a estrutura geral de um multiplicador de dois vetores de 8 bits representando números naturais (ou seja binários puros). Dois níveis da hierarquia do projeto aparecem, correspondendo às descrições da entidade e da arquitetura em VHDL. Este é o circuito a ser implementado. Mostra-se na Figura 1 apenas um possível diagrama de blocos do hardware para esta computação (importante: não estão sendo mostrados no diagrama os sinais de inicialização e relógio dos registradores, mas estes são obviamente necessários).

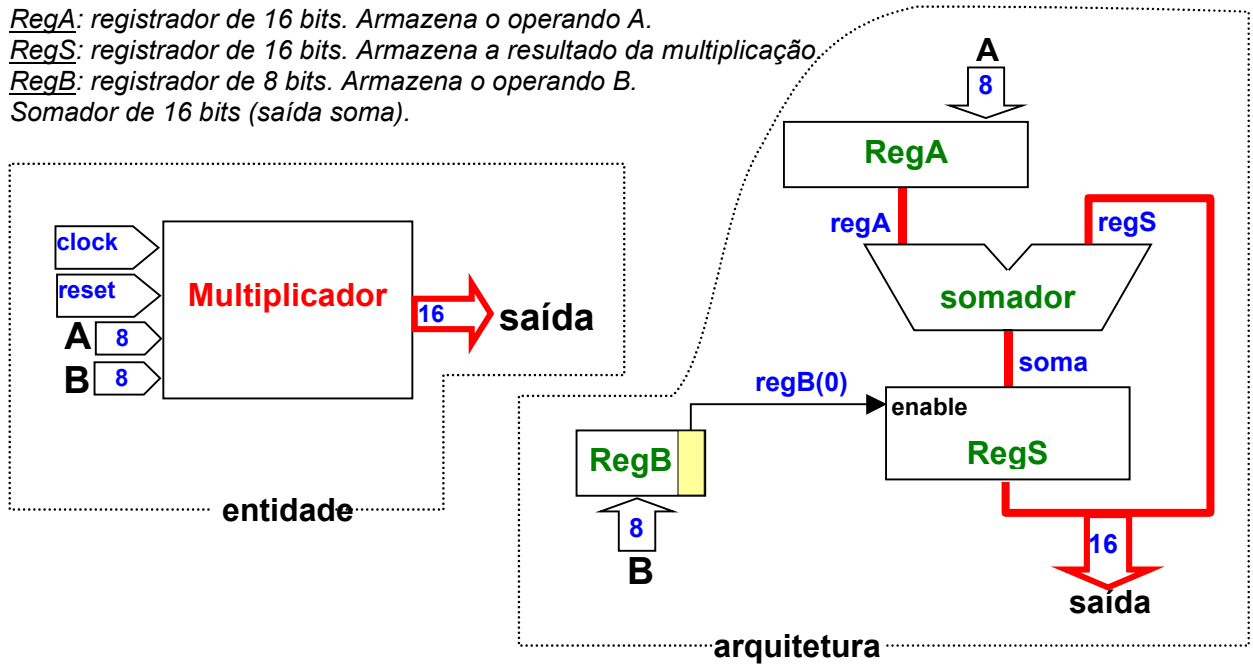



**Figura 1 – Arquitetura de um multiplicador por somas sucessivas.**

Esta versão é uma das mais simples formas de implementar um multiplicador em hardware. Para outras versões (mais eficientes) recomenda-se a leitura Seção 4.6 do livro "Organização e Projeto de Computadores - a interface hardware/software", de D. A. Patterson e J. L. Hennessy, Segunda Edição, a partir da página 144.

O funcionamento do multiplicador por somas sucessivas é o seguinte. Quando o bit menos significativo do operando B for 1 (denominado regB(0) na Figura 1) soma-se o resultado acumulado até o momento ao operando A, deslocado quantas vezes for necessário. A cada ciclo de relógio, ambos operandos são deslocados, porém em direções *contrárias* ("A" para a esquerda e "B" para a direita).

- Inicialmente o reset é aplicado, o que deve ter como efeito as seguintes ações (seu código VHDL deve garantir isto):
  - inicialização do registrador A: regA <= "00000000" & A
    (A é de 8 bits , mas regA é de 16 bits, deve-se usar concatenação em VHDL);
  - inicialização do registrador B: regB <= B;
  - reset do regS: regS <= (others=>'0');
- A cada ciclo de relógio, durante exatamente 8 ciclos:
  - *Na **subida** do relógio* armazena-se o resultado da soma se regB(0) for igual a '1';
  - *Na **descida** do relógio* desloca-se os registradores regA e regB

    dica: para um dos registradores: regA <= regA(14 downto 0) & "0";
- Como está sendo feito o procedimento, são necessários 8 ciclos de relógio após o sinal de reset para obter o resultado da multiplicação de 2 números de 8 bits.

  Faça um teste de mesa e verifique se o algoritmo funciona corretamente.

**Tarefa 1**: Implemente o multiplicador em VHDL, utilizando como modelo o circuito acumulador.

1. **Dicas:**
   - Implemente um package que contenha apenas a declaração de barramentos assim:

```
library IEEE;
use IEEE.Std_Logic_1164.all;

package mult is
 subtype reg16 is std_logic_vector(15 downto 0);
 subtype reg8 is std_logic_vector(7 downto 0);
end mult;
```

   - Desta forma, conforme a figura do multiplicador, tem-se para o somador e para a saída os seguintes comandos:

```
soma <= regA + regS;
saida <= regS;
```

Utilizando como modelo o laboratório sobre o circuito acumulador, implemente 3 processos, um relativo a cada registrador, tendo como controle os sinais de reset e clock. **Atenção:** 2 registradores são sensíveis à borda de subida do clock e 1 é sensível à borda de descida do clock. **Tarefa 2**: **Pesquise, Pense e Responda** - Porquê utilizar este procedimento?

   - A medida que for escrevendo o código VHDL, compile-o para correção dos eventuais erros (use a tecla de atalho F11).

2. Justifique o tamanho do multiplicador. **Tarefa 3**: **Pesquise, Pense e Responda** - Porquê são necessários 16 bits de saída, se os operandos A e B são de 8 bits?

## Parte III - Implementação do Bloco de Controle do Módulo Multiplicador

Como queremos prototipar o circuito na plataforma XS40/XST-1, sabe-se que não há nesta recursos para entrar com os valores de "A" e "B" ao mesmo tempo. Observando a plataforma, nota-se que existem os seguintes recursos de entrada e saída: 8 chaves do tipo "dip-switch", 2 chaves de pressão, 3 displays de sete segmentos e 8 leds. Desta forma, deve-se implementar um bloco de controle do multiplicador como um máquina de 4 estados, ativada por uma tecla de pressão, onde os estados possuem as seguintes funções:

- Estado 1: armazena o operando A, a partir do dado de entrada contido nas chaves dip-switch;
- Estado 2: armazena o operador B, a partir do dado de entrada contido nas chaves dip-switch;
- Estado 3: inicializa o multiplicador;
- Estado 4: realiza a multiplicação (resultado pronto após 8 ciclos de relógio neste estado).

O código abaixo ilustra uma possível implementação do dito bloco de controle em VHDL:

```
----------------------------------------------------------------------
-- CONTROLE DO MULTIPLICADOR
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.mult.all; -- usar o nome do package definido pelo grupo no início do projeto !!

entity ctrl_mul is
                port( data : in reg8;
                 key, ck, reset : in std_logic;
                 saida : out reg16);
end;

architecture a1 of ctrl_mul is

 component multiplicador is
                port( completar a descrição dos pinos );
 end component;

 type State_type is (S1, S2, S3, S4);
 signal EA: State_type;
 signal A,B: reg8; -- reg8 deve estar declarado no package
 signal mulreset : std_logic;
 begin

 -- maquina de estados para controlar o multiplicador
 process (reset, key)
 begin
 if reset='0'then
 EA <= S1;
 mulreset <= '0';
 elsif key'event and key='0' then
       case EA is
                when S1 => EA <= S2; A<=data;
                when S2 => EA <= S3; B<=data; mulreset <= '1';
                when S3 => EA <= S4; mulreset <= '0';
                when S4 => EA <= S4;
       end case;
 end if;
 end process;

 -- instanciacao do multiplicador
 X1: multiplicador port map( completar a descrição );

 end a1;
```

O circuito de controle pode ser visto como composto de 4 partes:

1. Declaração da entidade:

- 3 entradas tipo std_logic: *clock*, *reset* e *key*;
- 1 entrada de 8 bits: *data*, que será utilizada pelos operandos "A" e "B";
- *saída*, de 16 bits, que conterá o resultado da multiplicação.

2. Declaração do multiplicador (*component*); definição do tipo State_type, que será utilizado pela máquina de estados; e declaração das variáveis.
3. Processo responsável pela implementação do algoritmo de controle. Procedimento:
   - o *reset*='0' coloca o sistema no estado S1;
   - a primeira descida em *key* ocasiona o armazenamento de A e a passagem para o estado S2;
   - a segunda descida em *key* ocasiona o armazenamento de B, aciona o *mulreset* (*reset* do multiplicador) e a passagem para o estado S3;
   - a terceira descida em *key* ocasiona a remoção do *mulreset* e a passagem para o estado S4;
   - o sistema fica preso no estado 4 até a próxima ativação de *reset*.
4. Instanciamento do multiplicador (*port map*).

## Parte IV – Simulação do Módulo Multiplicador

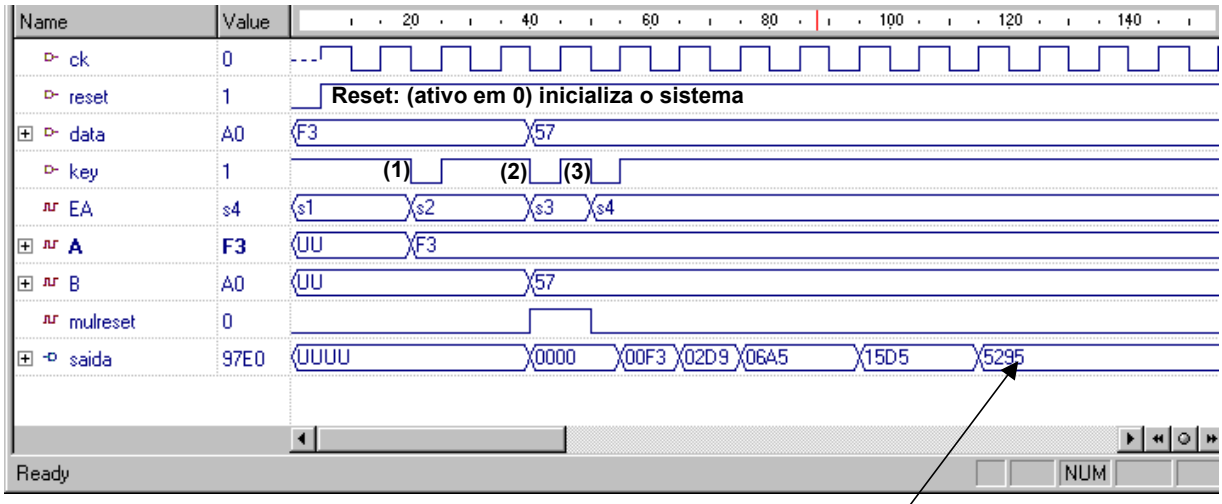
O circuito para testar o multiplicador deve controlar 4 sinais: *clock, reset, key* (chave, ativa na borda de descida) e data (dados – 8 bits). Há apenas uma saída, o valor da multiplicação através do sinal *saida*.

Uma possível implementação do *test_bench* está ilustrada abaixo:

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use work.mult.all;

entity tb is
end;

architecture rtl of tb is

 component ctrl_mul is
            port(completar a descrição dos pinos);
 end component;

 signal data : reg8;
 signal saida : reg16;
 signal ck, reset, key: std_logic ;
begin

 m1 : ctrl_mul port map(completar a descrição dos pinos);

--- fazer o processo do clock

 reset <='0', '1' after 5 ns;

 key <= '1', '0' after 20ns, '1' after 25 ns, '0' after 40ns, '1' after 45 ns,
          '0' after 50ns , '1' after 55 ns;

 data <= x"F3", x"57" after 40ns, x"67" after 450ns, x"9B" after 460ns;

end rtl;
```

**Analise o diagrama de tempos gerado por este comando**

A simulação deste test_bench está ilustrada no diagrama de tempos da Figura 2.




(1) 1ª ação da chave resulta no armazenamento de A
(2) 2ª ação da chave resulta no armazenamento de B e no reset do mult.
(3) 3ª ação da chave resulta no início da multiplicação

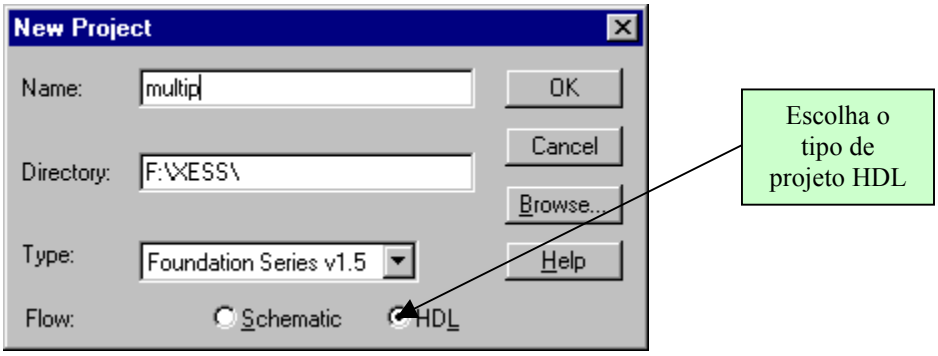
Resultado: disponível após 8 ciclos de clock
0F3H * 057H = 05295H

**Figura 2 – Diagrama de tempos da simulação do multiplicador por somas sucessivas.**

**Tarefa 4**: Modifique o test_bench para realizar 3 multiplicações, ao invés de apenas 1. Atenção: deixar ao menos 8 ciclos de clock para a realização da multiplicação.

## Parte V – Síntese , Implementação e Teste do Módulo Multiplicador

Após obter uma simulação correta (ou seja, após validar funcionalmente o multiplicador), feche o Active-VHDL e abra a ferramenta Foundation, escolhendo a opção novo projeto (ou File → new project), tendo o cuidado de especificar que o projeto a criar seja do tipo VHDL (botão HDL), conforme a janela de diálogo mostrada abaixo:




Passos a realizar para obter a versão executável em hardware:

1. Obter o arquivo multip.ucf na homepage da disciplina e substituir o seu arquivo .**ucf** original (na raiz do projeto), por este, com a definição correta dos pinos. **ATENÇÃO**: Não se esqueça de alterar o nome do arquivo, caso o seu projeto não se chame **multip**.
2. Obter o arquivo top.vhd da homepage da disciplina e o inseri-lo no diretório raiz do projeto.
3. Copiar o arquivo criado na **Parte II** (contendo package, blocos de dados e de controle do multiplicador), no diretório raiz do projeto. Para que esta descrição seja sintetizável, descartar o arquivo de test_bench.
4. Inserir os dois arquivos fontes VHDL no projeto usando a opção de menu: Document → Add do gerenciador de projeto do Foundation.

Resumindo: deve-se acrescentar 2 arquivos VHDL e um UCF na raiz do projeto.

A Figura 3 abaixo ilustra como deve parecer a janela do Foundation após a inserção dos dois arquivos VHDL e do UCF. Importante: dê duplo clique sobre o arquivo UCF e verifique se ao final do arquivo os pinos estão corretamente instanciados.

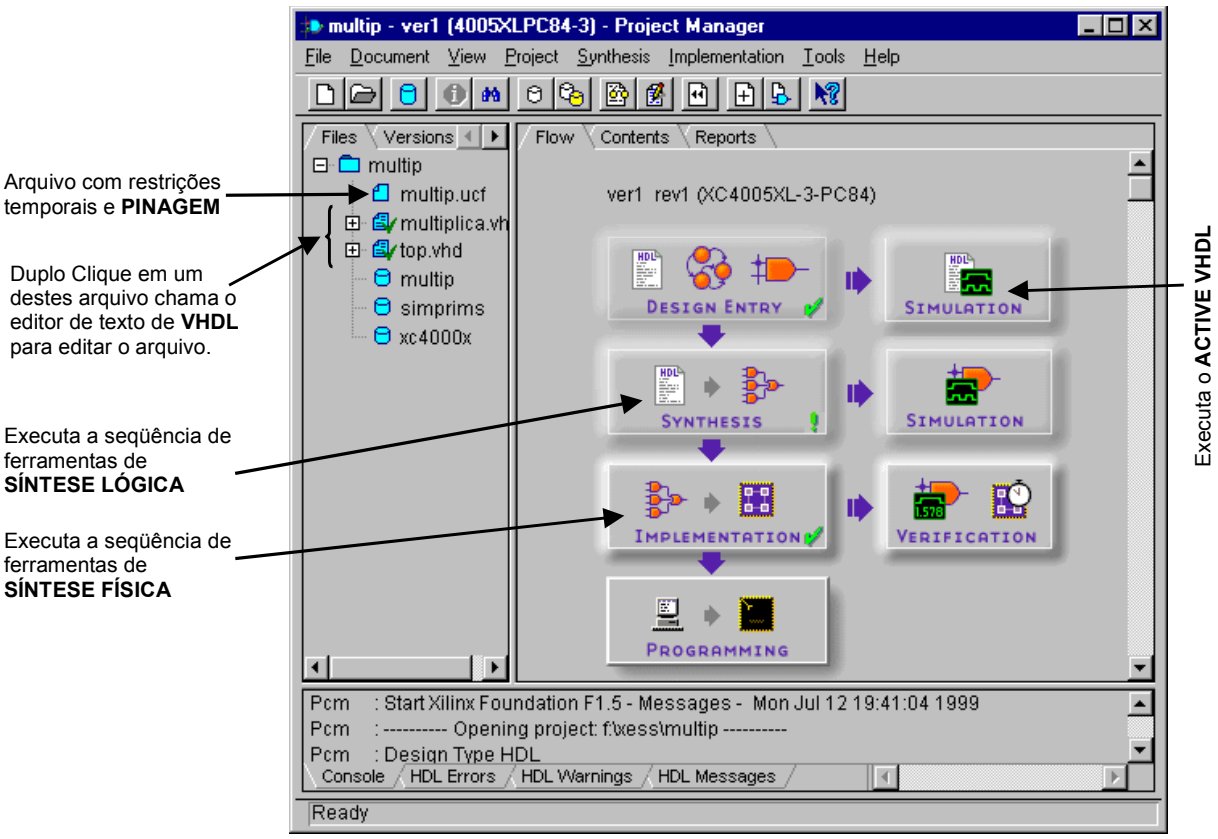
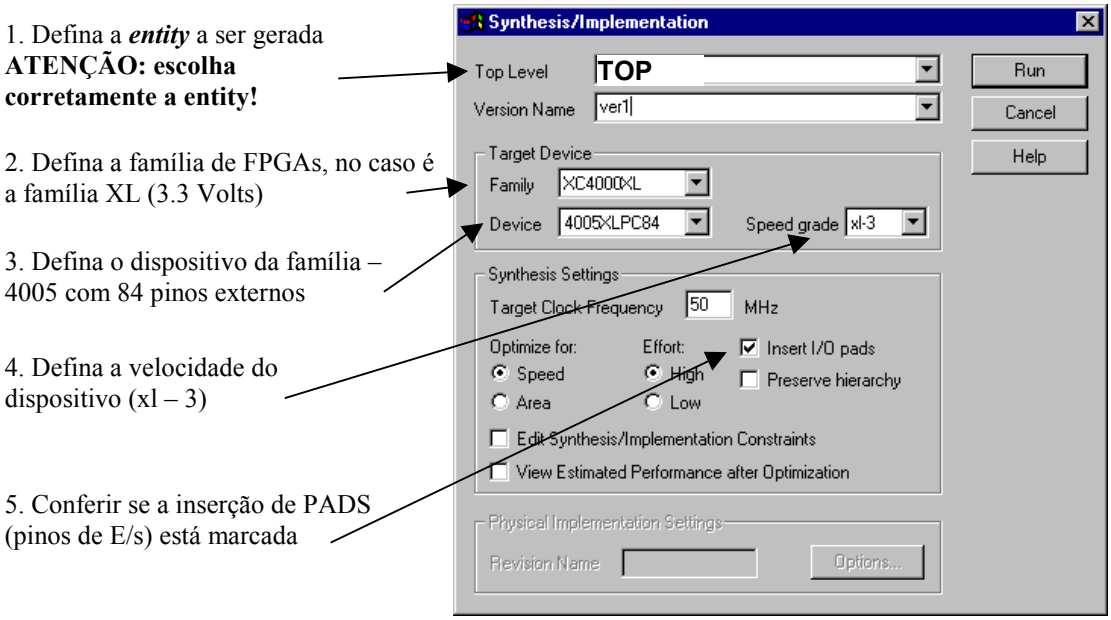



**Figura 3 – Janela do gerenciador de projetos do Foundation após criação do projeto multiplicador e inserção dos arquivos fonte em VHDL e do arquivo UCF no projeto.**

Ao Chamar a ferramenta de síntese lógica automática. Aparecerá o seguinte menu:




***ATENÇÃO: O erro mais comum é não definir a entity raiz, que no nosso casso chama-se "top".***

Execute a síntese lógica (*Run*). A caixa da ferramenta de síntese lógica conterá um "!" ao final da síntese (um *warning* apenas).

Chamar a síntese física automática. Não há nenhum parâmetro a ser inserido. Aparecerá a janela da Figura 4, com o progresso da síntese física. Observar as diversas etapas: mapeamento,

posicionamento e roteamento, análise de *timing* (cálculo automático do atraso no caminho mais longo no circuito – crítico) e geração do arquivo de download (extensão **.bit**).

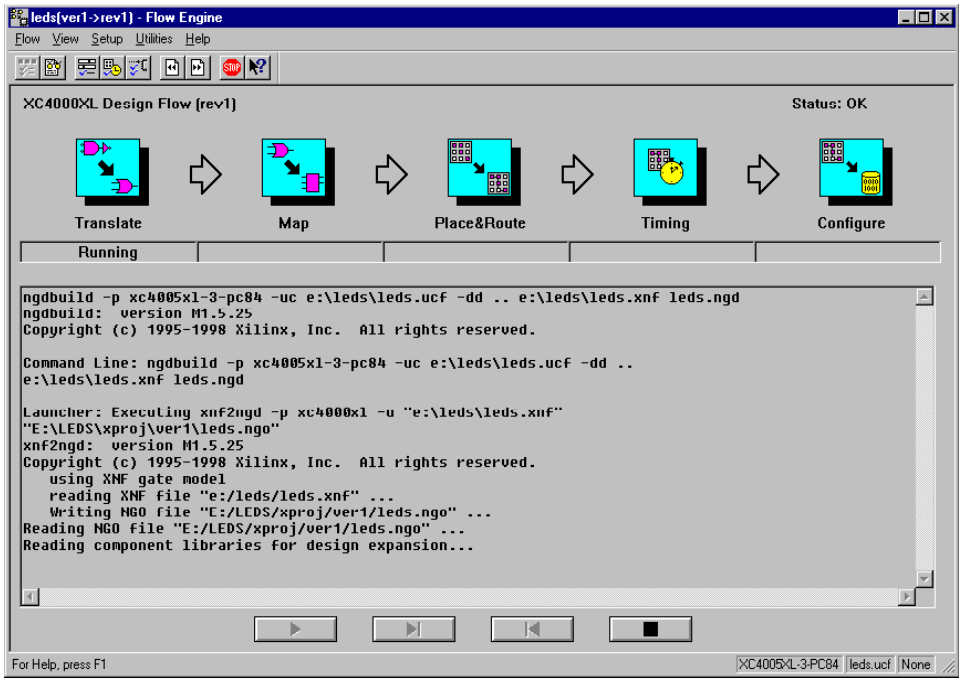



**Figura 4 – Janela que mostra o andamento da síntese física.**

Agora, analise os relatórios gerados pela tarefas de síntese, seja a síntese lógica (ou independente de tecnologia), seja a síntese física (também chamada de dependente de tecnologia). Existe no software Foundation uma orelha denominada Reports, na mesma sub-janela da janela de fluxo de projeto. Lá existem vários relatórios (ou diretórios de relatórios).

Observar, em "implementation report files" o relatório "post layout timing report", que durante a análise de caminho crítico, é indicada uma freqüência máxima de funcionamento de 34 MHz.

Certifque-se que a plataforma XS40/XST-1 de sua bancada esteja devidamente conectada ao computador via cabo paralelo, e que a plataforma está eletricamente alimentada. A seguir, abra o programa "gxsload", e arraste o arquivo "multip.bit" (que se encontra na raiz do projeto) para a janela do gxsload. Esta ação realiza o download do circuito na placa.

***Se tudo der certo, pronto! Estamos com o algoritmo descrito em VHDL implementado em hardware !***

Como utilizar o multiplicador:

1. Pressione *reset* (tecla do meio), até que todos os leds acendam.
2. Selecione o operando "A", através das dip-switch (um número binário de 8 bits).
3. Pressione *spare* (tecla da esquerda). Apenas os 4 leds da direita permanecem acesos.
4. Selecione o operando "B", através das dip-switch (outro número binário de 8 bits).
5. Pressione *spare*. Todos os leds apagam, e o display exibe 00. Significa multiplicador inicializado.
6. Pressione *spare* pela terceira vez. O resultado da multiplicação deve ser então exibido. Os 8 bits mais significativos são exibidos nos leds e os 8 bits menos significativos são exibidos nos displays 7 segmentos, como dois dígitos em hexadecimal.

**Tarefa 5**: Examine o layout do circuito gerado, através da ferramenta FPGA Design Editor, em *Tools → Implementation → FPGA Editor*.

**Tarefa 6**: Confira no "*implementation report files*" o relatório "*map report*". Lá está relatado que foram utilizados X CLBs (Y% do FPGA). **Pesquise, Pense e Responda**: Qual o valor de X e Y no seu projeto? A Figura 5 mostra um exemplo de resultado obtido (não necessariamente igual para todos).

Design Summary
---
Number of errors: 0
Number of warnings: 1
Number of CLBs: 82 out of 196 41%
CLB Flip Flops: 59
CLB Latches: 0
4 input LUTs: 153
3 input LUTs: 15 (5 used as route-throughs)
Number of bonded IOBs: 34 out of 65 52%
IOB Flops: 8
IOB Latches: 0
Number of clock IOB pads:1 out of 12 8%
Number of BUFGLSs: 3 out of 8 37%
Number of oscillators: 1
Total equivalent gate count for design: 1470
Additional JTAG gate count for IOBs: 1632

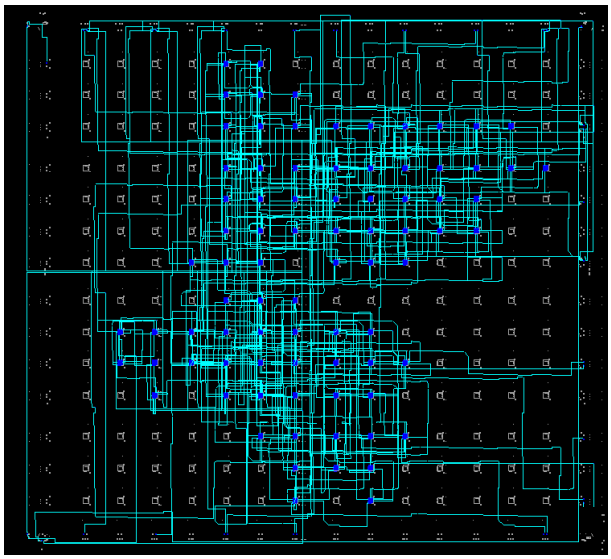


**Figura 5 – Resultados da síntese física, numéricos e gráficos.**

**Tarefa 7**: Examine o arquivo ucf, analisando a relação dos pinos na entity e neste arquivo.

**Tarefa 8**: Escolha a opção de menu Tools → Simulation/Verification → Interactive Timing Analyzer". Nesta ferramenta, escolha a opção de menu Analyze → Advanced Design. Ao final deste relatório há um resumo da análise de timing do circuito implementado.

## Parte VI - A Fazer e a Entregar

- O projeto completo compactado pelo Foundation (opção de menu Design → Archive Design... do Ambiente Active-HDL), com formas de onda salvas em arquivo dentro do projeto;
- Relatório com as respostas às questões aqui colocadas dentro das **Tarefas** via menção **Pesquise, Pense e Responda**;
- Mostrar o projeto funcionando ao professor até o início da próxima aula.

Percentuais de nota atribuídos às Atividades:

| Item Avaliado | Percentual |
|---|---|
| **Projeto completo e simulações** | **50%** |
| **Demonstração do projeto, funcional ou não, ao professor, em aula** | **20%** |
| **Relatório com respostas às questões e tarefas realizadas** | **30%** |