

Apostila

Implementações com FPGAs Descrição de Hardware

Autor: Eng. Wagner Rambo*

Área: Sistemas Digitais

Nível: Intermediário

*Engenheiro em Eletrônica, Computadores e Telecomunicações



www.wrkits.com.br

Copyright WR Kits 2015 (todos os direitos reservados): Proibida reprodução total ou parcial sem prévia autorização por escrito do autor. Copyright protegido pela Lei de Direitos Autorais LEI N° 9.610, de 19 de Fevereiro de 1998.

Sumário

INTRODUÇÃO	3
1. FPGAs.....	4
2. VHDL	6
3. VERILOG.....	9
4. NÍVEIS DE ABSTRAÇÃO	11
4.1. Domínio Comportamental.....	12
4.2. Domínio Estrutural.....	12
4.3. Domínio Físico.....	12
5. criando um novo projeto no quartus II	13
6. descrevendo uma latch em vhdl	21
7. descrevendo uma latch em verilog	30
8. descrevendo uma latch com block diagram/schematic.....	34
9. compilação, análise e síntese de um projeto	38
10. simulando um projeto	41
11. definindo os pinos físicos a serem utilizados.....	48
12. conversão do arquivo .sof em .jic	50
13. gravando a memória externa epcs4.....	56
14. teste prático do circuito latch	61
CONCLUSÃO	65
ANEXOS	66

INTRODUÇÃO

A presente apostila tem por objetivo introduzir o leitor no universo dos FPGAs (*Field Programmable Gate Array*), ou em português, matrizes de portas programáveis em campo. A Ross Freeman, um dos fundadores da Xilinx Inc., é atribuída a invenção desta classe de dispositivos lógicos, que foi lançada no ano de 1985.

Este material tem ênfase no kit didático EE02-SOQ, produzido pelo Instituto de Tecnologia Emerson Martins e que pode ser adquirido pelo site professoremersonmartins.com.br. Ao final da leitura desta apostila, o projetista estará apto para utilizar o kit, onde será abordado um projeto com exemplo de descrição de hardware em diagrama esquemático, na linguagem de descrição VHDL e na linguagem de descrição Verilog.

Além de aprender a criar o projeto nas linguagens supracitadas, no compilador Quartus II da empresa Altera, o usuário vai aprender o passo a passo para conversão do arquivo com extensão .sof para .jic, e a gravação do arquivo convertido na memória externa EPICS4 presente na placa de desenvolvimento.

O kit é baseado no FPGA da família Cyclone II modelo EP2C5T144C8N, de 144 pinos, apresenta regulador de tensão (3,3V) e cristal de 20MHz. O gravador utilizado é o USB Blaster, cujos drives estão presentes nos arquivos do Quartus II.

1. FPGAs

São dispositivos lógicos que, como o próprio nome sugere, consistem em matrizes de portas programáveis. Entretanto, deve-se ter em mente, que o termo “programável” está mais para “configurável”. Existe uma grande diferença entre FPGAs e processadores. Enquanto que os processadores, mais especificamente os microcontroladores, são dispositivos programáveis, cujos algoritmos e instruções são executados de forma sequencial (não existe nada rodando paralelamente) os FPGAs, por sua vez, consistem em um hardware totalmente configurável pelo usuário. Em outras palavras, o hardware que o usuário descreve, é de fato idealizado no FPGA.

Os FPGAs são largamente utilizados na indústria atual para o projeto e desenvolvimento de circuitos integrados, até mesmo processadores. Na Figura 1, pode-se observar a representação de uma célula de um FPGA.

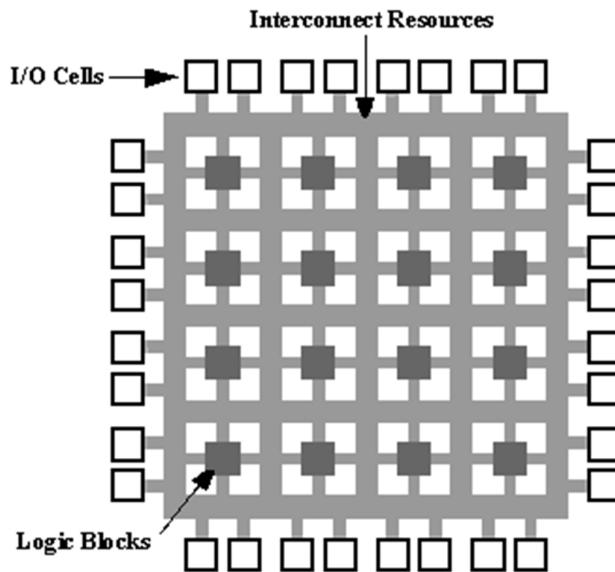


Figura 1: Representação da célula de um FPGA.

A disposição dos blocos lógicos no formato de matriz, facilita a configuração do dispositivo, para o circuito idealizado pelo projetista, via descrição de hardware. Através das interconexões matriciais, este hardware se configura, e as operações

ocorrem de forma paralela, tal como se o usuário houvesse montado este circuito em uma matriz de contatos (Figura 2).

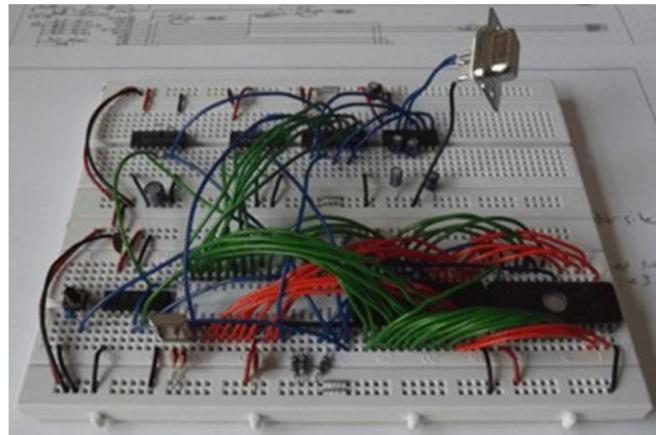


Figura 2: Circuitos digitais montados em matriz de contatos (protoboard).

Logo, além de utilizar um FPGA para o projeto de um circuito integrado, que pode conter um número muito elevado de blocos lógicos (número esse limitado pela quantidade de células do modelo do FPGA utilizado), o projetista pode implementar circuitos de teste que normalmente levariam horas realizando as conexões entre circuitos integrados discretos em uma protoboard.

Em um FPGA existem três módulos que são blocos de entrada e saída, blocos lógicos configuráveis e uma matriz de chaves de interconexão. Esses blocos lógicos consistem em circuitos combinacionais e sequenciais, que compreendem portas lógicas e flip-flops. A interconexão entre esses blocos, é realizada através de um arquivo binário, o *bitstream*. Na maioria dos FPGAs, este arquivo tem que ser recarregado após o corte de energia e, por este motivo, os kits de desenvolvimento em geral apresentam memórias externas como é o caso do EE02-SOQ, ênfase da presente apostila.

2. VHDL

Como um modo claro de documentar os projetos desenvolvidos sobre circuitos integrados de alta velocidade VHSIC (*Very High Speed Integrated Circuit*) em meados da década de 1980, mais precisamente no ano de 1983, o Departamento de Defesa dos Estados Unidos desenvolveu a linguagem de descrição de hardware conhecida como VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) que posteriormente foi padronizada pelo Instituto de Engenheiros Elétricos e Eletrônicos (IEEE), que o adotou como padrão (Standard 1076) em 1987. O padrão IEEE 1064 e IEEE 1076.3 foram propostos com o objetivo de introduzir facilidades à linguagem, sendo o primeiro contendo o pacote “Std_logic_1164” e o segundo os pacotes “Numeric_std” e “Numeric_bit”. Pacote (package), consiste em um local para armazenamento de informações de uso comum como tipos de dados, funções, etc.

A linguagem VHDL permite a descrição da estrutura de projeto que pode ser decomposta em elementos secundários e como estes são interconectados, além de permitir a especificação das funções dos elementos usando uma forma familiar de linguagem de programação. Analogamente às demais linguagens de programação, o VHDL também obedece alguns padrões de estruturação. Mas, deve-se salientar o fato que o código irá descrever um circuito digital, o seu comportamento e a maneira pela qual serão tratados os sinais envolvidos. Para exemplificar, tomemos como base o circuito da Figura 3.

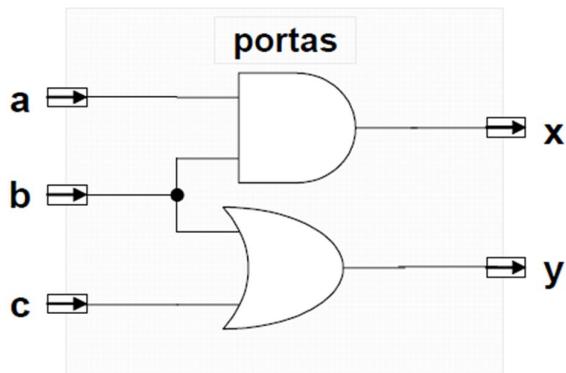


Figura 3: Exemplo de circuito combinacional.

O circuito combinacional da Figura 3 consiste em uma porta lógica AND de duas entradas e uma porta OR de duas entradas, ligadas conforme o mesmo diagrama esquemático. Na linguagem VHDL descreve-se este circuito em duas seções, sendo a primeira conhecida como entidade (entity) que pode ser vista como um nível mais alto de abstração, sendo um bloco único onde apresentam-se as entradas e saídas, conforme Figura 4.

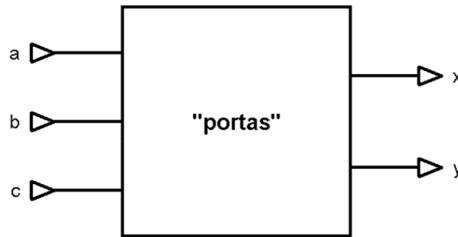


Figura 4: Entidade de um circuito.

A segunda seção consiste na arquitetura do projeto. Esta arquitetura é o exato comportamento do hardware, ou seja, deve-se descrever as funções lógicas conforme o diagrama esquemático do circuito que se pretende implementar. Um exemplo de código em VHDL para o circuito da Figura 3 é apresentado no Box 1.

```
--  
-- Exemplo de Código em VHDL (circuito Figura 3)  
--  
-- Autor: Eng. Wagner Rambo  
--  
-- 2015  
  
entity portas is port  
(  
    a      :  in  bit;  
    b      :  in  bit;  
    c      :  in  bit;  
    x      :  out bit;  
    y      :  out bit  
);  
end portas;  
  
architecture hardware of portas is  
begin  
    x <= a AND b;  
    y <= c OR b;  
end hardware;
```

Box 1: Código em VHDL para o circuito da Figura 3.

No código, tudo que estiver após os dois traços “--” é visto como comentário pelo compilador, sendo ignorado. Os comentários servem para orientar o desenvolvedor quanto a função de cada linha do código, assim como em qualquer outra linguagem de programação. Primeiro declarou-se a entidade, com o nome “portas” e em seguida suas respectivas entradas e saídas.

Após, na arquitetura de nome “hardware”, após o comando “begin”, definiu-se o comportamento do circuito, com as operações lógicas “AND” e “OR” das respectivas entradas e saídas. Não está exemplificado no código do Box 1 a inclusão de bibliotecas, que será explanada nos exemplos de projetos desta apostila, assim como outros detalhes da sintaxe do VHDL.

3. VERILOG

A linguagem Verilog é outra forma de descrição de hardware criada entre 1983 e 1984 por Prabhu Goel e Phil Moorby, para a Automated Integrated Design Systems, que mais tarde virou a Gateway Design Automation. A empresa foi comprada em 1990 pela Cadence Design Systems, que passou a ser detentora dos direitos sobre as linguagens de descrição Verilog e Verilog-XL.

Observando o sucesso da linguagem VHDL, a empresa Cadence tornou a linguagem Verilog aberta à padronização. O Instituto de Engenheiros Elétricos e Eletrônicos a tornou o padrão Standard 1364-1995, conhecido como Verilog-95. Recebeu novas atualizações para melhorias em 2001 e 2005 (IEEE Standard 1364-2001 e IEEE Standard 1364-2005, respectivamente).

A principal característica de um projeto em Verilog se dá na separação hierárquica de módulos que contém conexões e registradores. O comportamento desses módulos se dá por processos sequenciais e paralelos. Dentro de blocos “begin/end” são executados processos sequenciais. Todos os demais processos são executados de forma paralela, assim como em VHDL. O Box 2 traz o código em Verilog do mesmo circuito da Figura 3.

```
/*
Exemplo de Código em Verilog (circuito Figura 3)

Autor: Eng. Wagner Rambo

2015

*/
module portas(a,b,c,x,y);           //Declaração do módulo
  input a,b,c;                      //Entradas digitais
  output x,y;                       //Saídas digitais
  assign x = a & b;                 //x recebe a E b
  assign y = a | c;                 //y recebe a OU b
endmodule                           //Final do módulo
```

Box 2: Código em Verilog, para o circuito da Figura 3.

Os comentários em Verilog são exatamente como na Linguagem C padrão ANSI, onde duas barras “//” comenta-se uma única linha, e pode-se comentar mais linhas de forma concomitante abrindo com barra-asterisco e fechando com asterisco-barra “/* */”. Ao invés de declararmos uma entidade, declaramos um módulo com o nome “portas” e as entradas e saídas são declaradas com os comandos “input” e “output” respectivamente.

A descrição do comportamento do circuito é após o comando “assign”, que seria similar à arquitetura no VHDL. Mais detalhes sobre Verilog serão abordados no exemplo de projeto desta apostila.

4. NÍVEIS DE ABSTRAÇÃO

Os três níveis de abstração básicos de uma linguagem de alto nível são a Modelagem de projetos eletrônicos complexos (especificações); a Simulação dos modelos de componentes e a Síntese Lógica, que consiste na conversão da linguagem de alto nível VHDL e geração de uma lista otimizada de Portas Lógicas e Registradores (RTL) que então pode ser implementada (Síntese Física). Na Figura 5 pode-se observar os níveis de abstração de um sistema digital em um diagrama conhecido como Diagrama Y (Gajski e Kuhn, 1993).

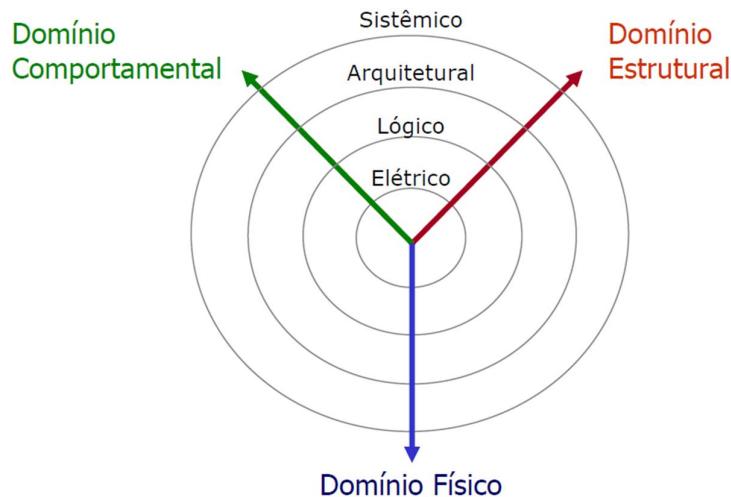


Figura 5: Níveis de abstração de um sistema digital (Diagrama Y).

O Diagrama Y representa o processo de projeto de sistemas digitais, onde os círculos representam os domínios de descrição e os eixos radiais correspondem a domínios de descrição. As intersecções dos círculos com os eixos representam descrições de projeto.

No Nível Sistêmico, o sistema digital é descrito como um conjunto de algoritmos e módulos capazes de executar estes algoritmos, que podem ou não ter um mapeamento direto para um hardware existente. Exemplos: processadores e memórias.

No Nível Arquitetural os módulos do sistema digital são componentes complexos da teoria de circuitos digitais e a forma de descrevê-los: registradores, decodificadores, unidades lógicas aritméticas, multiplexadores.

No Nível Lógico, o sistema digital é descrito a partir de noções elementares da teoria de circuitos digitais: portas lógicas, flip-flops, equações Booleanas, grafos de dados.

O Nível Elétrico é onde os modelos dos componentes provém da teoria de circuitos elétricos e eletrônicos: transistores, resistores, capacitores, etc.

4.1. Domínio Comportamental

Também pode ser chamado de Domínio Funcional, envolve descrições com informações sobre o comportamento do sistema sem se preocupar em como tal comportamento pode ser obtido, seja do ponto de vista físico, seja do ponto de vista estrutural.

4.2. Domínio Estrutural

Contém as descrições com informações sobre como interconectar blocos de base de comportamento do sistema conhecido para realizar certa funcionalidade, sem se preocupar com a disposição física destes no sistema, ou seja, representando apenas a topologia do sistema.

4.3. Domínio Físico

Pode ser definido como Domínio Geométrico, contém as descrições com informação geométrica sobre os componentes/módulos e/ou sobre a disposição espacial destes no sistema a ser fabricado.

5. CRIANDO UM NOVO PROJETO NO QUARTUS II

Para efetuar o download da versão correta do software Quartus II, siga as instruções do vídeo “Dica de Software #18”, disponibilizado pelo WR Kits Channel no seguinte link: https://www.youtube.com/watch?v=XYQgjl7_C6E

Apesar de ser um processo relativamente demorado, a instalação do Quartus II é simples e não será detalhada nesta apostila, pois basta ir clicando em “next” nas janelas que seguirão, selecionando também a pasta destino para instalação, algo totalmente intuitivo.

Passaremos agora a descrever o processo de criação de um novo projeto no software compilador Quartus II. Ao abrir o compilador a primeira janela que você enxergará é a da Figura 6. No caso estamos utilizando a versão 13.0sp1, pois é a mais recente que ainda suporta os FPGAs da família Cyclone II, modelo este presente no kit de desenvolvimento EE02-SOQ.

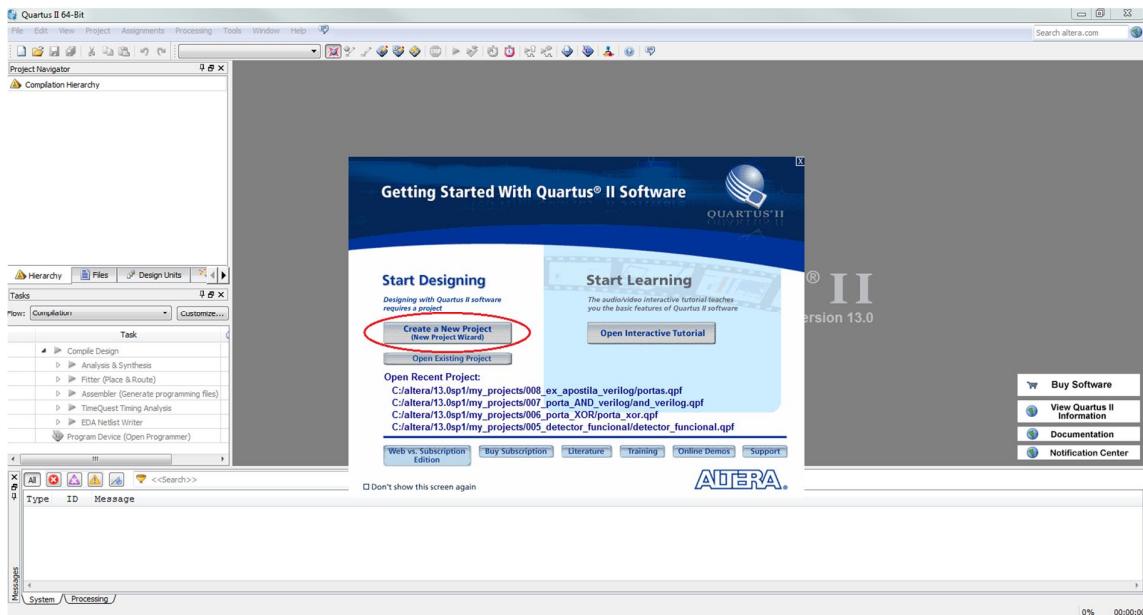


Figura 6: Tela inicial do Quartus II 13.0sp1.

Clique no botão “Create a New Project (New Project Wizard)”, conforme assinalado na Figura 6. Na janela de introdução clique em Next (Figura 7).

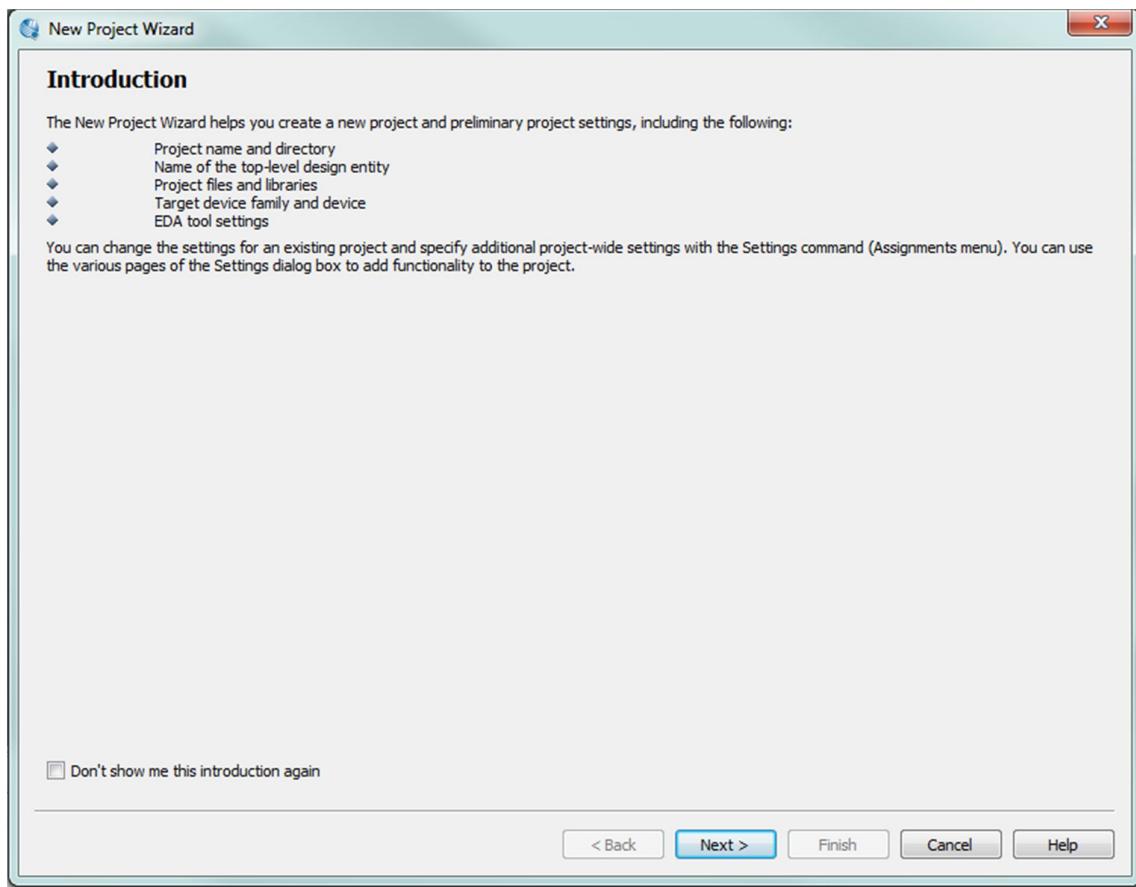


Figura 7: Iniciando a criação de um novo projeto.

Na próxima janela, você irá selecionar a pasta de destino para o seu projeto, clicando no primeiro botão de três pontos “...” (Figura 8).

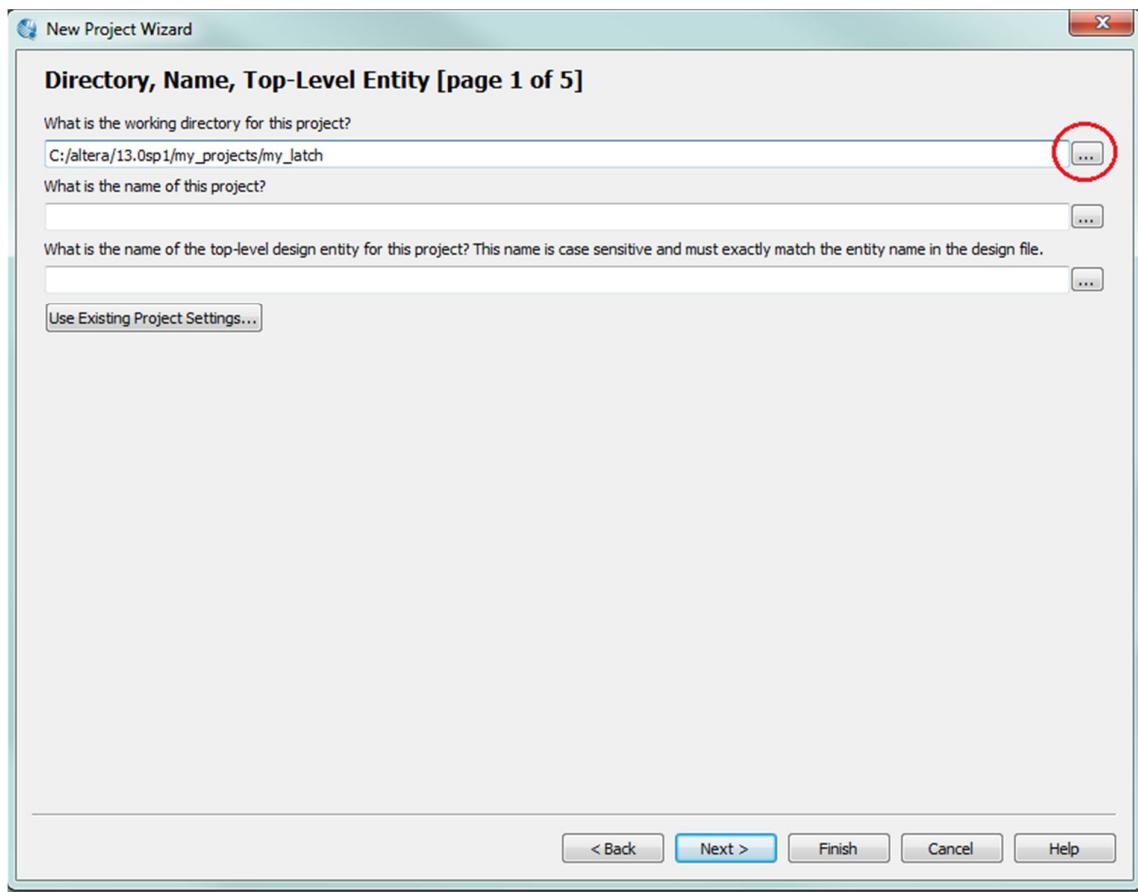


Figura 8: Pasta de destino do novo projeto.

Olhando para a Figura 8, já estamos com a pasta de destino selecionada que tem o nome de “my_latch”. O ideal é que você sempre crie uma pasta exclusiva para cada projeto, evitando possíveis erros que possam ocorrer, devido arquivos de projetos diferentes encontrarem-se em uma mesma pasta. Ainda na mesma janela, você escreve no segundo campo o nome do seu projeto. O que você escrever no segundo campo, será replicado automaticamente no terceiro campo. Optamos pelo mesmo nome da pasta destino “my_latch”, pois consiste no hardware que iremos sintetizar no compilador e efetuar a comprovação prática de funcionamento mais tarde nesta mesma apostila. Após escrever o nome clique em “Next”. A próxima janela que abrir será para adicionar arquivos ao projeto (Figura 9).

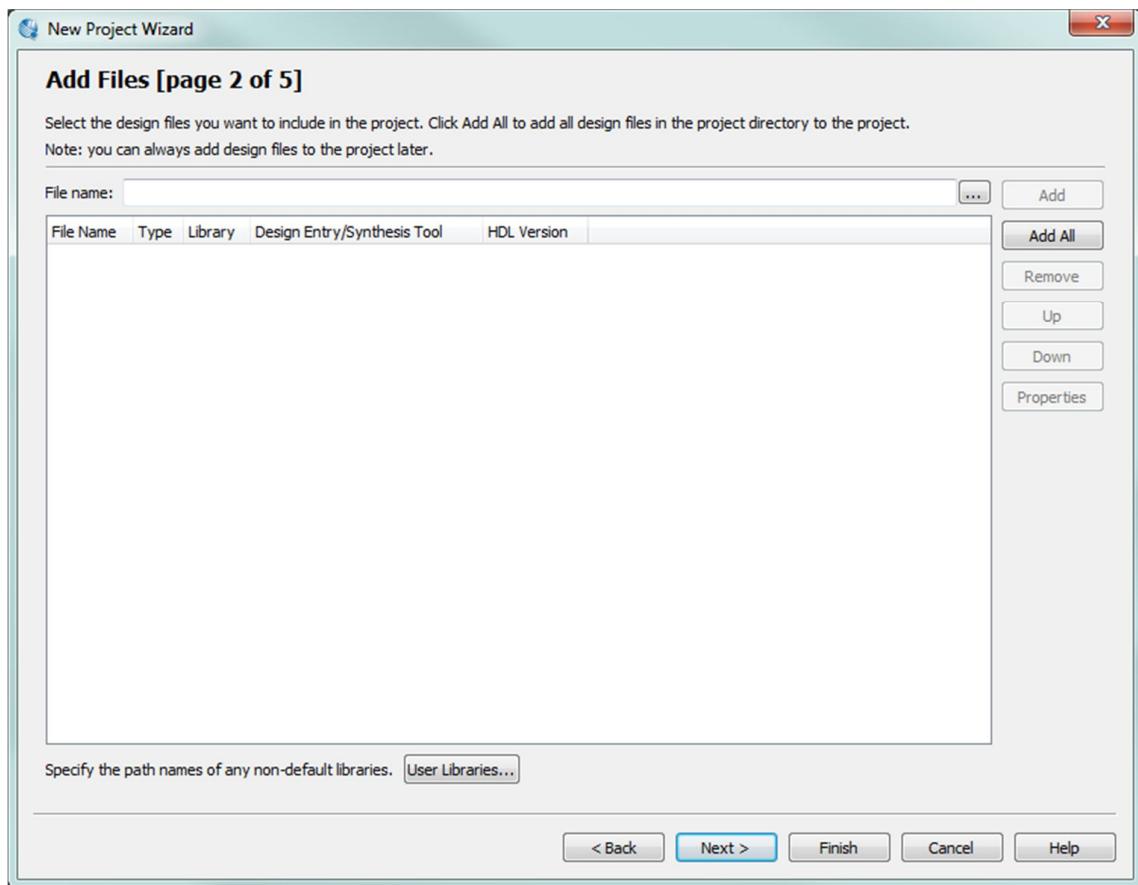


Figura 9: Adição de arquivos externos ao projeto.

Como não será adicionado arquivo algum, clique em Next. A janela exibida na Figura 10 é para selecionarmos o FPGA a ser configurado no projeto. Como o kit de desenvolvimento EE02-SOQ apresenta um FPGA da família Cyclone II, modelo EP2C5T144C8N, configure a sua janela como a imagem. Lógico que se tiver um outro kit baseado em outro modelo de FPGA, deverá configurá-la para tal.

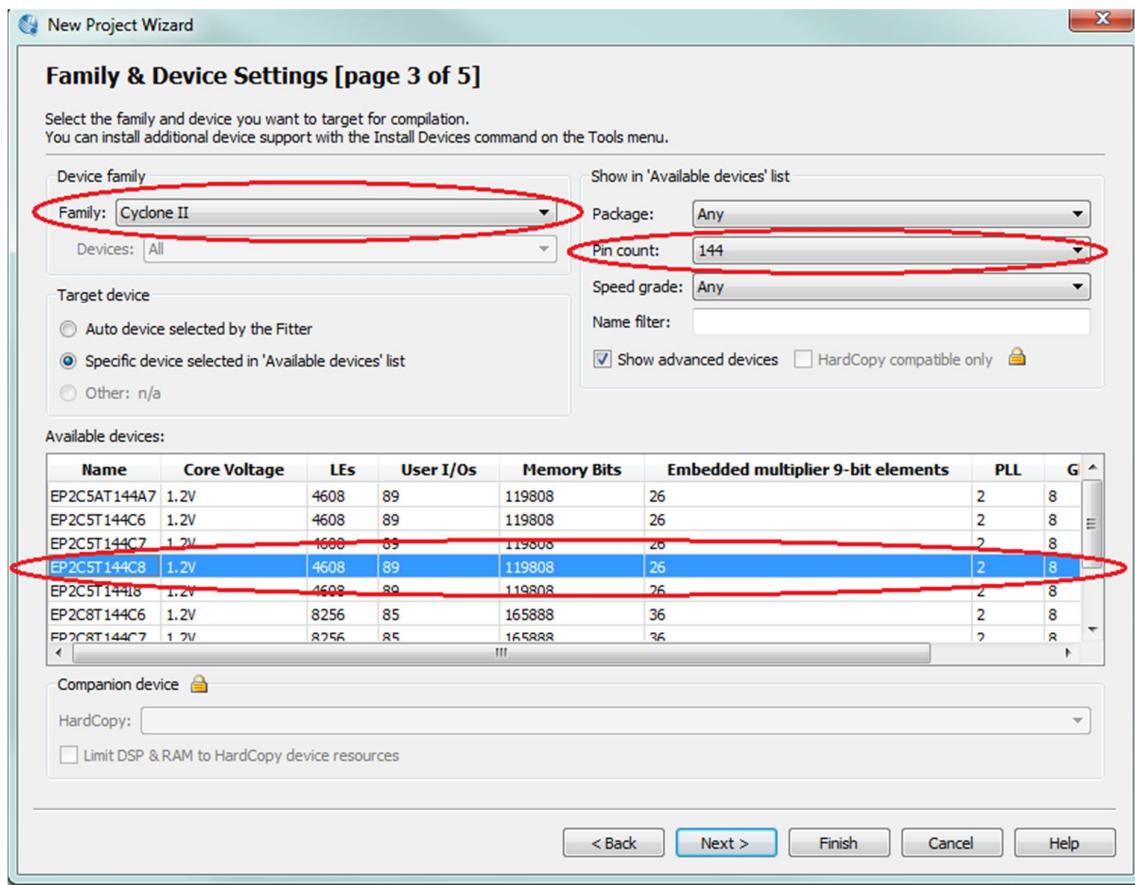


Figura 10: Selecionando o dispositivo do seu kit de desenvolvimento.

Selecione a família, o número de pinos e o modelo do FPGA presente no seu kit e clique em Next. A janela seguinte serve para especificar outras ferramentas a serem utilizadas no desenvolvimento do seu projeto, como não utilizaremos nenhuma, apenas clicaremos em Next (Figura 11).

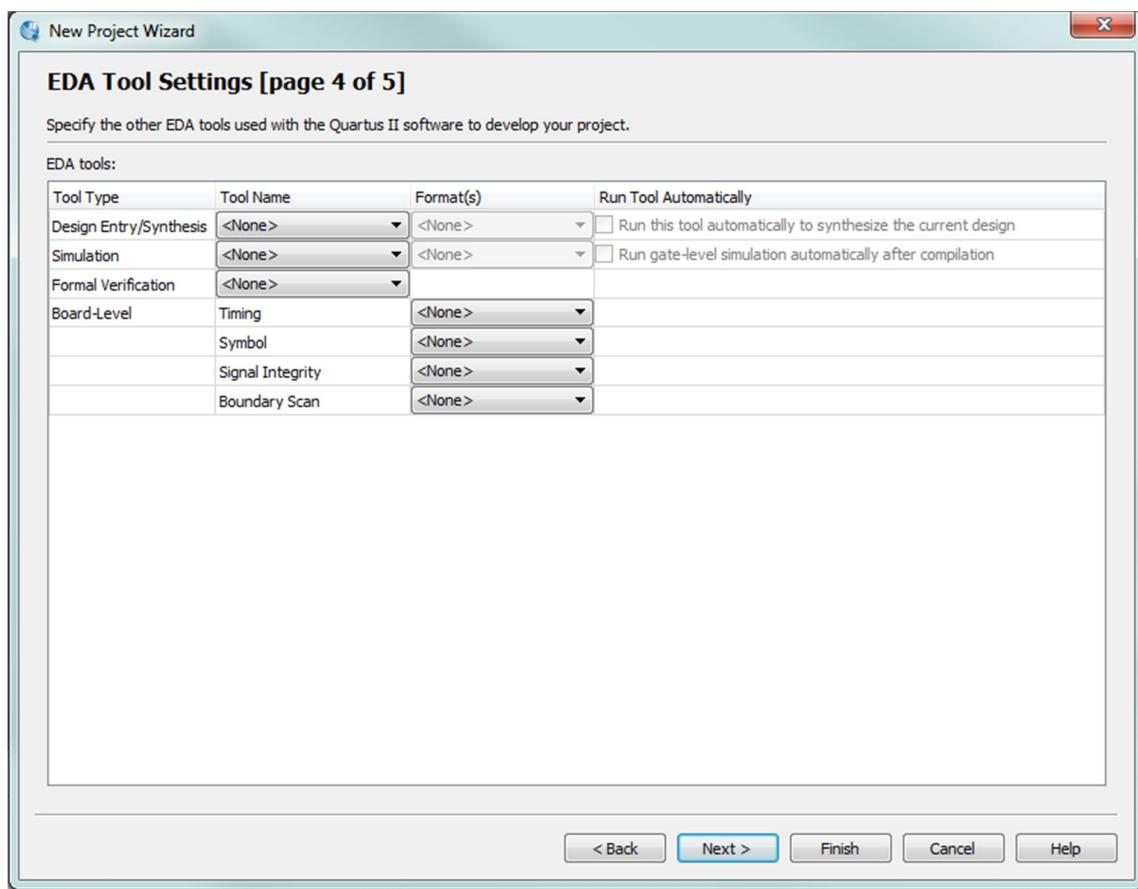


Figura 11: Adição de outras ferramentas no implemento do seu projeto.

Na próxima janela, o compilador apresenta um resumo de todas as configurações feitas anteriormente por você. Revise se está tudo conforme desejado e clique em Finish, para encerrar o processo de criação (Figura 12).

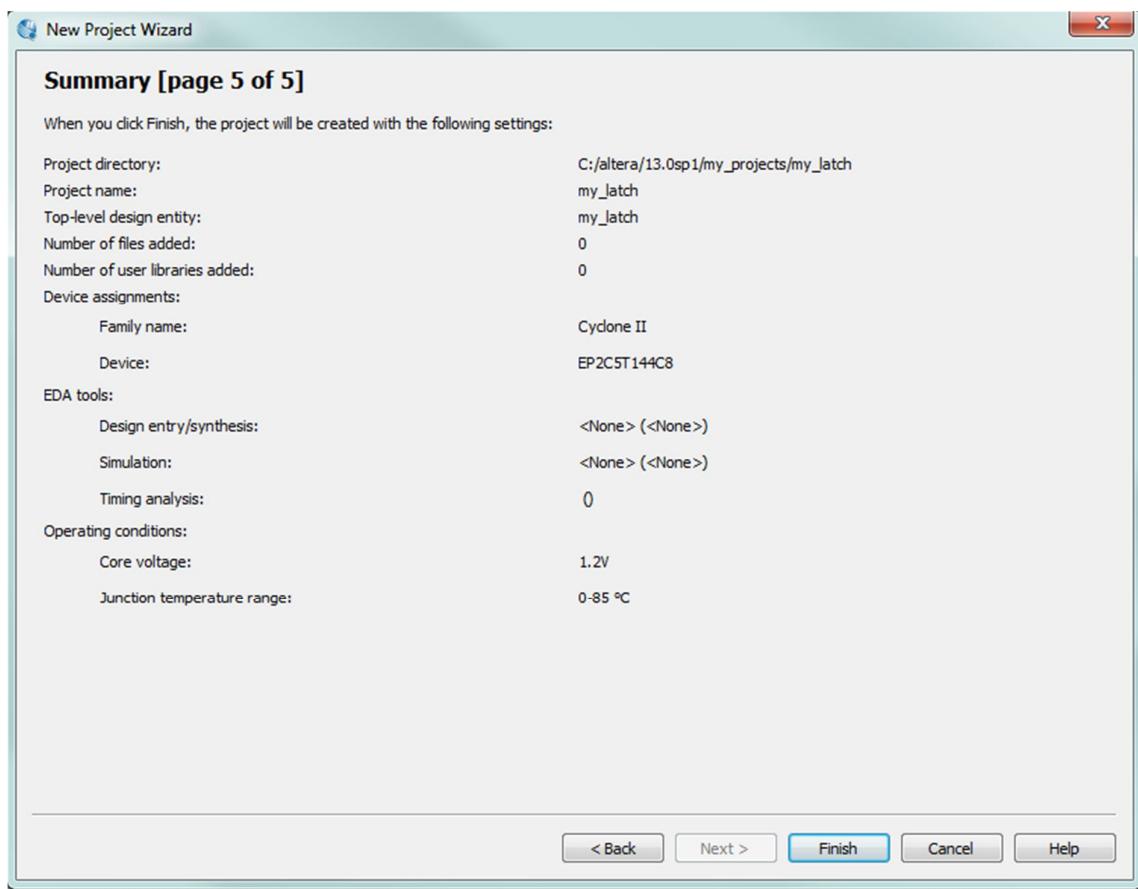


Figura 12: Resumo do novo projeto criado.

Com isso, pode-se observar no canto superior esquerdo do compilador Quartus II o seu projeto novo, incluído (Figura 13). No caso, nomeamos ele como “my_latch”, pois é justamente este circuito de exemplo que será implementado.



Figura 13: Projeto incluído no compilador no campo “Entity”.

Observe que o campo “Entity”, onde o nome de seu projeto está inserido, consiste na entidade mencionada no capítulo 2 desta apostila, sobre VHDL. Conforme dito no início da apostila, iremos demonstrar este projeto sugerido de três formas diferentes, que são a descrição em VHDL, a descrição em Verilog e a descrição em diagrama de circuito propriamente dito, para podermos comparar os métodos.

6. DESCREVENDO UMA LATCH EM VHDL

Antes de iniciarmos a descrição da Latch em VHDL, vamos inicialmente entender como este dispositivo funciona, a nível de hardware. Uma latch consiste no flip-flop mais básico que existe, também conhecido como flip-flop SR e é o dispositivo que originou os circuitos sequenciais. Chama-se “SR” por apresentar uma entrada de Set e outra de Reset, conforme Figura 14.

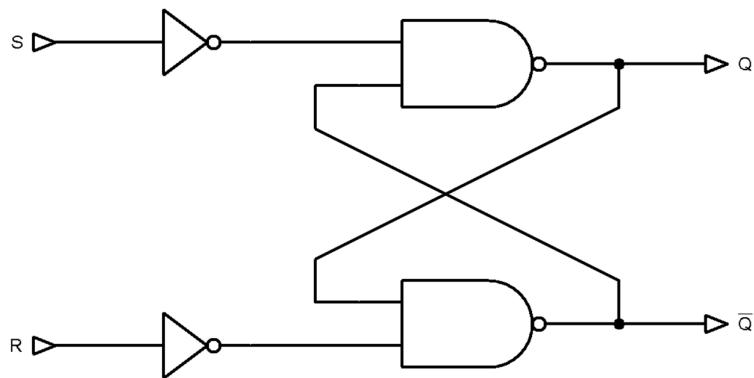
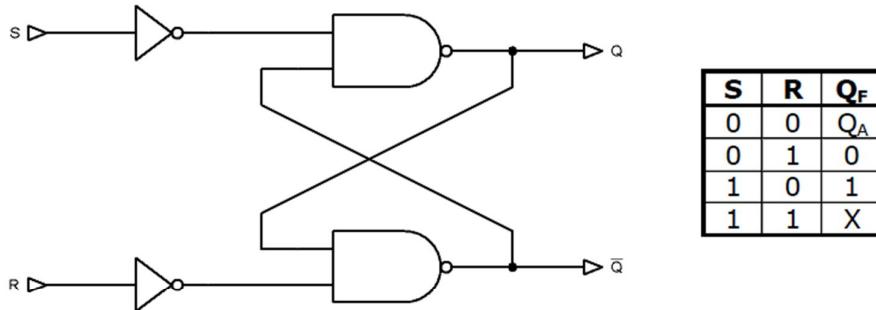


Figura 14: Flip-Flop SR (Latch).

Devido a realimentação, as saídas são injetadas juntamente com as variáveis de entrada. Logo, a saída dependerá, também, destes estados. É interessante observar, e é uma característica clássica dos flip-flops, que existe um inversor natural, que consiste no complemento da saída Q , saída \bar{Q} . Como uma é complemento da outra, obviamente seus níveis lógicos sempre serão opostos. O circuito apresentado na Figura 14 é composto por quatro portas lógicas, sendo dois inversores e duas portas NAND. Na Figura 15, pode-se observar a tabela verdade de uma latch e seu respectivo comportamento.



S = Set - quando em nível 1, Q=1

R = Reset - quando em nível 1, Q=0

Q_A = valor atual da saída Q

Q_F = valor futuro da saída Q

Figura 15: Tabela verdade e comportamento de uma latch.

Observando-se o diagrama esquemático e a tabela da Figura 15, conclui-se que, havendo nível lógico alto na entrada de Set, a saída Q terá nível lógico alto. Havendo nível lógico alto na entrada de Reset, a saída Q terá nível lógico baixo. Ambas as entradas com nível lógico baixo, a saída Q apresentará o estado atual, e isso já a caracteriza como um dispositivo de memória. Ambas as entradas em nível alto não é permitido, visto que a saída apresentará um estado indeterminado. Outro ponto que vale salientar deste flip-flop é que não existe um clock o controlando.

Para mais detalhes você pode conferir a vídeo aula do WR Kits Channel que explica o latch, disponível no link <https://www.youtube.com/watch?v=4ifBIRtl65A>.

Existe ainda uma outra forma de construir um flip-flop SR, utilizando-se apenas duas portas lógicas NOR, conforme Figura 16.

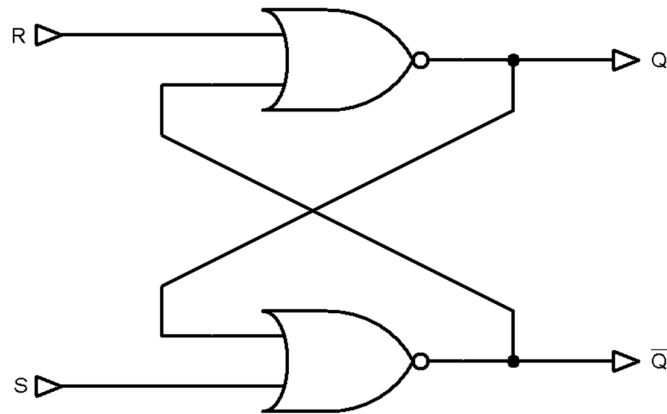


Figura 16: Flip-Flop SR com portas OR.

E este circuito apresentado na Figura 16 apresenta a mesma tabela verdade do circuito da Figura 14, logo, terá o mesmo comportamento.

Para a descrição de hardware, nos basearemos neste circuito da Figura 16, por apresentar um diagrama simplificado. Exemplos de códigos para outros dispositivos estarão disponíveis nos anexos desta documentação.

Vamos então criar um novo código em VHDL, utilizando o projeto que configuramos no capítulo 5. Para criar um novo arquivo clique em “File” e depois “New”, ou simplesmente no ícone em branco, canto superior esquerdo, destacado na Figura 17. Ainda como opção, pode utilizar o atalho “Ctrl + N” no teclado.



Figura 17: Criando um novo arquivo para o seu projeto.

Na janela que aparecer clique na opção “VHDL File” e após clique em OK (Figura 18).

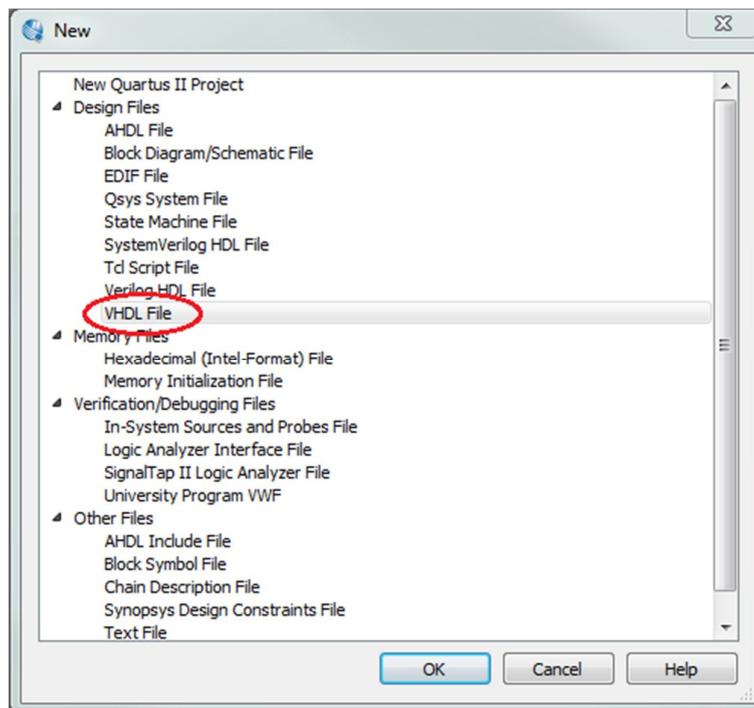


Figura 18: Selecionando a opção de novo arquivo em VHDL.

Antes de mais nada, clique em “File”, depois “Save As” e salve este arquivo com o mesmo nome utilizado para o seu projeto, no caso “my_latch” em extensão .vhd, na mesma pasta em que salvou o projeto criado. Veja detalhadamente na Figura 19.

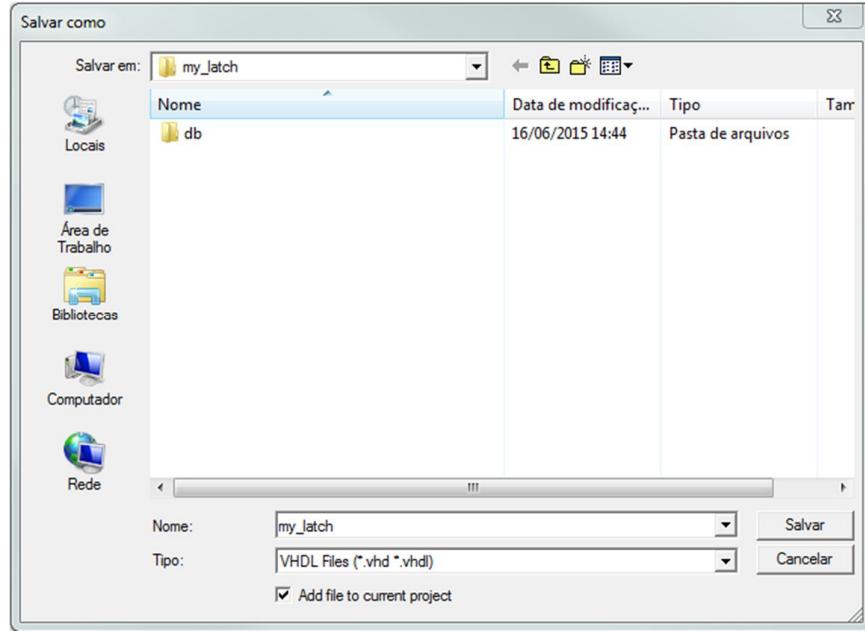


Figura 19: Pasta de destino do seu arquivo VHDL.

Estando com o projeto e o arquivo devidamente salvos, passaremos a realizar a descrição de hardware, em linguagem VHDL, do circuito latch proposto da Figura 16. O cabeçalho é o início de todo código. Nele vão informações relevantes como breve descrição do que o código faz, entradas e saídas utilizadas, nome do autor e data de criação. Conforme mencionado no capítulo 2 desta apostila, os comentários na linguagem VHDL são com dois traços “- -”, logo, o cabeçalho deverá ser feito utilizando os dois traços. A seguir no Box 3, veja como elaboramos o cabeçalho para o projeto da latch.

```
--  
-- Flip-Flop SR (Latch) descrito em VHDL  
--  
-- Entradas: S (Set) e R (Reset)  
--  
-- Saídas: Q e Qn  
--  
-- Autor: Eng. Wagner Rambo  
--  
-- Data: Junho de 2015  
--
```

Box 3: Cabeçalho do código em VHDL.

Aqui, apenas uma sugestão, você pode incluir o máximo de informações que quiser no cabeçalho do seu projeto, um resumo do que o código faz, o nome da

entidade e arquitetura, qualquer informação que irá facilitar posterior leitura e interpretação de seu código, isso será de grande ajuda em códigos maiores.

A seguir, no Box 4, veja como inserir as bibliotecas necessárias ao seu projeto.

```
library IEEE;           -- biblioteca IEEE
use IEEE.std_logic_1164.all; -- utilizado padrão lógico 1164
```

Box 4: Inserindo bibliotecas do projeto.

O comando “library IEEE;” informa a inserção da biblioteca do Instituto de Engenheiros Elétricos e Eletrônicos. Após, com o comando “use IEEE.std_logic_1164.all;” é informada a inserção do padrão lógico 1164, que consiste em um pacote inserido na linguagem VHDL, conforme explanado no capítulo 2, para introduzir facilidades no desenvolvimento dos códigos.

Também no capítulo 2, explanou-se sobre a entidade do projeto, que consiste no nível mais alto de abstração, onde declaram-se as entradas e saídas utilizadas em seu hardware. Veja a declaração da entidade e todas entradas e saídas da latch no Box 5.

```
entity my_latch is port          -- declaração da entidade
(
    S : in      std_logic;  -- entrada Set
    R : in      std_logic;  -- entrada Reset
    Q : inout   std_logic;  -- entrada/saída Q
    Qn : out     std_logic  -- saída Q negado
);
end my_latch;                  -- fim da declaração
```

Box 5: Entidade do projeto latch.

Com o comando “entity” você declara a entidade. É importante salientar que o nome da entidade, deverá ter o mesmo nome do seu projeto criado, do contrário ocorrerão erros. Por este motivo o nome de nossa entidade na declaração é “my_latch”, que consiste no nome dado originalmente para o projeto proposto. Após os comandos “is port” inicia-se a declaração de entradas e saídas.

O Set e o Reset foram declarados com os nomes “S” e “R”, respectivamente e ambos são entradas. O que os definem como entradas no código é o “in”. O que os

definem como entradas digitais é o “std_logic”. A saída \bar{Q} é declarada com o nome de “Qn”, seguida de dois pontos “.” e o “std_logic”, que a denota como digital.

Uma peculiaridade ocorre neste projeto. Observe que a saída Q do latch foi declarada como bidirecional, com o comando “inout”. Isto é necessário porque esta saída também é uma das entradas da segunda porta lógico OU. Mas porque Qn também não foi declarada como bidirecional? Entenderemos isso na descrição da arquitetura do projeto, evidenciada no Box 6.

```
-- declaração da arquitetura
architecture hardware of my_latch is
signal notQ: std_logic;      -- sinal /Q

begin                      -- início do hardware

    Q    <= R NOR notQ;      -- Q recebe R não-ou /Q
    notQ <= S NOR Q;        -- /Q recebe S não-ou Q
    Qn   <= notQ;           -- Qn recebe notQ

end hardware;               -- final do hardware
```

Box 6: Arquitetura da latch.

Com o comando “architecture” declara-se a arquitetura pertinente ao projeto, sendo que você pode nomeá-la da forma que lhe convir. No exemplo, nomeamos a arquitetura como “hardware”. Em uma tradução livre, pode-se dizer que a declaração diz “Arquitetura hardware da entidade my_latch é”, veja que temos que inserir o nome da entidade (my_latch, no caso), na declaração da arquitetura. As palavras “of” e “is” completam a declaração. Na linha seguinte, declaramos um sinal em nosso circuito com o comando “signal”. Aí está o porquê de Qn ter sido declarada na entidade apenas como saída. Pois o sinal utilizado tem o nome de $notQ$. Sempre que existe algum ramo ou nó em nosso circuito podemos o estar declarando como um sinal. O comando “begin” designa o início do comportamento do hardware e a informação paralela que descreve o comportamento do circuito.

Quando queremos realizar uma atribuição em VHDL, utilizamos o sinal de menor-igual “ $<=$ ”. Entretanto nesse caso o sinal de menor deve ser entendido como uma seta que aponta para o bit que recebe a atribuição. Inicia-se a descrição com o comando “ $Q <= R \text{ NOR } notQ;$ ”, isto é, o bit bidirecional Q recebe a operação lógica

NÃO-OU (NOR) entre a entrada R (Reset) e o sinal declarado *notQ*. Em seguida, escreve-se “*notQ* <= S NOR Q;”, que significa dizer que o sinal *notQ* recebe a operação lógica NOR entre a entrada S (Set) e o bit bidirecional *Q*. Perceba que com estas duas linhas, nosso circuito está descrito em VHDL. Para que possamos testar na prática o sinal *notQ*, precisamos declarar uma saída, por isso que *Qn* é declarado como saída na entidade, para utilizá-lo como bit auxiliar. Por isto que atribui-se diretamente o sinal *notQ* a esta saída auxiliar *Qn*, com o comando “*Qn* <= *notQ*;”. E vale salientar que se escrevêssemos essas linhas em qualquer ordem, iria funcionar do mesmo jeito pois toda essa informação se dá de forma paralela. A descrição da arquitetura se encerra com a linha “end hardware;”, onde hardware é o nome dado inicialmente para arquitetura.

O código completo em VHDL para o circuito latch pode ser visto no Box 7.

```
--  
-- Flip-Flop SR (Latch) descrito em VHDL  
--  
-- Entradas: S (Set) e R (Reset)  
--  
-- Saídas: Q e Qn  
--  
-- Autor: Eng. Wagner Rambo  
--  
-- Data: Junho de 2015  
--  
  
library IEEE;                      -- biblioteca IEEE  
use IEEE.std_logic_1164.all;        -- utilizado padrão lógico 1164  
  
entity my_latch is port           -- declaração da entidade  
(  
    S : in  std_logic;  -- entrada Set  
    R : in  std_logic;  -- entrada Reset  
    Q : inout std_logic; -- entrada/saida Q  
    Qn : out  std_logic -- saida Q negado  
);  
end my_latch;                      -- fim da declaração  
  
-- declaração da arquitetura  
architecture hardware of my_latch is  
signal notQ: std_logic;            -- sinal /Q  
  
begin  
    -- início do hardware  
  
    Q    <= R NOR notQ;           -- Q recebe R não-ou /Q  
    notQ <= S NOR Q;              -- /Q recebe S não-ou Q  
    Qn   <= notQ;                 -- Qn recebe notQ  
  
end hardware;                      -- final do hardware
```

Box 7: Circuito latch da Figura 16 descrito em VHDL.

A próxima etapa consistirá em efetuar a compilação, análise e síntese do nosso projeto, para verificar possíveis erros de sintaxe em nosso código, o que culminaria no mau funcionamento do circuito na prática.

Após uma verificação completa, comprovando-se que o código está sintaticamente correto, pode-se proceder à simulação do mesmo, utilizando as ferramentas do Quartus II destinadas a este propósito.

Entretanto, todas estas etapas serão explanadas em detalhe nos capítulos mais adiante desta apostila, pois inicialmente demonstraremos como descrever o circuito da Figura 16 na linguagem Verilog e também como efetuar sua montagem utilizando a ferramenta de diagramas esquemáticos do Quartus II.

7. DESCREVENDO UMA LATCH EM VERILOG

No capítulo anterior, demonstramos como descrever um circuito latch na linguagem VHDL e também apresentamos o funcionamento detalhado do circuito com sua respectiva tabela verdade. No presente capítulo, veremos como efetuar a descrição do mesmo circuito utilizando a linguagem Verilog. Iremos aproveitar o mesmo projeto. Feche o arquivo VHDL e clique em “File > New”, ou diretamente no ícone de página em branco do canto superior esquerdo, ou ainda com o atalho “Ctrl + N” no teclado. Selecione a opção “Verilog HDL File” e clique em “Ok” (Figura 20).

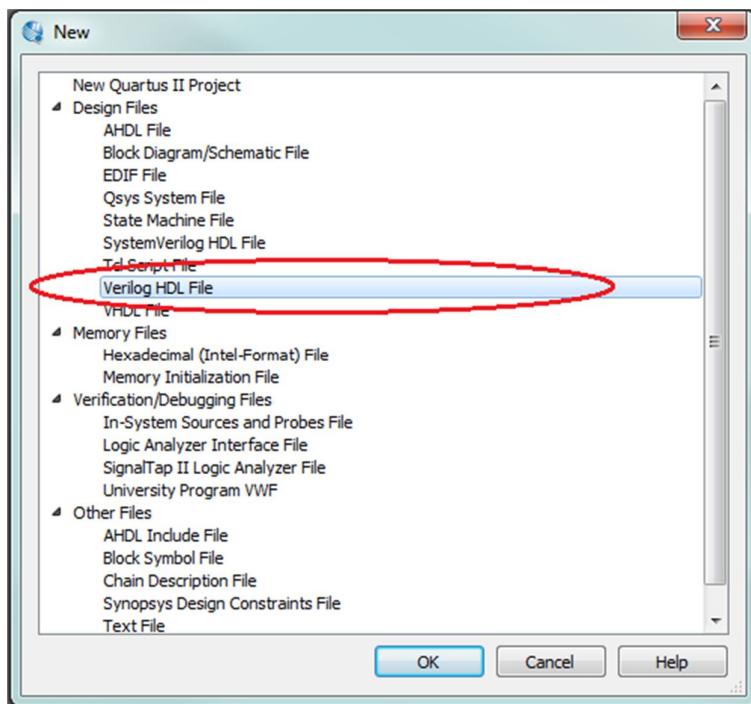


Figura 20: Criando um novo arquivo em Verilog.

Se preferir, como forma de aprendizado, poderá criar um novo projeto antes, repetindo os passos do capítulo 5. Antes de qualquer coisa, clique em “Ctrl + S” no teclado para salvar o arquivo. Salve na mesma pasta do projeto criado com o nome “my_latch.v”. A extensão “.v” designa um arquivo em Verilog.

A primeira coisa a se fazer, conforme especificado no capítulo anterior é o cabeçalho do seu projeto, e este sempre é feito utilizando-se os comentários. Comentários em Verilog podem ser feitos de duas formas, como mencionado no capítulo 3, que são linha de comentário e bloco de comentários. Os mesmos

comandos da linguagem C padrão ANSI são utilizados, portanto, para comentar apenas uma linha, utiliza barra-barra “//” e para comentar um parágrafo inteiro, utilize barra-asterisco para abrir o bloco e asterisco-barra para fechar “/* */”. No Box 8 demonstramos um exemplo de cabeçalho para o código proposto.

```
/*
    Flip-Flop SR (Latch) em Verilog

    Entradas: S (Set) e R (Reset)

    Saídas: Q e Qn

    Autor: Eng. Wagner Rambo

    Data: Junho de 2015
*/
```

Box 8: Cabeçalho do código em Verilog.

Os projetos em Verilog são desenvolvidos em módulos, com a declaração “module”, veja no Box 9 o módulo da latch.

```
module my_latch(S,R,Q,Qn); //declaração do módulo

    input S,R;                //entradas do sistema
    output Q,Qn;              //saídas do sistema
    wire Qs, notQ;            //sinais do sistema
```

Box 9: Declaração do módulo da latch em Verilog.

Existem outras formas de se implementar este módulo, mas procurou-se fazer de forma semelhante ao código em VHDL do capítulo anterior, para haver coerência entre as metodologias. O nome do módulo é determinado logo após o comando “module”, no caso do exemplo chamamos de “my_latch”. Entre parênteses, após o nome, informa-se as entradas e saídas do sistema, no caso, utilizou-se S e R como entradas e Q, Qn como saídas.

As entradas são destacadas com o comando “input”, separadas por vírgula. Como são duas entradas, a linha das entradas ficou “input S,R;”. Declarou-se as saídas Q e Qn com o comando “output Q, Qn;”. A forma de se declarar sinais ou ramos internos de um circuito em Verilog é com o comando “wire”. Como as saídas

Q e \bar{Q} também comportam-se como entradas, optou-se por declarar ambas como sinais internos do circuito “wire Qs, notQ;”.

A descrição comportamental propriamente dita pode ser vista no Box 10.

```
assign Qs = ~(R | notQ);      // Q = R NOR notQ
assign notQ = ~(S | Qs);     // notQ = S NOR Q
assign Qn = notQ;           // Qn recebe sinal notQ
assign Q = Qs;

endmodule                  // Final do módulo
```

Box 10: Descrição comportamental do hardware.

Utilizando o comando “assign” descreve-se o comportamento das entradas, saídas e sinais do sistema. Na primeira linha tem-se “Qs=~(R | notQ);”, que significa dizer que Qs recebe a operação NÃO-OU entre a entrada R e o sinal notQ, onde a barra vertical “|” é a operação lógico OU e o sinal de til “~” a negação, caracterizando-a como uma NÃO-OU. Pode-se notar novamente semelhança entre Verilog e C ANSI. Na segunda linha tem-se “notQ = ~(S | Qs);”, que significa dizer que notQ recebe a operação NÃO-OU entre Set e Qs. Nossas saídas auxiliares Q e Qn, recebem diretamente o valor dos sinais Qs e notQ, de modo a podermos verifica-las na simulação ou na implementação prática. Note que este exemplo não requer inclusão de bibliotecas específicas.

O código completo da latch em Verilog pode ser visto no Box 11.

```

/*
Flip-Flop SR (Latch) em Verilog

Entradas: S (Set) e R (Reset)

Saídas Q e Qn

Autor: Eng. Wagner Rambo

Data: Junho de 2015
*/

module my_latch(S,R,Q,Qn); //declaração do módulo

input S,R; //entradas do sistema
output Q,Qn; //saídas do sistema
wire Qs, notQ; //sinais do sistema

assign Qs = ~(R | notQ); // Q = R NOR notQ
assign notQ = ~(S | Qs); // notQ = S NOR Q
assign Qn = notQ; // Qn recebe sinal notQ
assign Q = Qs;

endmodule // Final do módulo

```

Box 11: Circuito latch da Figura 16 descrito em Verilog.

O código estando pronto pode-se efetuar a análise, síntese e compilação do mesmo e isto será explanado em capítulo futuro.

Como último exemplo de descrição de hardware, demonstraremos no próximo capítulo como implementar a latch desenhandosse o próprio diagrama esquemático com a ferramenta “Block Diagram/Schematic File” do Quartus II.

8. DESCREVENDO UMA LATCH COM BLOCK DIAGRAM/SCHEMATIC

No presente capítulo ensinaremos a utilizar a ferramenta Block Diagram/Schematic do Quartus II, desenvolvendo o mesmo circuito da Figura 16, para que seja possível uma comparação entre os possíveis métodos de descrição de hardware por parte do leitor.

Crie um novo projeto no Quartus II e vá em “File > New”. Na janela que abrir, selecione a opção “Block Diagram/Schematic File” e clique em “Ok” (Figura 21).

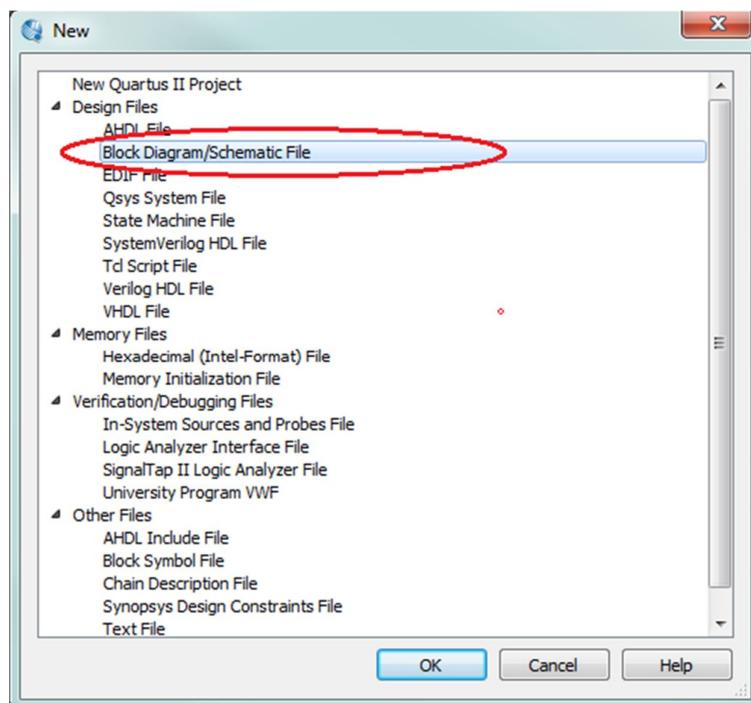


Figura 21: Novo arquivo em *Block Diagram/Schematic*.

Quem já está acostumado com simulação de circuitos eletrônicos em softwares do gênero, terá facilidade em utilizar esta ferramenta do Quartus II. O ambiente será carregado em seu projeto conforme mostra a Figura 22.

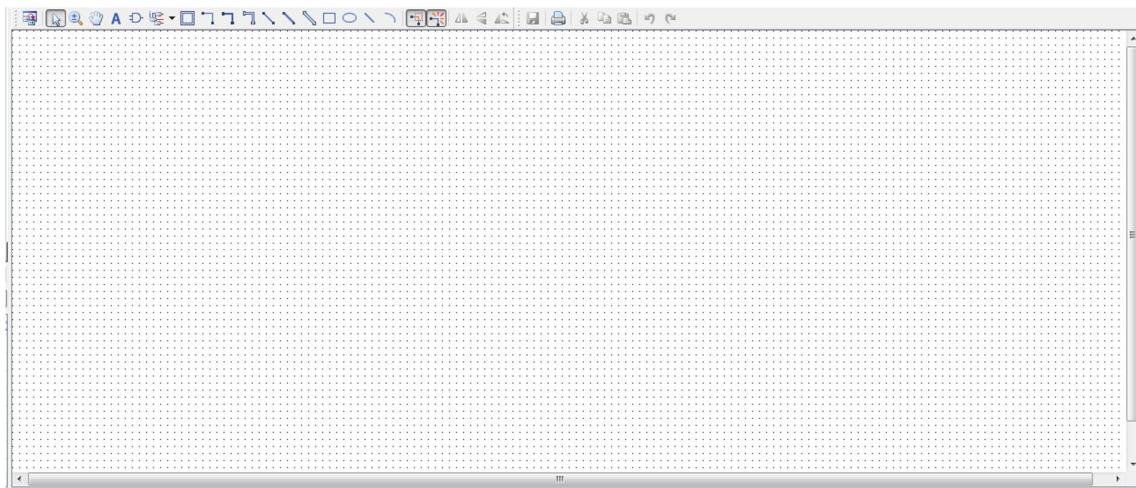


Figura 22: Ambiente de simulação de diagramas esquemáticos.

Vamos proceder à montagem do diagrama esquemático conforme Figura 16, inicialmente clique no ícone que tem uma porta lógica e na pasta “primitives”, depois em “logic” e vá até o dispositivo “nor2”, para selecionar uma porta NOR de duas entradas, e clique em “Ok” (Figura 23).

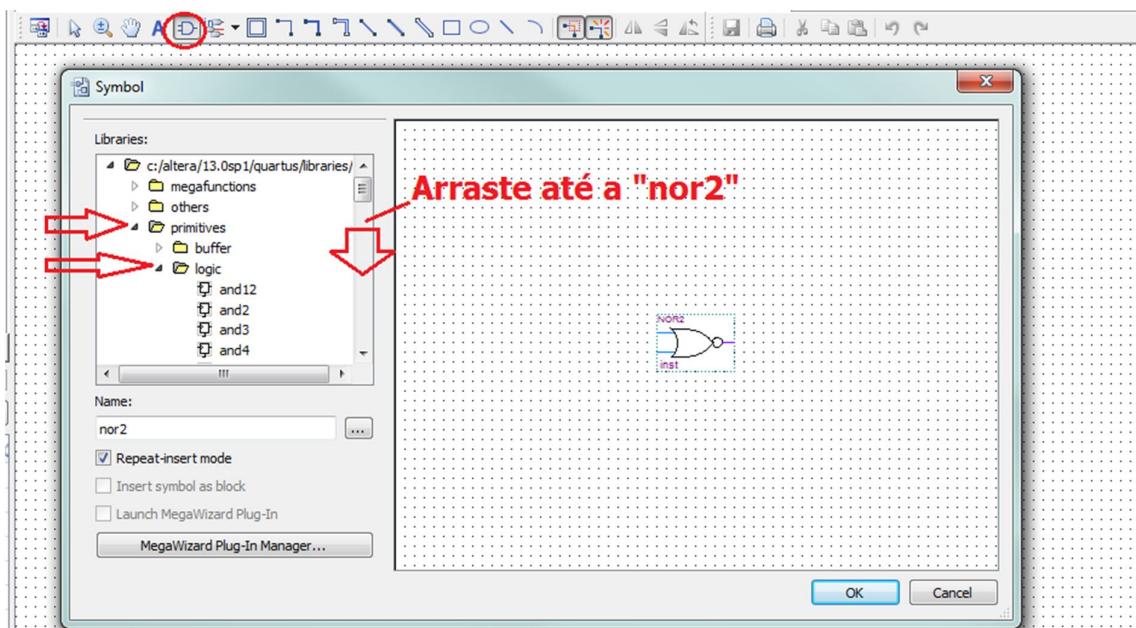


Figura 23: Selecionando a porta lógica NOR de duas entradas.

Adicione duas portas NOR de duas entradas na área de trabalho da ferramenta, clicando com o botão esquerdo do mouse. Para parar de adicionar portas, pressione “ESC” no teclado. Selecione a ferramenta “Orthogonal Node Tool”

e mantendo o botão esquerdo do mouse pressionado, desenhe as wires do circuito conforme a Figura 24.

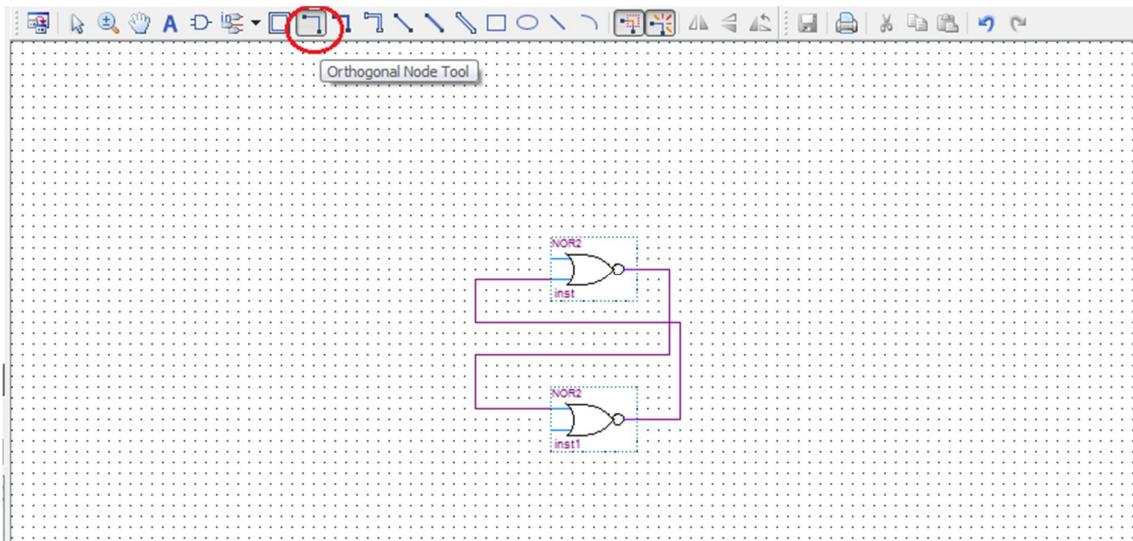


Figura 24: Desenho das linhas do circuito.

Selecione as entradas e saídas, adicione-as ao diagrama esquemático e ligue-as aos respectivos pontos do circuito finalizando o diagrama. Para renomear, dê duplo clique em cima dos nomes e escreva as entradas R e S e saídas Q e Qn (Figura 25).

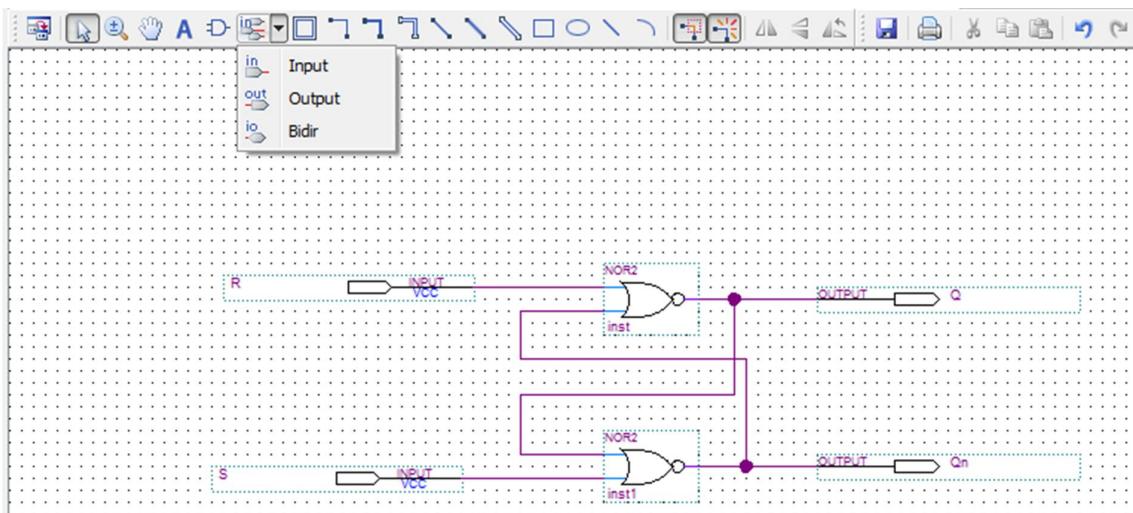


Figura 25: Diagrama esquemático da latch finalizado.

Com isso, seu diagrama estará finalizado pronto para se efetuar os procedimentos de análise, síntese e compilação, que serão detalhados no próximo capítulo. Não explanamos muito detalhadamente o uso desta ferramenta de

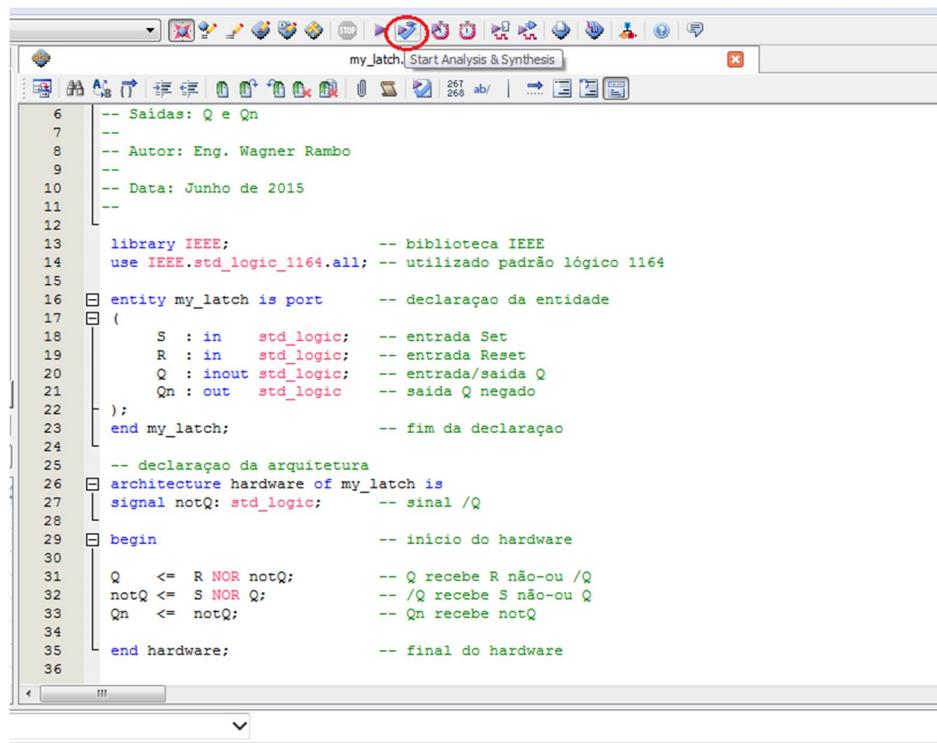
diagramas esquemáticos pelo fato da mesma ser totalmente intuitiva e por acreditar que o leitor já tenha certo conhecimento na montagem de diagramas com este tipo de software. Para mais detalhes, consulte o help do Quartus II e explore por si próprio todos os recursos do Block Diagram/Schematic File.

Com ele, você poderá desenvolver os mais diversificados circuitos, embora valha salientar que muitos projetos são de implementação mais rápida nas linguagens de descrição, que são os padrões mais utilizados na indústria atualmente.

9. COMPILAÇÃO, ANÁLISE E SÍNTESE DE UM PROJETO

Neste breve capítulo veremos como efetuar a compilação, análise e síntese de um projeto criado no Quartus II. Todos os demais passos abordados na presente apostila tomarão como base o primeiro arquivo criado em VHDL, pois são exatamente iguais com os demais arquivos. Recomenda-se que você efetue os mesmos procedimentos para os outros arquivos, a fim de praticar os processos de compilação, conversão, simulação, gravação no kit e testes práticos abordados nos próximos capítulos.

Antes de compilarmos o projeto, precisamos verificar se o mesmo não apresenta erros. Para isto, com o arquivo VHDL aberto, efetua-se a análise e a síntese do projeto. Clique no ícone conforme Figura 26, na opção “Start Analysis & Synthesis”.



```

6  -- Saídas: Q e Qn
7  --
8  -- Autor: Eng. Wagner Rambo
9  --
10 -- Data: Junho de 2015
11 --
12
13 library IEEE;           -- biblioteca IEEE
14 use IEEE.std_logic_1164.all; -- utilizado padrão lógico 1164
15
16 entity my_latch is port  -- declaração da entidade
17 (
18     S : in  std_logic;    -- entrada Set
19     R : in  std_logic;    -- entrada Reset
20     Q : inout std_logic; -- entrada/saída Q
21     Qn : out  std_logic; -- saída Q negado
22 );
23 end my_latch;          -- fim da declaração
24
25 -- declaração da arquitetura
26 architecture hardware of my_latch is
27 signal notQ: std_logic; -- sinal /Q
28
29 begin                  -- inicio do hardware
30
31     Q  <= R NOR notQ;   -- Q recebe R não-ou /Q
32     notQ <= S NOR Q;   -- /Q recebe S não-ou Q
33     Qn <= notQ;        -- Qn recebe notQ
34
35 end hardware;          -- final do hardware
36

```

Figura 26: Iniciando a análise e síntese do arquivo em VHDL.

O progresso deste processo você confere à esquerda, conforme Figura 27 e, caso não existam erros, será exibida a mensagem “Analysis & Synthesis was successful”.

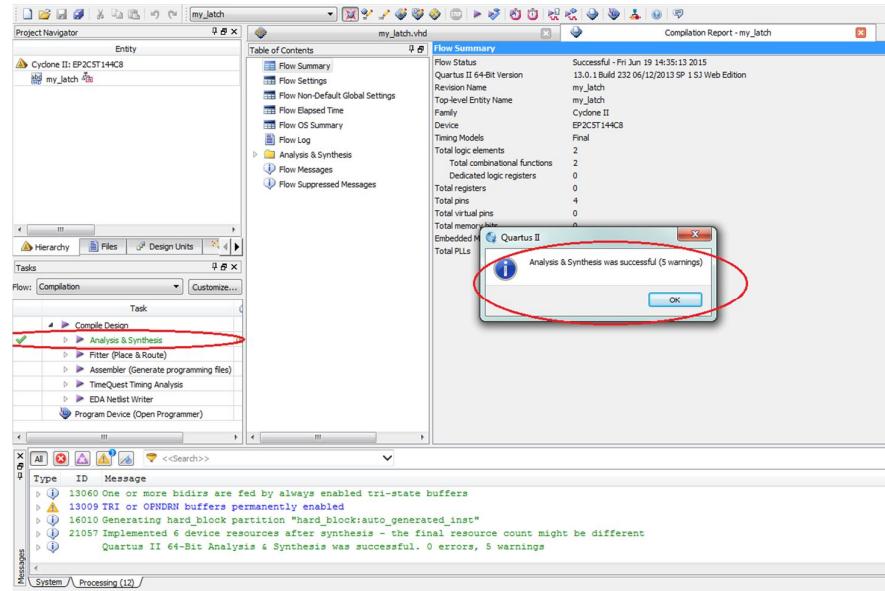


Figura 27: Análise e síntese do arquivo VHDL.

No rodapé, você pode visualizar os possíveis avisos. Nesta análise, houve o retorno de 5 avisos (warnings). Estes avisos dizem respeito às configurações do compilador e não irão afetar o projeto final, portanto, podem ser desconsiderados. Caso o sistema acuse algum erro, este sim, deverá ser corrigido. De qualquer forma, sempre deve-se dar uma lida nos avisos para assegurar-se que não irão afetar o projeto final. Após este procedimento, efetue a compilação do projeto, para gerar o arquivo com extensão .sof (Figura 28). Clique no ícone “Start Compilation” e aguarde o final do processo, que não deverá retornar nenhum erro. Se aparecer a mensagem “Full Compilation was successful”, significa que seu projeto não contém erros.

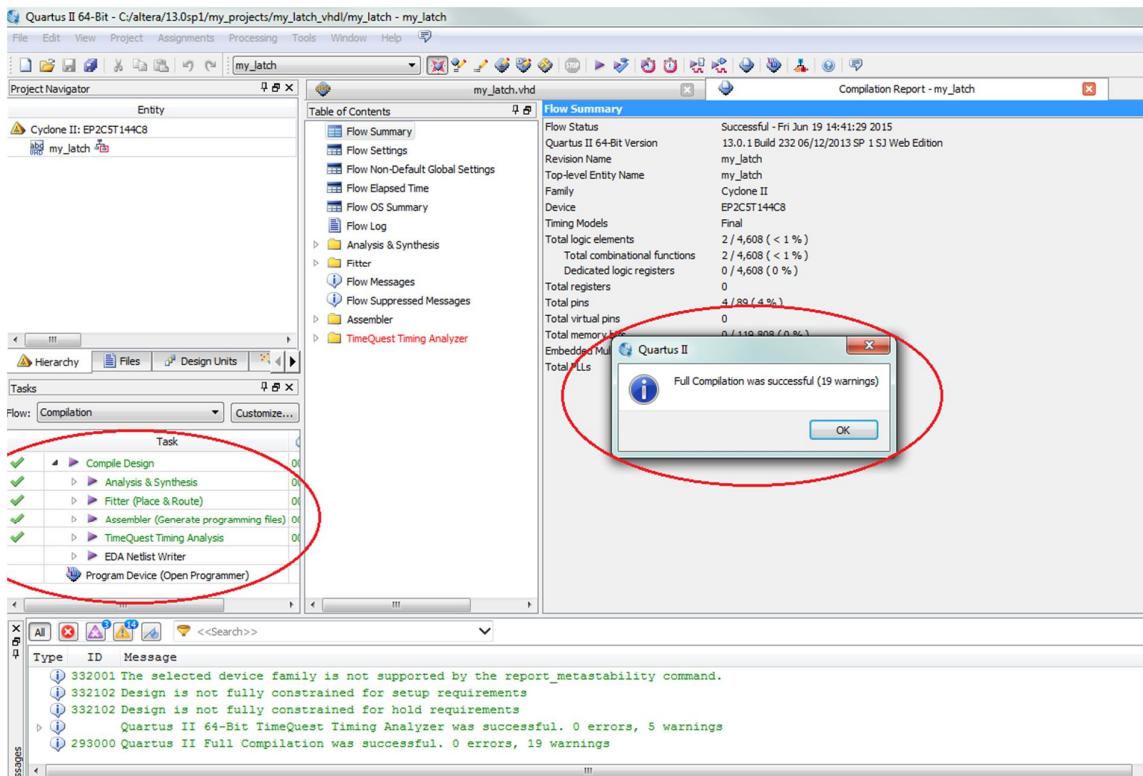


Figura 28: Compilação bem sucedida.

Pode-se observar que agora são 19 warnings retornados, mas a compilação foi bem sucedida. Novamente, atentando-se para estes avisos, nota-se que dizem respeito apenas a configurações do compilador, ausência de clock no circuito (o que está correto, pois consiste em um latch SR sem entrada de clock), encontrados dois pinos de saída sem capacidade associada (que são nossas saídas auxiliares). Vá até a pasta em que o projeto está salvo e entre na pasta “output_files” (criada pelo próprio compilador) e assegure-se de que há um arquivo “my_latch.sof”.

Se os procedimentos estiverem dentro dos conformes até aqui, podemos passar para o capítulo seguinte, onde iremos efetuar a simulação do dispositivo descrito em hardware, utilizando a ferramenta Vector Wave Form do Quartus II.

Se houverem erros, provavelmente é algum detalhe da sintaxe do código. O relatório do rodapé sempre indica o número da linha de seu código onde o erro está, verifique cada uma até conseguir efetuar a análise, síntese e compilação do projeto sem erros detectados, sempre efetuando a interpretação dos possíveis warnings que irão surgir.

10. SIMULANDO UM PROJETO

A etapa de simulação tem a função de comprovar o funcionamento do dispositivo descrito previamente em hardware, antes de efetuar a conversão do arquivo .sof e posterior gravação no kit EE02-SOQ. Mesmo que a análise, síntese e compilação não indiquem erros sintáticos, não significa que seu dispositivo não contenha nenhum erro semântico.

A priori, deve-se ter o conhecimento do comportamento exato do circuito que deseja-se implementar no FPGA. No caso do exemplo abordado, consiste na latch, onde seu comportamento foi explanado no capítulo 6 desta apostila. Tomaremos novamente como base o arquivo em VHDL criado no mesmo capítulo. De qualquer forma, aconselha-se ao leitor que faça as mesmas simulações também nos demais modos de descrição, o Verilog e o Diagrama Esquemático.

Com o projeto e o arquivo em VHDL abertos utilize o atalho “Ctrl + N” no teclado para criar um novo arquivo. Em “Verification/Debugging Files”, selecione a opção “University Program VWF” e clique em “Ok”, conforme Figura 29.

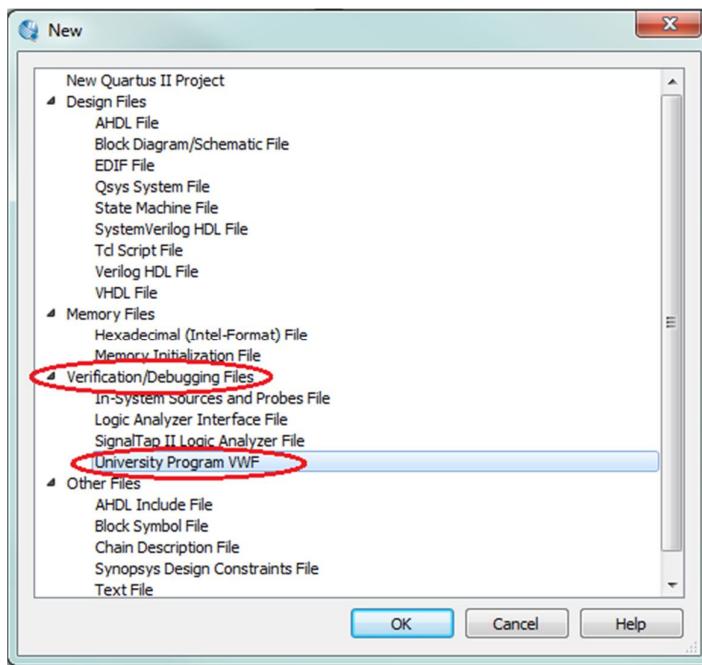


Figura 29: Criando um novo arquivo para simulação “Vector Wave Form”.

Após clicar em “Ok”, irá abrir a janela da ferramenta, conforme Figura 30.

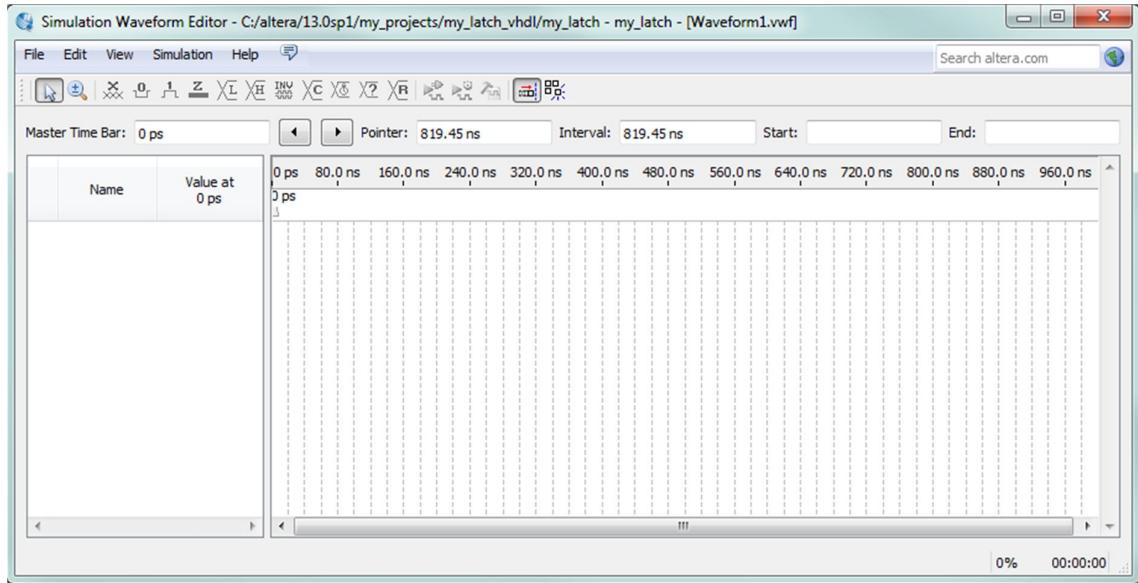


Figura 30: Ferramenta de simulação aberta.

Clique com o botão direito do mouse no canto em branco à esquerda da janela, e depois em “Insert Node or Bus”. (Figura 31).

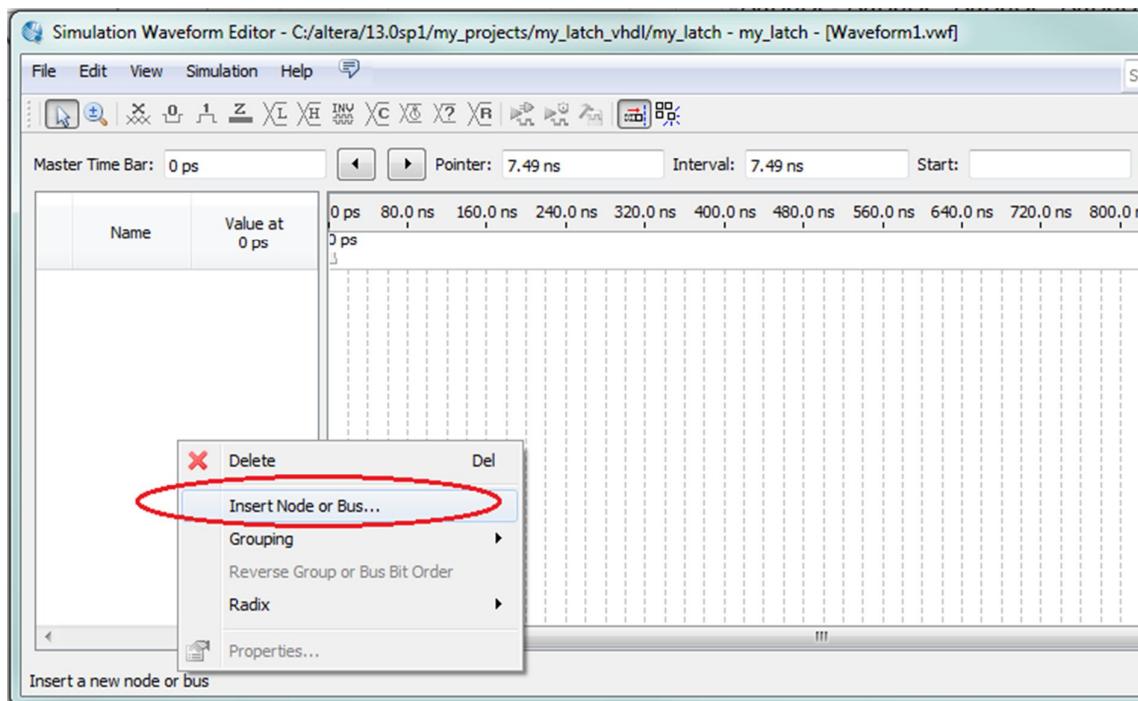


Figura 31: Inserindo as entradas e saídas para simulação.

Na janela que abrir, clique em “Node Finder...” (Figura 32).

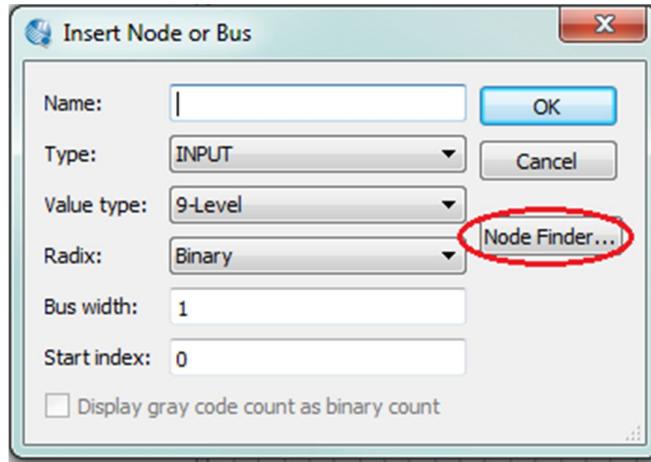


Figura 32: Encontrando as entradas e saídas.

Depois, clique em “List”. Você verá a lista de entradas e saídas do circuito latch à esquerda. Pode-se adicionar apenas algumas entradas e saídas para a simulação, mas no caso, simularemos todas elas logo, temos que adicioná-las clicando no botão “>>” localizado ao centro da janela. Dessa forma, todas as entradas e saídas aparecerão na lista da direita e você poderá clicar em “Ok” (Figura 33).

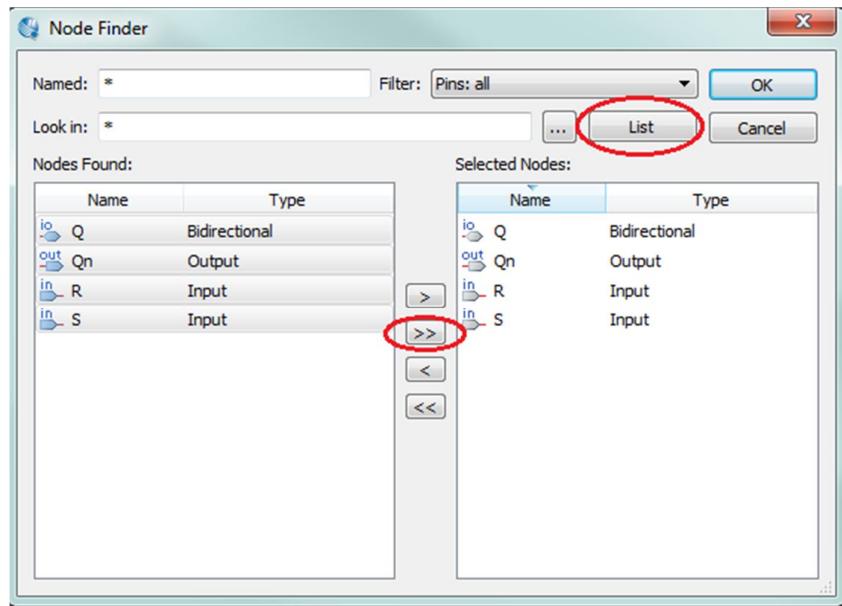


Figura 33: Listando e adicionando as entradas e saídas para a simulação.

Clique novamente em “Ok” na janela “Insert Nodes or Bus” para adicionar todas as entradas e saídas no simulador (Figura 34).

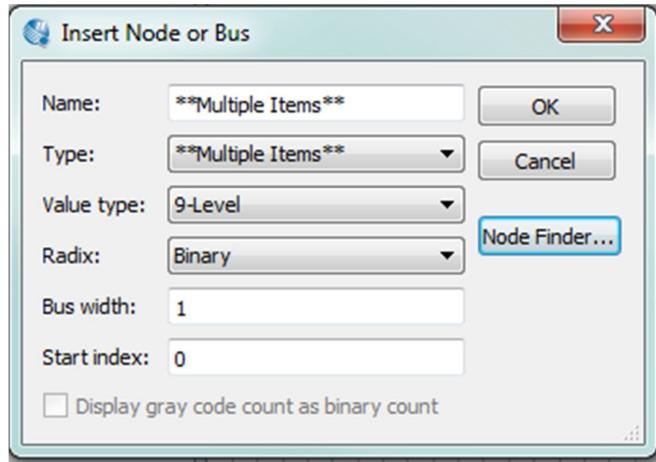


Figura 34: Confirmando a inclusão de todas entradas e saídas.

Após, à esquerda você pode organizar as entradas e saídas clicando e mantendo pressionado o botão esquerdo do mouse, então é só arrastar na ordem desejada. No caso, ordenamos de cima para baixo: S, R, Q e Qn, pois julgamos ser a melhor ordem para se efetuar a análise das formas de onda. Para adicionar um clock nas entradas, clique na entrada S inicialmente e depois no botão “Overwrite Clock” e selecione um período de 10ns na janela “Clock” que abrir, clicando por último em “Ok” (Figura 35).

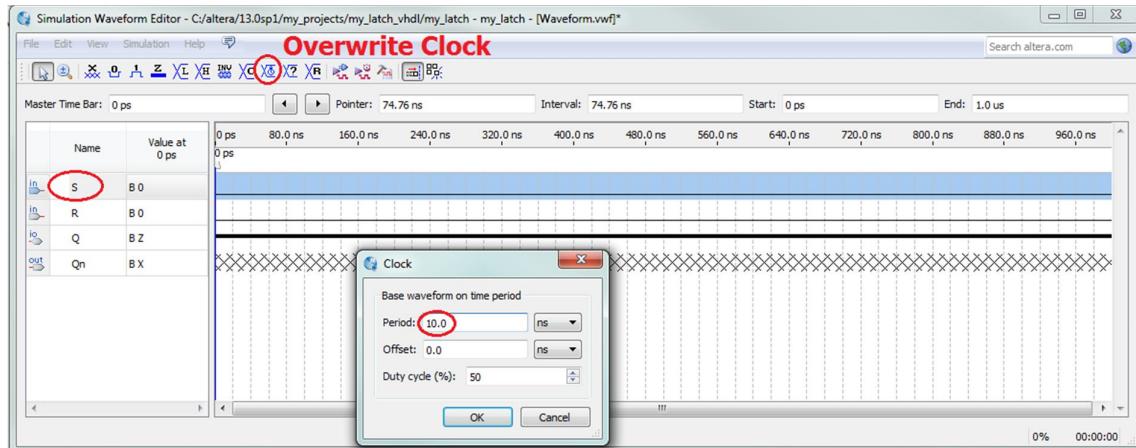


Figura 35: Configurando uma frequência de 10ns para a entrada S.

Repita o procedimento supracitado para a entrada R, entretanto dessa vez configure seu clock com período de 20ns. Após configurar ambas as entradas, as formas de onde se apresentarão conforme Figura 36, e o projeto estará pronto para ser simulado.

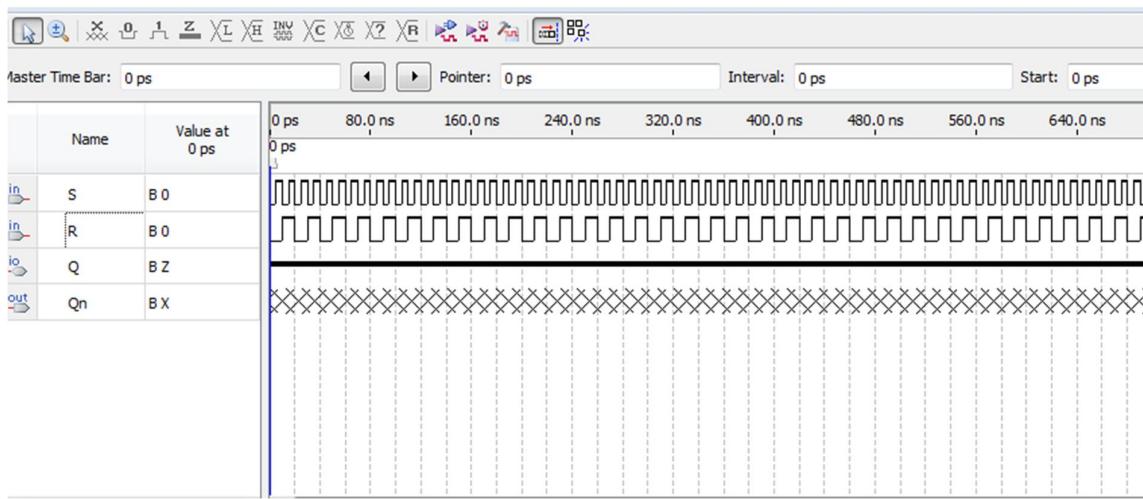


Figura 36: Formas de onda de entrada configuras para a simulação.

Clique em “Run Functional Simulation” e o Quartus II perguntará se você deseja salvar o arquivo de Waveform. Clique em “Yes” e salve na pasta do seu projeto com a extensão .vWF (Figura 37).

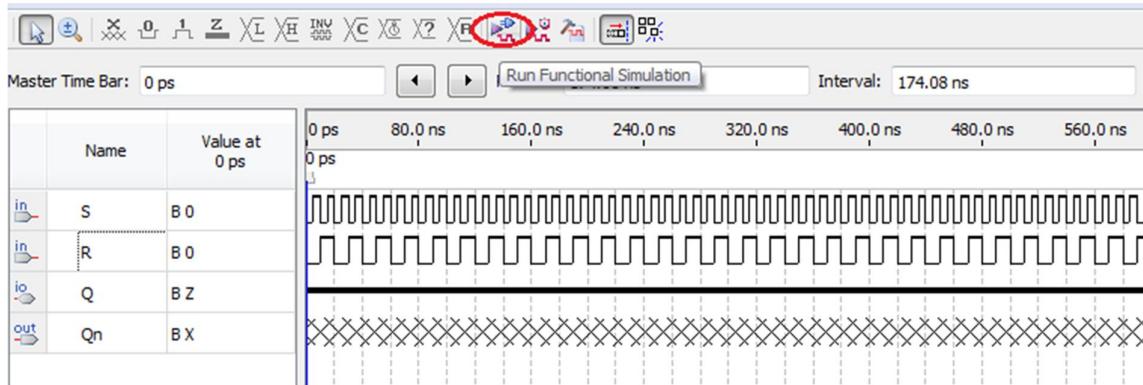
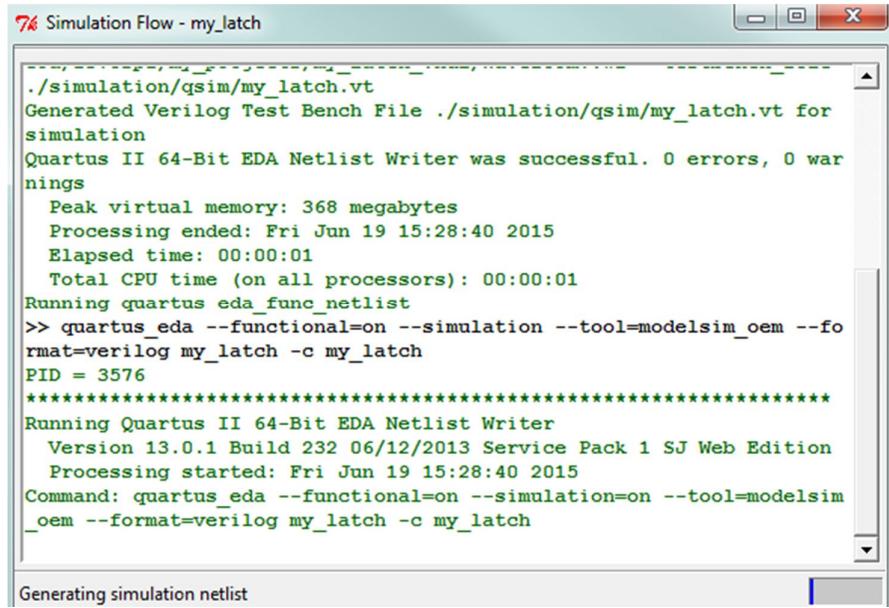


Figura 37: Rodando a simulação.

O status de simulação irá aparecer (Figura 38).



```

./simulation/qsim/my_latch.vt
Generated Verilog Test Bench File ./simulation/qsim/my_latch.vt for
simulation
Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 war
nings
Peak virtual memory: 368 megabytes
Processing ended: Fri Jun 19 15:28:40 2015
Elapsed time: 00:00:01
Total CPU time (on all processors): 00:00:01
Running quartus_eda_func_netlist
>> quartus_eda --functional=on --simulation --tool=modelsim_oem --fo
rmat=verilog my_latch -c my_latch
PID = 3576
*****
Running Quartus II 64-Bit EDA Netlist Writer
Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition
Processing started: Fri Jun 19 15:28:40 2015
Command: quartus_eda --functional=on --simulation=on --tool=modelsim
_oem --format=verilog my_latch -c my_latch

```

Generating simulation netlist

Figura 38: Status da simulação em progresso.

Após, será aberta uma nova janela, com a resposta das saídas do circuito, conforme Figura 39.

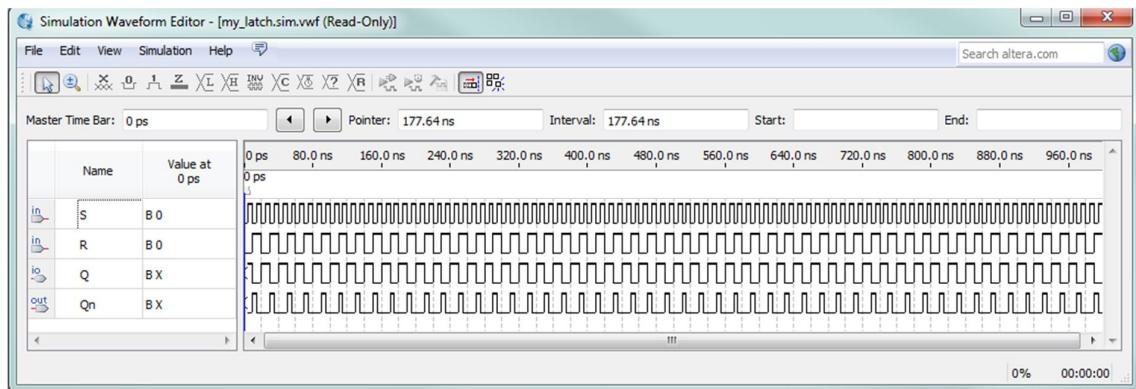


Figura 39: Formas de onda das entradas e saídas do sistema.

O próximo passo, consiste em uma análise visual das formas de onda, para verificar se estão coerentes com a tabela verdade prevista para a latch. Utilize a ferramenta de zoom para que fique mais fácil de visualizar. Veja na Figura 40 o trecho inicial das formas de onda.

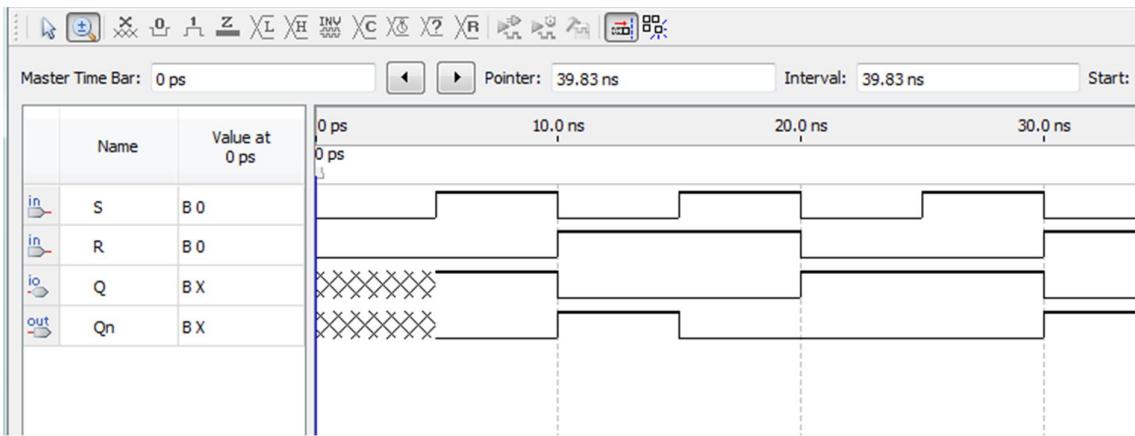


Figura 40: Zoom das formas de onda do sistema.

Baseando-se as formas de onda da Figura 40, elabora-se a seguinte tabela verdade:

R	S	Q	Q _n
0	0	Q_{ATUAL}	$\overline{Q_{ATUAL}}$
0	1	1	0
1	0	0	1
1	1	-	-

Tabela 1: Tabela verdade do circuito descrito em VHDL.

Pode-se portanto constatar que o flip-flop SR está funcionando conforme previsto. O sistema inicia com ambas as entradas em nível lógico baixo, por isso, as saídas estão indeterminadas (veja as pequenas grades diagonais logo no início das formas de onda das saídas). Com Set em nível alto, tem-se nível alto na saída Q e nível baixo na saída Q_n. Com Reset em nível alto, tem-se nível baixo na saída Q e nível alto na saída Q_n. Ambas as formas de onda em "1", o sistema demonstra ambas as saídas em "0". Mas sabemos que esta combinação não é permitida, portanto na prática, não efetuar-se-á este teste. Após o primeiro Set, sempre que ambas as entradas estão em nível baixo, a saída Q apresenta o estado atual e a saída Q_n apresenta o inverso, caracterizando-o como um dispositivo de memória.

Depois de todas essas análises, chegou a hora de efetuar a conversão do arquivo .sof para .jic, a fim de o gravarmos na memória externa do kit de desenvolvimento para enfim efetuarmos o teste prático, mas antes, vamos escolher os pinos do FPGA para associar às entradas e saídas da latch.

11. DEFININDO OS PINOS FÍSICOS A SEREM UTILIZADOS

Neste capítulo, iremos definir a quais pinos físicos do FPGA nossas entradas e saídas estarão associadas. Para isso, clique no botão “Pin Planner” do Quartus II e abrirá a ferramenta indicada na Figura 41, com a imagem física do FPGA EP2C5T144C8N da família Cyclone II.

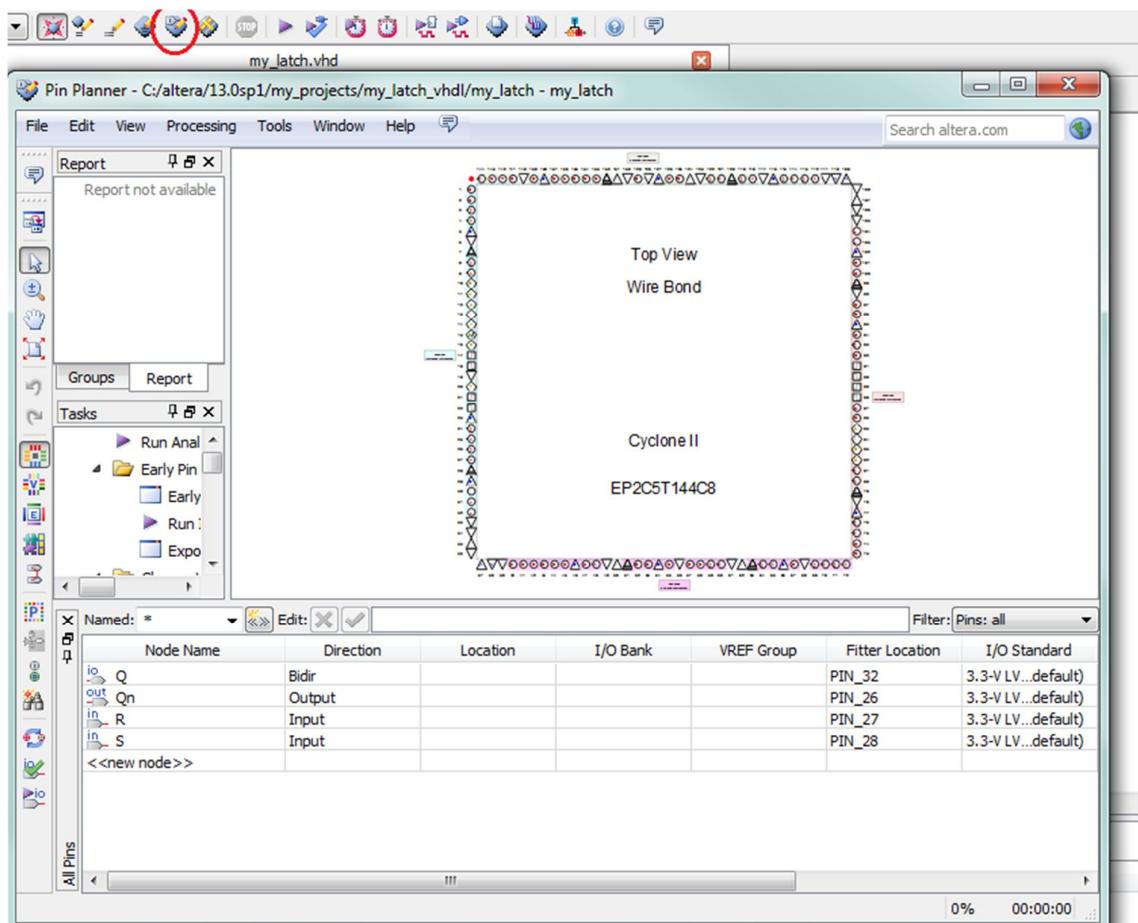


Figura 41: Ferramenta Pin Planner do Quartus II.

Verifique na própria placa do kit EE02-SOQ os pinos físicos disponíveis para utilização, todos eles estão identificados junto aos respectivos soquetes. Em nosso caso, iremos selecionar os pinos 94 e 96 para as respectivas entradas S e R, e os pinos 99 e 100 para as respectivas saídas Q e Qn do flip-flop SR. Vamos descrever como selecionar o pino 94 para a entrada S. Para os demais, o processo é exatamente o mesmo.

Clique no pino 94 na própria imagem física do FPGA e abrirá um campo à direita “Pin Properties”. Clique na aba “Node Name” e selecione a entrada S para este pino. Verifique no rodapé que a entrada S foi associada ao pino 94 do FPGA (Figura 42).

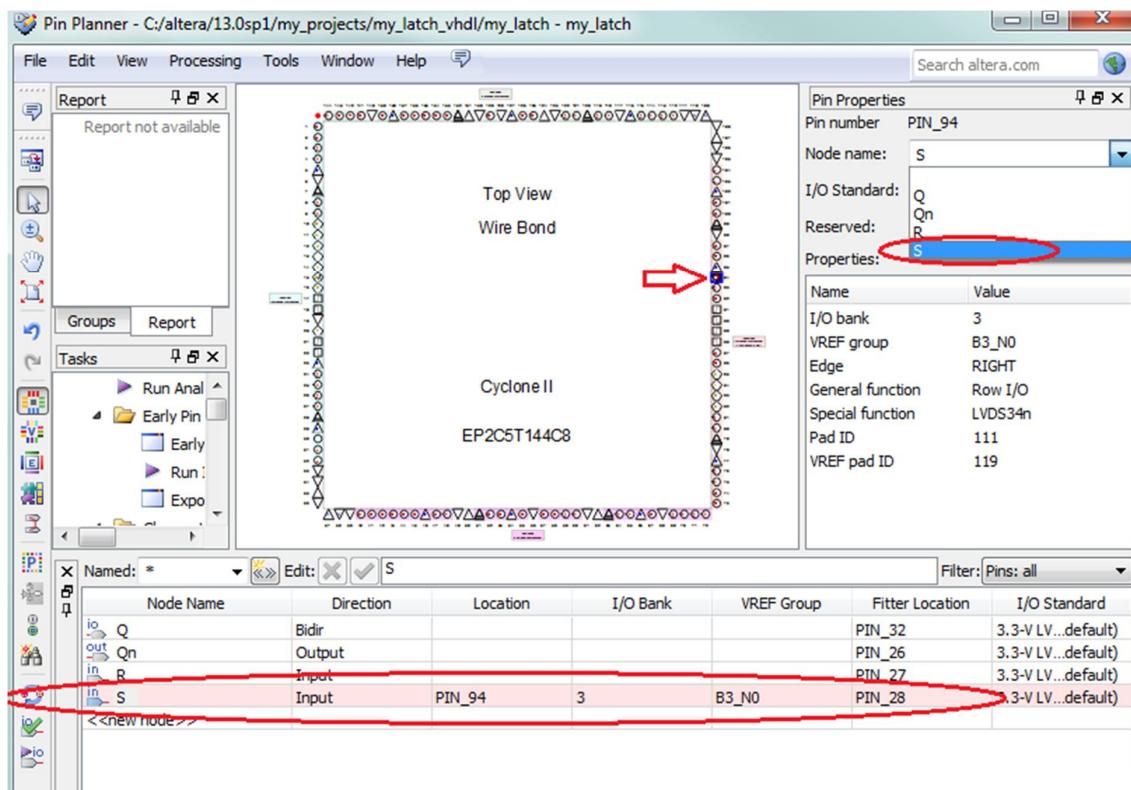


Figura 42: Associando a entrada S ao pino 94 do FPGA.

Repita o mesmo processo para a entrada R e para as saídas Q e Qn e verifique no rodapé que todos os pinos estão associados às respectivas entradas e saídas, conforme Figura 43.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
io_Q	Bidir	PIN_99	3	B3_N0	PIN_32	3.3-V LV...default)		24mA (default)	
out_Qn	Output	PIN_100	3	B3_N0	PIN_26	3.3-V LV...default)		24mA (default)	
in_R	Input	PIN_96	3	B3_N0	PIN_27	3.3-V LV...default)		24mA (default)	
In_S	Input	PIN_94	3	B3_N0	PIN_28	3.3-V LV...default)		24mA (default)	
<<new node>>									

Figura 43: Pinos associados às respectivas entradas e saídas.

Por fim, efetue a análise, síntese e compilação novamente para podermos efetuar a conversão do arquivo .sof em .jic e posterior gravação no kit EE02-SOQ.

12. CONVERSÃO DO ARQUIVO .SOF EM .JIC

Para que seja possível manter o código salvo em nossa placa após o corte de energia, é necessária uma memória externa, a EPCS4. A compilação do Quartus II gera um arquivo com extensão .sof, entretanto este não pode ser gravado diretamente na memória externa. É necessária a conversão para a extensão .jic compatível com a memória. Veja neste capítulo como realizar a conversão, utilizando o próprio Quartus II. Inicialmente, com o projeto e o arquivo em VHDL abertos, vá em “File” e depois “Convert Programming Files...” (Figura 44).

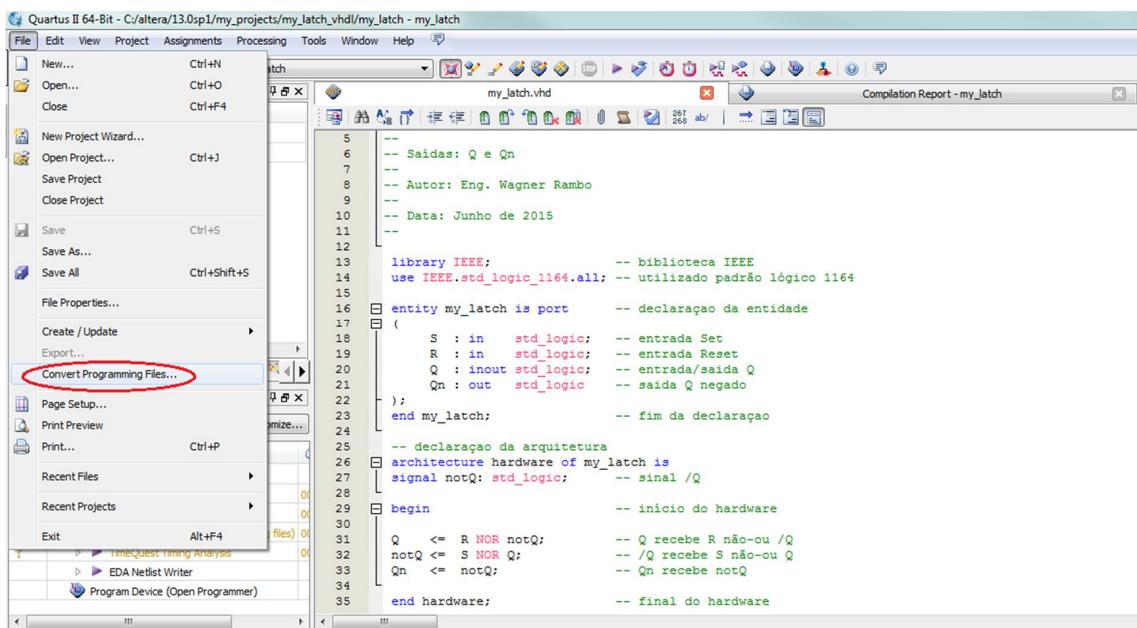


Figura 44: Abrindo a ferramenta de conversão.

Na janela que abrir, em “Programming file type”, escolha a opção “JTAG Indirect Configuration File (.jic)”. Depois selecione a memória utilizada em “Configuration device”, no caso é a EPCS4. Em “File name” indique o caminho para a pasta de criação do seu projeto. Confira na Figura 45.

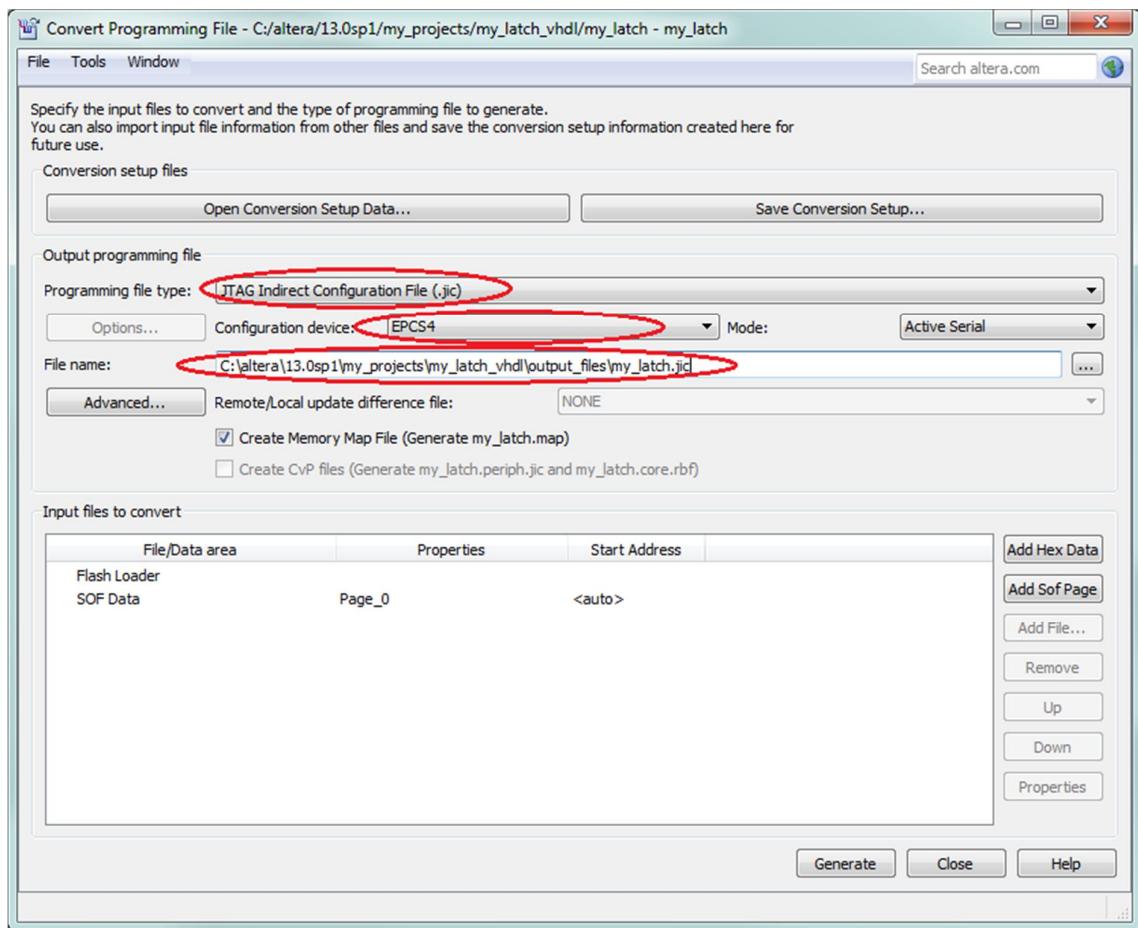


Figura 45: Configurações iniciais para conversão.

Clique em “SOF Data” [1] e depois em “Add File” [2], navegue até a pasta onde está o seu arquivo .sof gerado na compilação, selecione-o com duplo clique. O mesmo irá aparecer logo abaixo na lista da esquerda [3]. Confira na Figura 46.

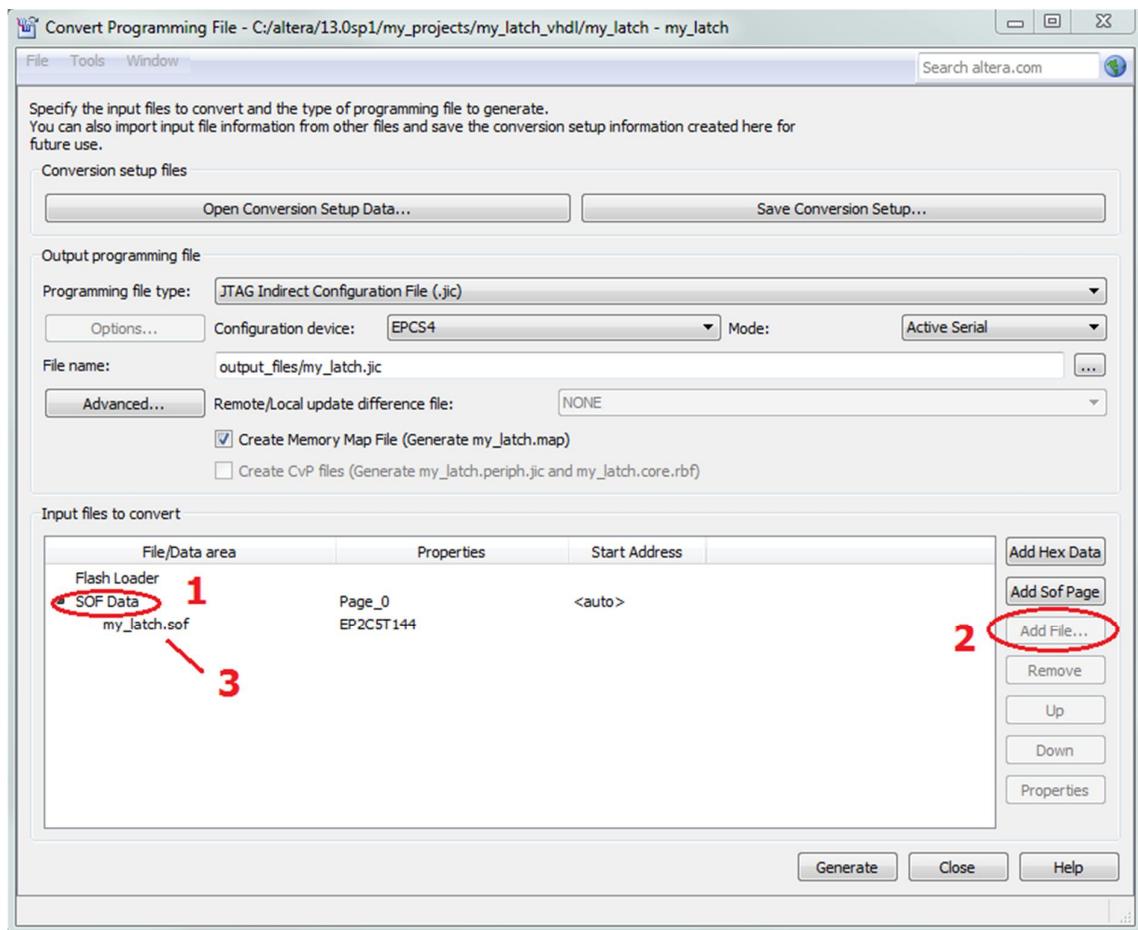


Figura 46: Adicionando o arquivo para conversão.

Agora clique em “Flash Loader” e em seguida em “Add Device” (Figura 47).

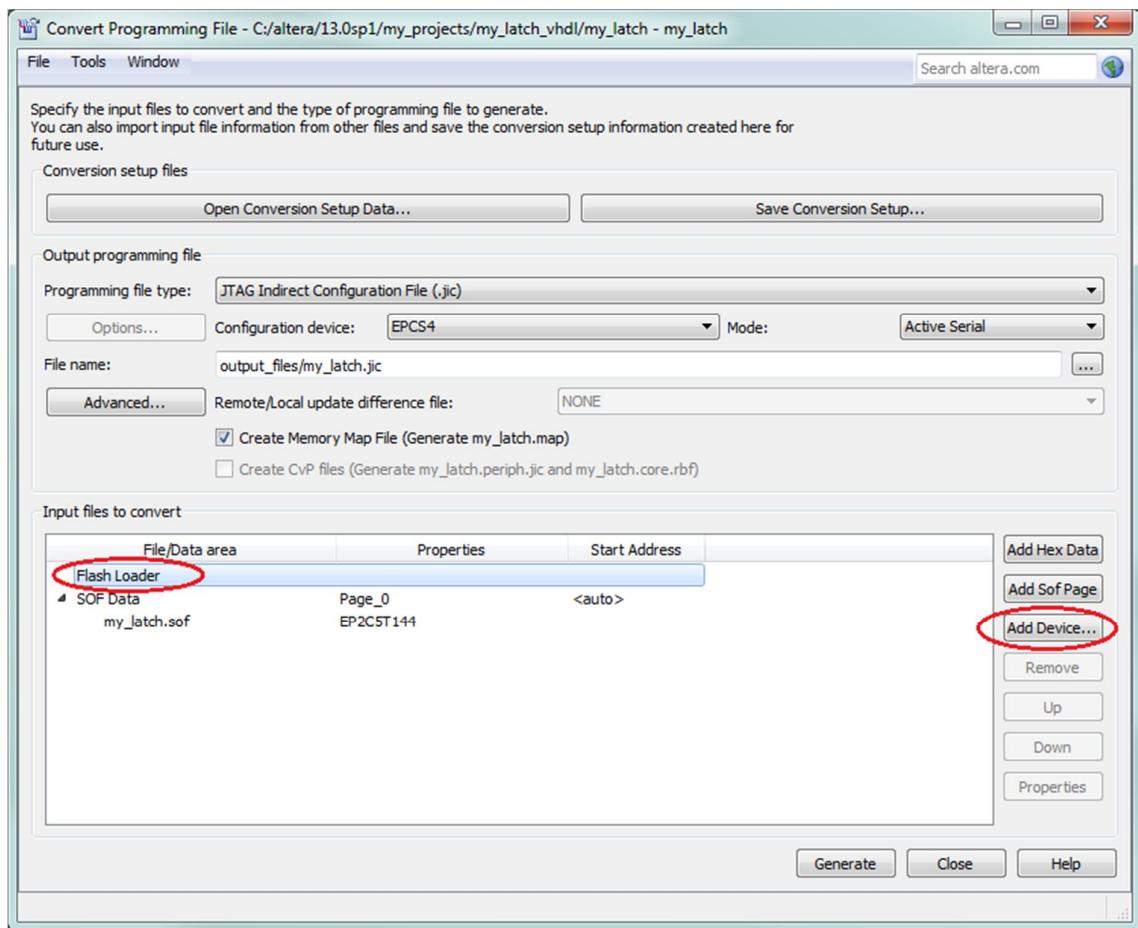


Figura 47: Adição do FPGA.

Na janela que abrir marque a opção “Cyclone II” à esquerda e depois “EP2C5” à direita, pois é o modelo do FPGA do kit EE02-SOQ (Figura 48).

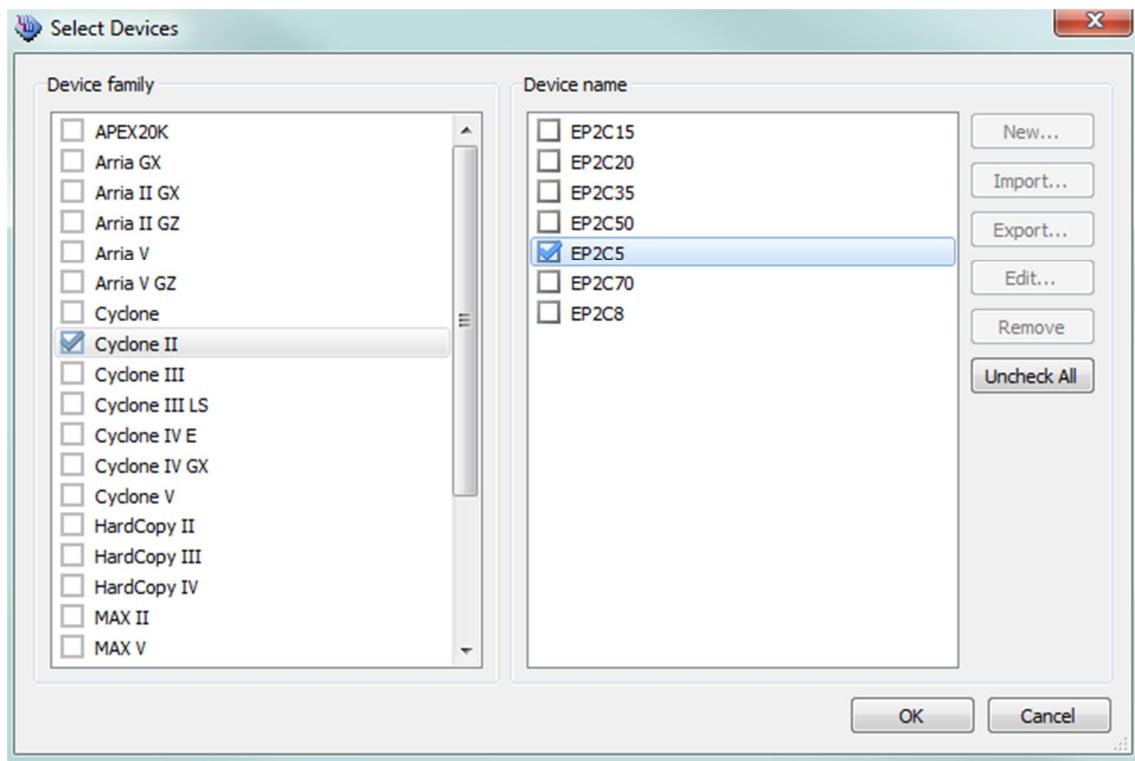


Figura 48: Selecionando o FPGA do kit EE02-SOQ.

Finalmente, clique em “Generate” e se tudo estiver correto, aparecerá a mensagem “Generated successfully” (Figura 49). Clique em “Ok” e o arquivo .jic estará gerado e pronto para ser gravado na memória externa, etapa esta que abordaremos no próximo capítulo.

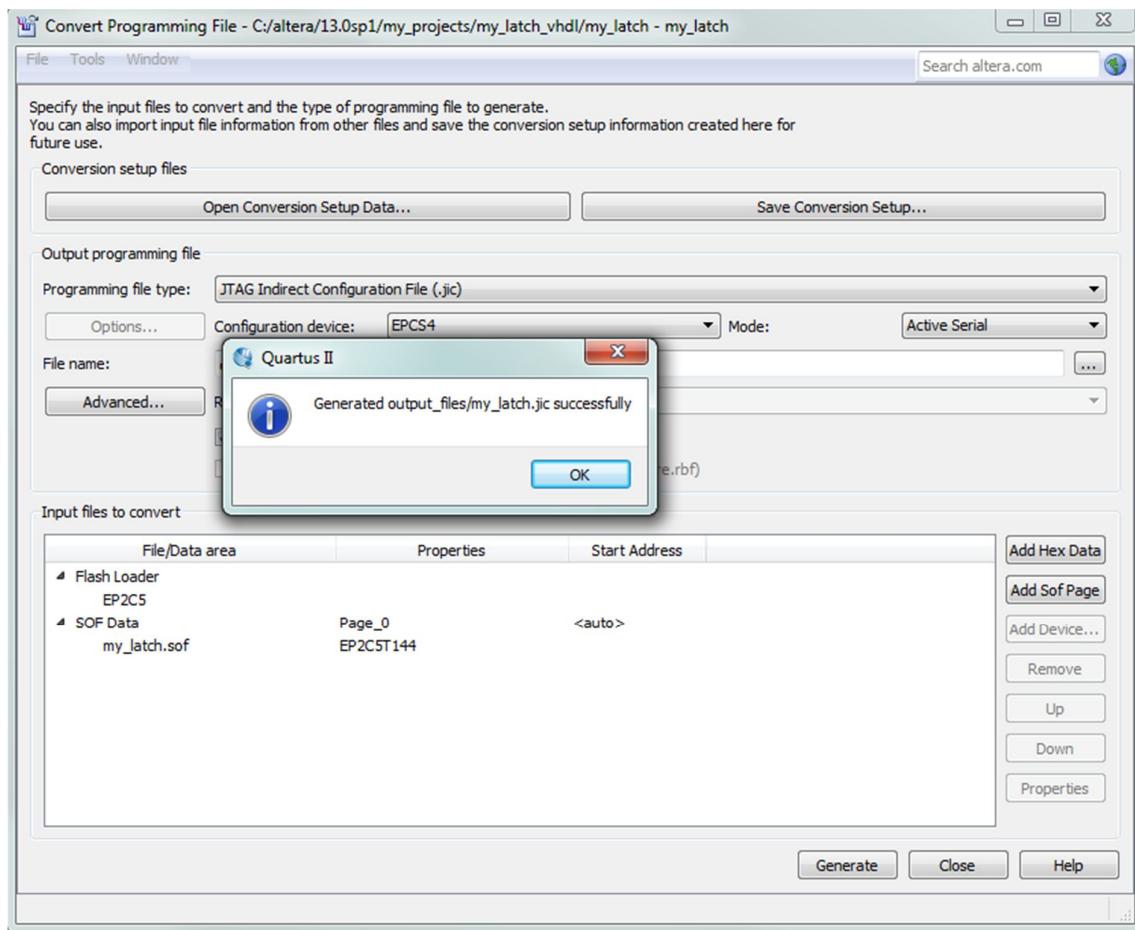


Figura 49: Arquivo .jic gerado com sucesso.

Navegue até a pasta “output_files” do seu projeto e confirme que o arquivo .jic está criado, antes de ir para o próximo capítulo.

13.GRAVANDO A MEMÓRIA EXTERNA EPCS4

Para gravar o arquivo .jic na memória externa EPCS4 e posteriormente efetuar-se o teste e comprovar o funcionamento de nosso projeto, iremos utilizar o gravador USB Blaster da Altera (Figura 50), que acompanha o kit EE02-SOQ.



Figura 50: Gravador USB Blaster da Altera.

Abra o Quartus II e o seu projeto com arquivo VHDL. Plugue o gravador em uma porta USB do seu computador. Se for a primeira vez que está conectando o gravador ao seu computador, seu hardware será instalado automaticamente com os drivers já nativos do próprio Quartus II. Após aparecer a mensagem de que seu hardware novo foi instalado corretamente e está pronto para ser utilizado, vá ao gerenciador de dispositivos do computador para comprovar isso (Figura 51).

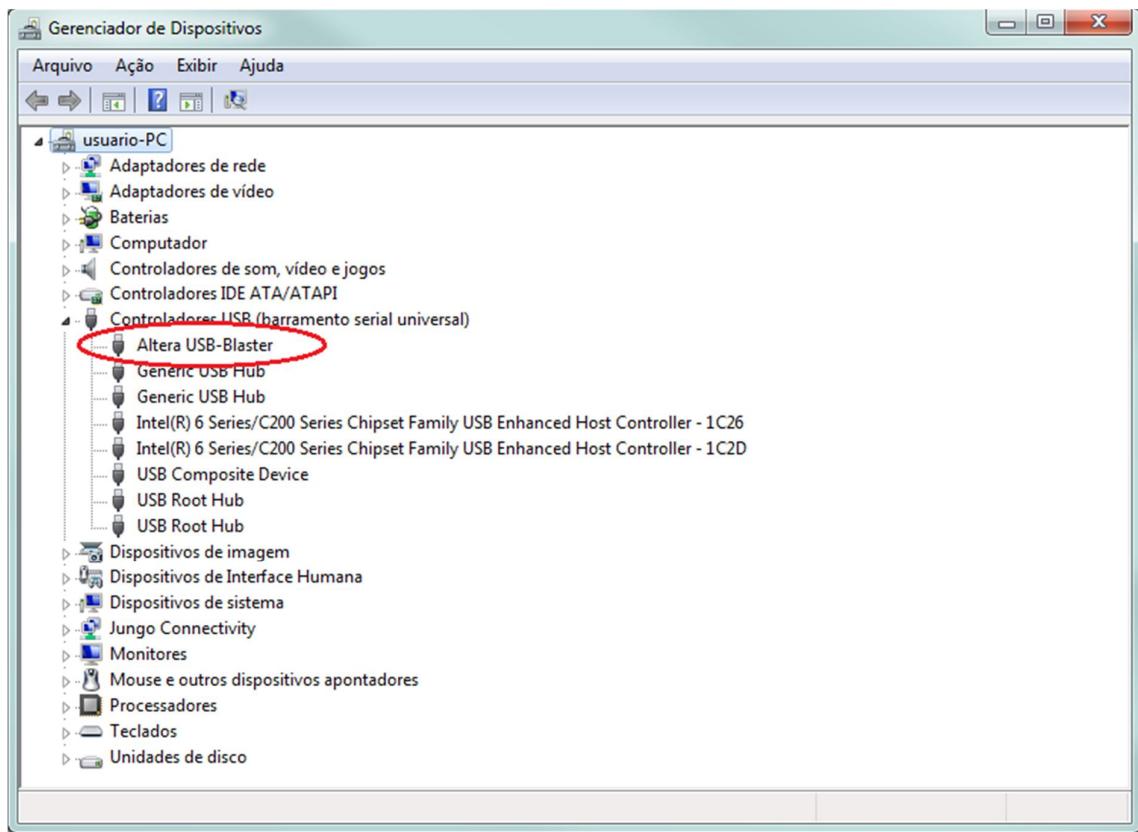


Figura 51: *USB Blaster reconhecido pelo computador.*

Se houver qualquer problema, você pode atualizar os drivers manualmente, indo nesta opção no gerenciador de dispositivos e navegando até a pasta quartus/drivers, no diretório onde foi instalado o compilador. Esta pasta deve ser selecionada para a atualização de drivers.

Depois, conecte o flat do USB Blaster no conector GRAVADOR da placa do kit EE02-SOQ, ligue a fonte de alimentação e pressione a chave S1. Deve verificar que o LED da placa irá acender, indicando a energização, conforme Figura 52.

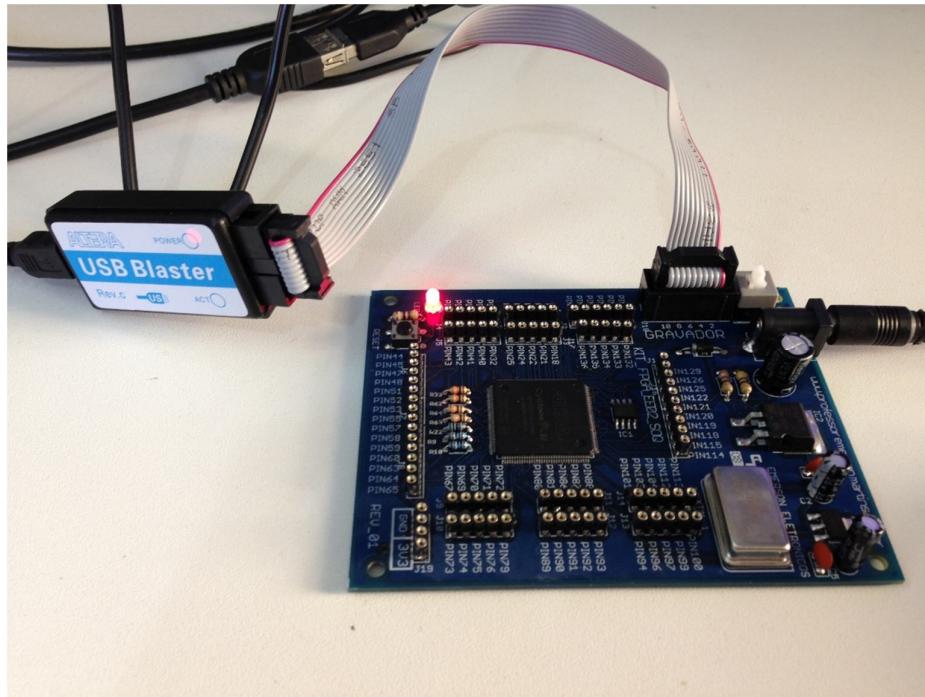


Figura 52: Placa energizada e gravador conectado ao soquete.

Agora clique no botão “Programmer” do Quartus II, depois em “Add File” e na pasta dentro da pasta do seu projeto “output_files”, selecione o arquivo .jic gerado no capítulo anterior (Figura 53).

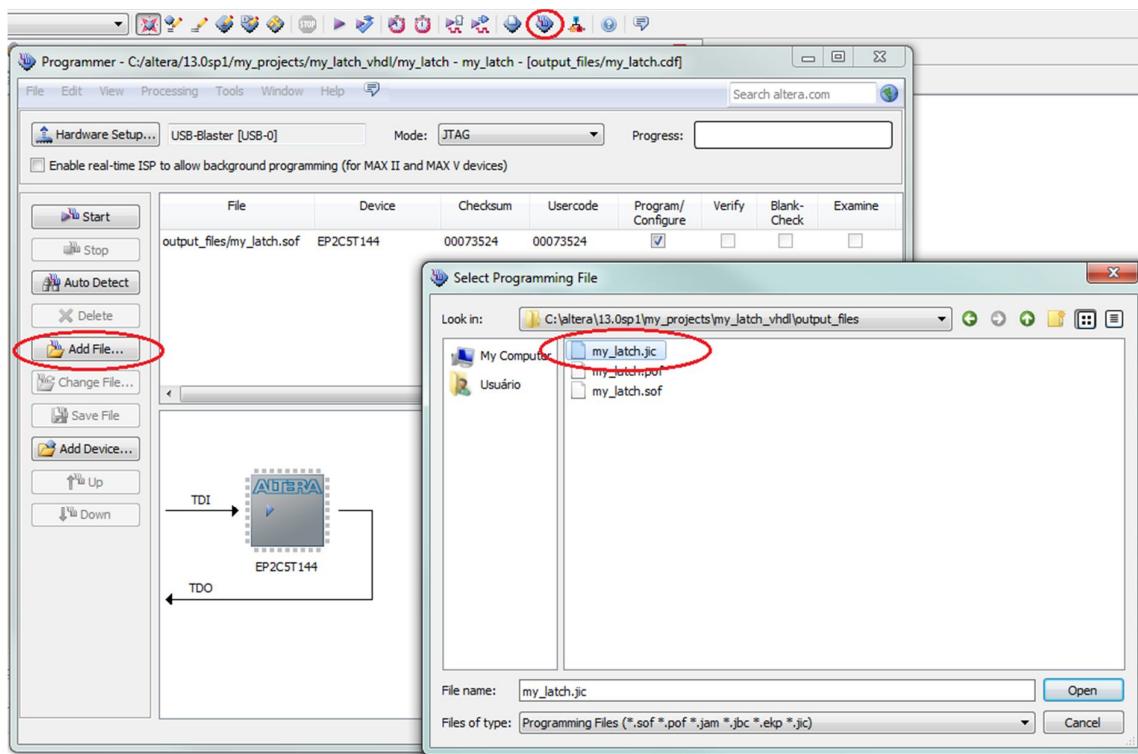


Figura 53: Selecionando o arquivo para a gravação.

Marque as duas opções conforme a Figura 54 e clique em “Start”.

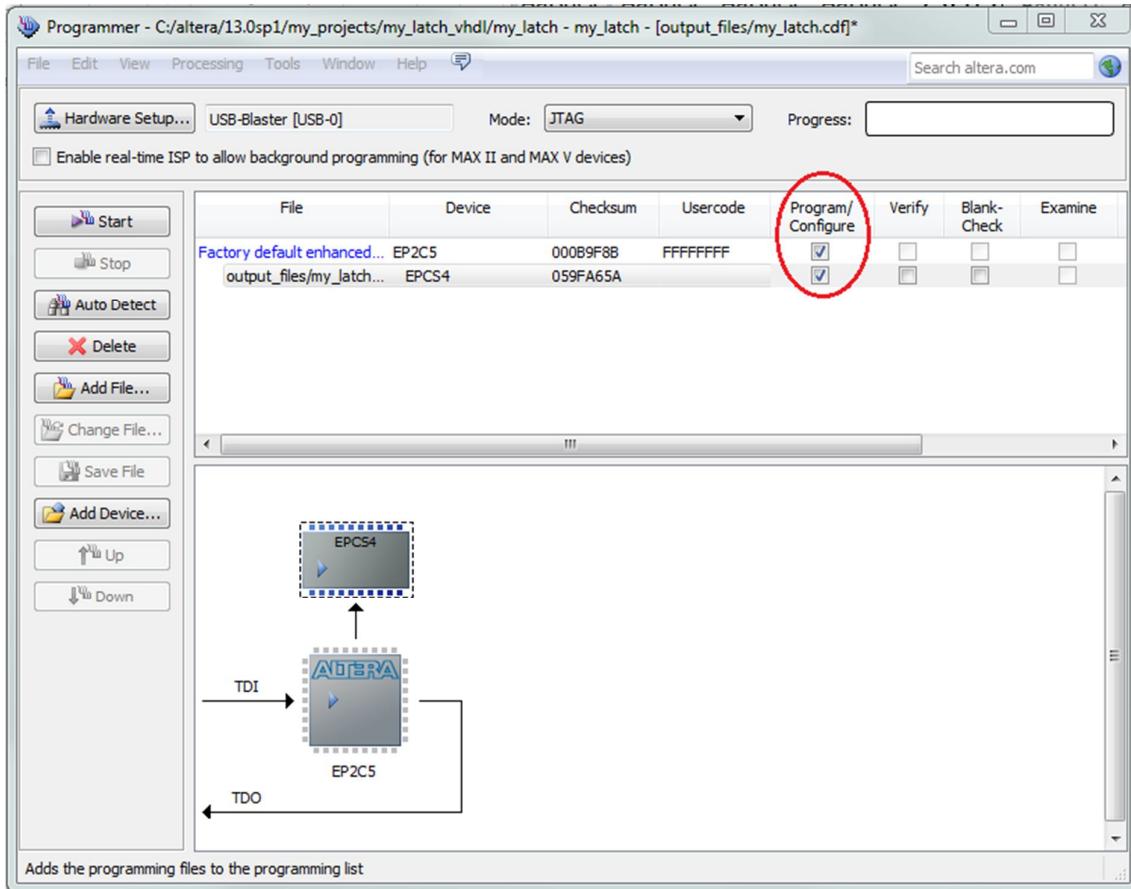


Figura 54: Configurando o programador.

Aguarde alguns segundos para que o processo de gravação seja bem sucedido, conforme Figura 55. Surgindo a mensagem, desligue a placa, desconecte o gravador e o seu kit estará pronto para o teste prático, demonstrado no próximo capítulo.

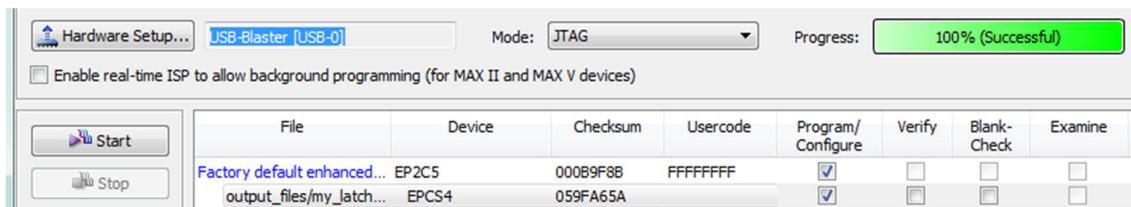


Figura 55: Gravação bem sucedida.

14. TESTE PRÁTICO DO CIRCUITO LATCH

Para realmente comprovar o funcionamento do seu projeto, deve-se efetuar o teste prático do circuito latch proposto. O diagrama esquemático para o latch pode ser apreciado na Figura 56, onde os pinos de entrada e saída do kit EE02-SOQ estão configurados conforme associações realizadas no capítulo 11 desta apostila.

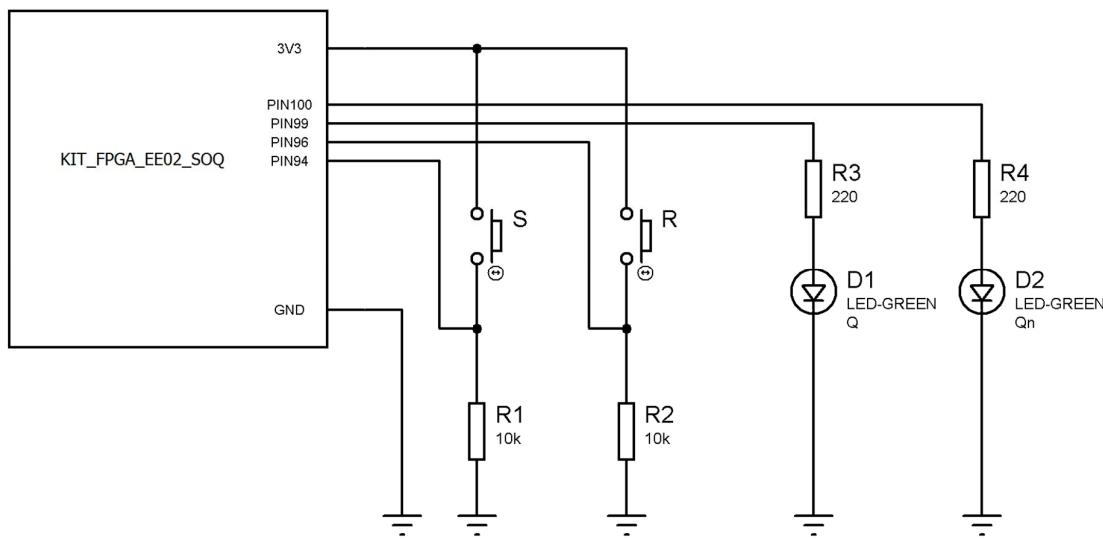


Figura 56: Diagrama esquemático para o teste da latch.

Como pode-se observar pelo diagrama esquemático da Figura 56, os LEDs foram conectados, através de resistores limitadores de corrente de 220Ω , às respectivas saídas associadas aos pinos 99 e 100 do kit FPGA. Nível alto fará com que o LED seja acionado. Duas chaves tátteis foram utilizadas para representar as entradas S e R. Os resistores R1 e R2 desempenham o papel de resistores de pull-down, mantendo nível lógico baixo nas entradas 94 e 96. Dessa forma, pressionando-se o botão S estaremos setando o flip-flop e pressionando o R estaremos resetando, e a resposta das saídas deverá ser conforme comportamento previsto para uma latch. O kit apresenta dois conectores GND e dois 3V3 para expandir a alimentação. Os componentes extra estarão ligados em uma protoboard, conforme Figura 57.

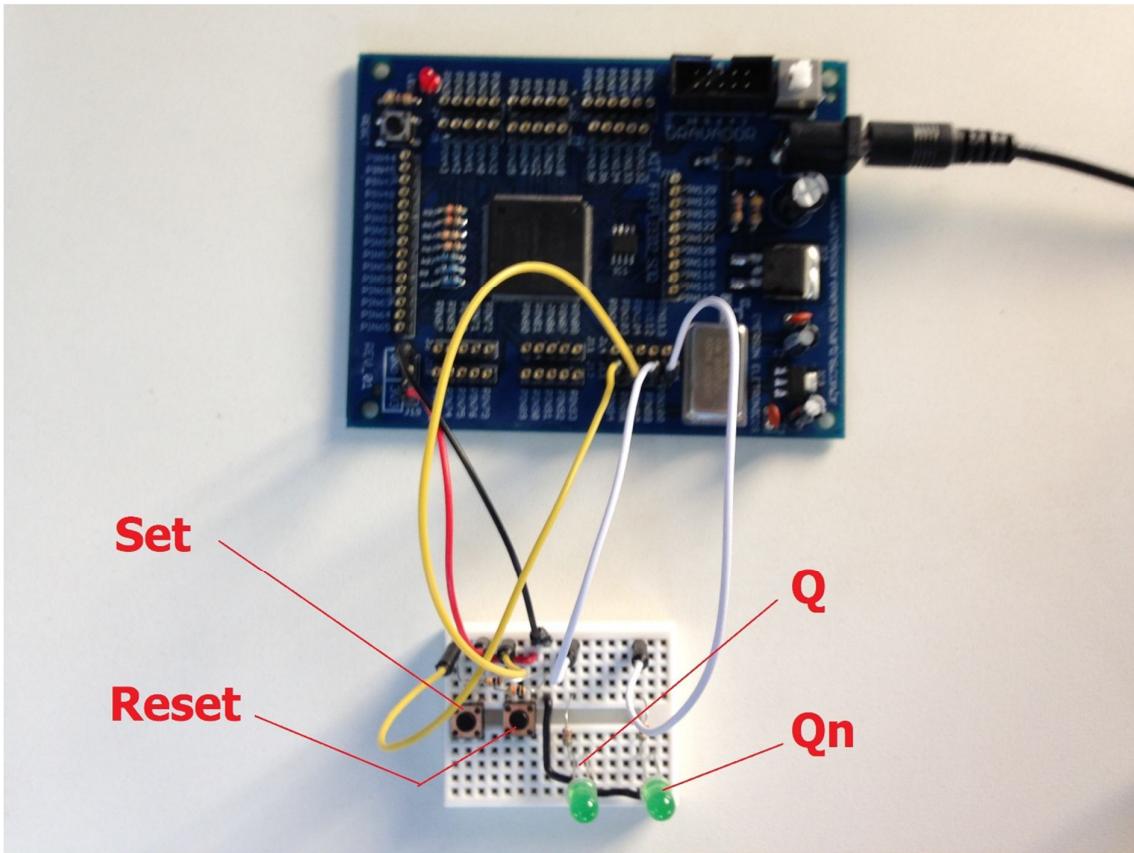


Figura 57: Circuito real montado para o teste prático.

Na imagem da Figura 57 estão assinaladas as entradas e saídas da latch. Após conferir criteriosamente toda sua montagem, verificando se não há nenhum erro recorrendo, ligue o circuito. Ao pressionar o botão de Set deverá constatar que apenas o LED Q ficará aceso (Figura 58).

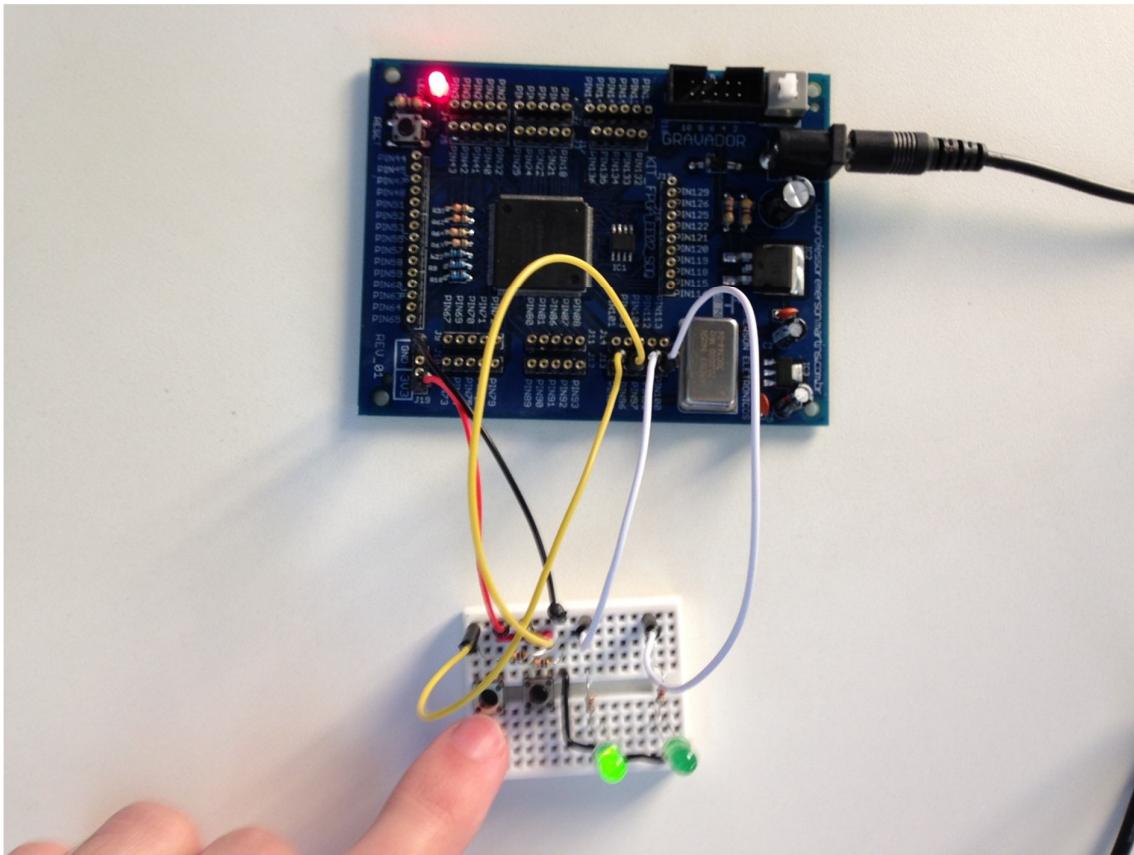


Figura 58: Teste do botão de Set.

Ao pressionar o botão de Reset, deverá constatar que apenas o LED Qn estará aceso, comprovando o funcionamento do projeto, bem como conversão e gravação bem sucedida do arquivo .jic em seu kit FPGA (Figura 59).

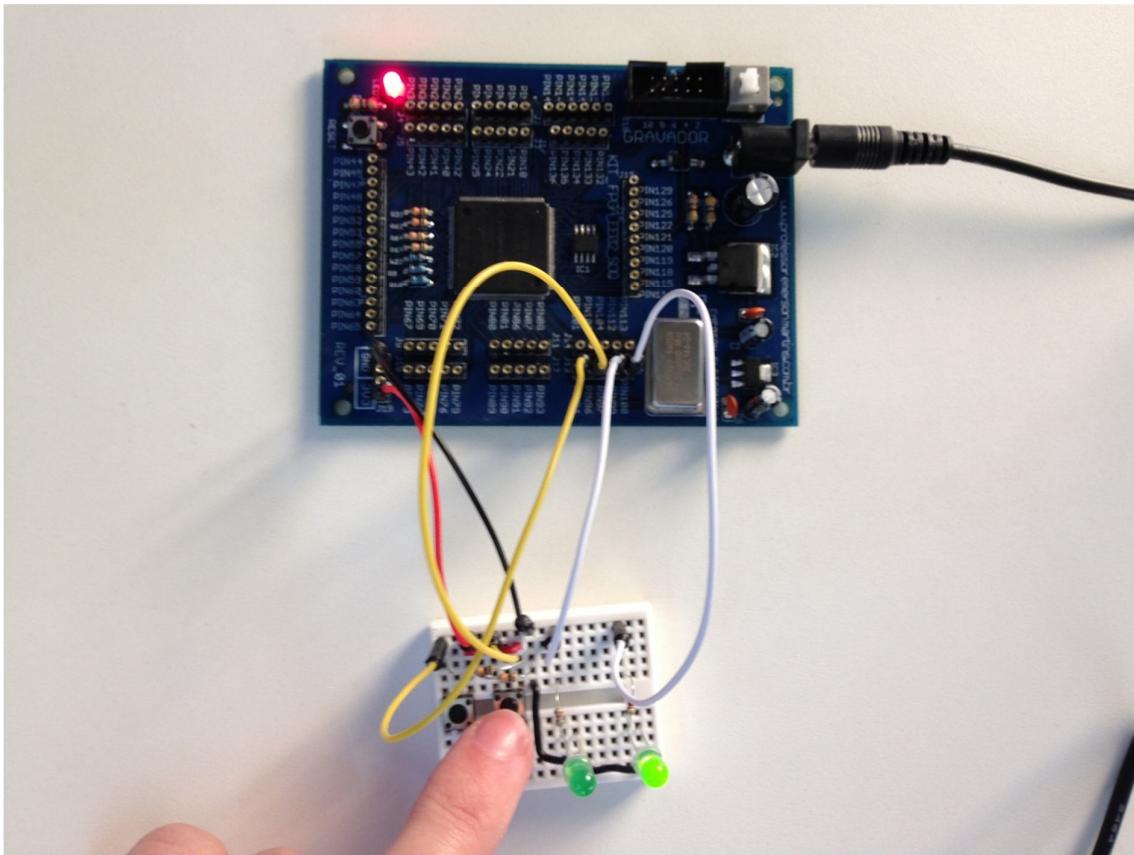


Figura 59: Teste do botão de Reset.

Comprovado o funcionamento, agora é só partir para projetos mais interessantes que estão disponíveis nos anexos, e também para seus próprios projetos, que serão possíveis com o aprendizado adquirido ao longo do curso de FPGA ministrado pelo WR Kits Channel.

CONCLUSÃO

A presente apostila apresentou de caráter introdutório, a inserção do usuário ao mundo dos FPGAs, falando um pouco sobre VHDL e Verilog, sobre as principais ferramentas do compilador Quartus II bem como explicação de sua utilização, dando ênfase ao kit EE02-SOQ desenvolvido pelo Instituto de Tecnologia Emerson Martins.

Para um aprendizado completo, deve-se ler e reler a presente obra, aprofundando-se nos exercícios, bem como análise, síntese e compilação dos arquivos Verilog e diagrama esquemático, visto que abordou-se apenas o exemplo de simulação e testes práticos com o arquivo VHDL.

Aconselha-se também o acompanhamento detalhado do curso de FPGA, ministrado pelo WR Kits Channel, onde eventuais dúvidas poderão ser esclarecidas nas aulas e através da seção de comentários.

ANEXOS

Os anexos da presente apostila estão disponíveis em arquivos individuais na pasta das mesma, e trazem novos projetos em VHDL para o leitor que deseja se aprofundar ainda mais, como multiplexadores e demultiplexadores, codificadores e decodificadores, bem como a introdução à simulação em um bancada de testes virtual, através do software ModelSIM da Altera.