UNIVERSIDADE FEDERAL DE LAVRAS

**Departamento de Ciência da Computação**

**Prof. Bruno de Abreu Silva**

**GCC260 – Laboratório de Circuitos Digitais**

*Introdução ao Verilog e software Quartus Prime*

## 1. Objetivos

- Conhecer a linguagem Verilog e suas características básicas;
- Utilizar o Quartus Prime para projeto de circuitos em Verilog e aprender novos recursos do software.

## 2. Verilog

Projetar e manter circuitos grandes por meio de diagrama de blocos pode ser uma tarefa complexa. Além disso, é mais interessante fornecer formas mais abstratas de descrever circuitos, como em termos de comportamentos e não simplesmente portas lógicas e suas conexões. Nesse sentido, diversas linguagens de descrição de hardware (*Hardware Description Languages* – HDLs) foram criadas. Dentre as mais usadas na indústria e nas universidades, podemos citar VHDL e Verilog.

Verilog foi desenvolvida por Prabhu Goel e Phil Moorby para a empresa *Gateway Design Automation* no início na década de 80 para ser usada para modelagem, simulação e síntese de circuitos digitais. Foi adquirida pela empresa CADENCE em 1985, tornada pública em 1990 e padronizada pelo IEEE em 1995. Atualizações foram feitas no padrão em 2001 e 2005.

Nesta prática, o objetivo não é realizar uma apresentação completa da linguagem e sim fornecer condições básicas para iniciar os estudos da linguagem. Consulte os materiais disponibilizados no Campus Virtual para aprender mais sobre Verilog.

### 2.1. Características básicas

| | | |
|---|---|---|
| **Comentários** | São ignorados pelo compilador e são usados para melhorar a legibilidade do código e para documentação | *// Comentário de uma linha*<br>*/* Este é um comentário que pode continuar nas linhas seguintes */* |
| **Números** | São definidos como número de bits, podem ser especificados em binário, octal, decimal ou hexadecimal | 3'b001<br>5'b11110<br>16'h5ED4<br>16'd24276<br>23 *// 32-bit decimal* |

| | | |
|---|---|---|
| **Identificadores** | Verilog é *case-sensitive*. Identificadores são definidos pelo usuário para variáveis, funções, módulos. Começam com uma letra ou *undescore*). | adder<br>by_8_shifter<br>_ABC_ */* diferente de _abc_ */*<br>Read_ *// usado para NOT read*<br>nRead *// usado para NOT read* |
| **Operadores** | Podem ser unários, binários ou ternários | >, +, ~, &, !=. |
| **Palavras Reservadas** | Possuem significado especial e não podem ser identificadores | **assign**, **case**, **while**, **wire**, **reg**, **and**, **or**, **nand**, **module**. |
| **Portas lógicas** | Podem ter uma saída e várias entradas. As palavras reservadas são: **and**, **nand**, **or**, **nor**, **xor**, **xnor** e **not**. | **and** c1(o, a, b, c, d);<br>**or** c2(o, a, b);<br>**not** c3(o, a); |
| **Valores** | Valores básicos para os tipos de dados | 0 (nível lógico zero)<br>1 (nível lógico 1)<br>X (nível desconhecido)<br>Z (alta impedância) |
| **Wire** | Representa um fio usado para conectar componentes | **wire** c; *// um simples fio*<br>**wire** [9:0] A; *// vetor de 10 fios* |
| **Reg** | Declarado para todos os objetos de dados do lado esquerdo das atribuições em blocos **initial** ou **always** | **reg** a;<br>**reg** [7:0] tom;<br>**reg** [5:0] b, c; |
| **Input, Output, Inout** | Usados para declarar entradas e saídas de um módulo ou tarefa. | **module** sample(b, e, c, a);<br>**input** a;<br>**output** b, e;<br>**output reg** [1:0] c; |
| **Operadores aritméticos** | + (adição)<br>- (subtração)<br>* (multiplicação)<br>/ (divisão)<br>% (módulo) | f = a + b;<br>g = c – n;<br>count = (count + 1) % 16; |
| **Operadores relacionais** | < (menor que)<br><= (menor ou igual a)<br>> (maior que)<br>>= (maior ou igual a)<br>== (igual a)<br>!= (diferente de) | **if**(x == y) e = 1;<br>**else** e = 0; |
| **Operadores bit-a-bit** | ~ (not bit-a-bit)<br>& (and bit-a-bit)<br>\| (or bit-a-bit)<br>^ (xor bit-a-bit)<br>~^ ou ^~ (xnor bit-a-bit) | **module** and2(a, b, c);<br>**input** [1:0] a, b;<br>**output** [1:0] c;<br>**assign** c = a & b;<br>**endmodule** |
| **Operadores lógicos** | ! (not lógico)<br>&& (and lógico)<br>\|\| (or lógico) | **if**((x == y) && (z)0 a = 1;<br>**else** a = !x; |
| **Operador de concatenação** | Combina dois ou mais operandos para formar um vetor maior | **wire** [1:0] a, b; **wire** [2:0] x;<br>**wire** [3:0] y, Z;<br>**assign** x = {1'b0, a};<br>**assign** y = {a, b};<br><br>**assign** {cout, y} = x + Z; |
| **Operador condicional** | Avalia uma de duas expressões baseado em uma condição | **assign** a = (g) ? x : y;<br>**assign** a = (inc == 1) ? a+1 : a-1 |

## 2.2. Descrição Estrutural

A descrição estrutural é feita pela declaração das portas lógicas, e realização de suas interconexões. Veja a descrição para o circuito referente à expressão *Y = ~AB + C*.

```
module circuito
   (
      input wire A, B, C, // Linhas de entrada
      output wire Y // Linha da saída
   );

   wire NA, NAB; // Sinais internos ao módulo

   not P1(NA, A);
   and P2(NAB, NA, B);
   or P3(Y, NAB, C);

endmodule
```

## 2.3. Descrição por Fluxo de Dados

Nesse tipo de descrição, não é necessário listar as portas lógicas e suas conexões, mas sim as operações lógicas implementadas, por exemplo, as expressões em termo de produto geradas pelo mapa de Karnaugh. Assim, o projetista não precisa se preocupar com o circuito gerado e a própria ferramenta de síntese irá definir. É uma forma de descrição mais abstrata que a estrutural. Veja a descrição para o circuito referente à mesma expressão anterior *Y = ~AB + C*. Após o **assign** é descrita a função. Outras funções poderiam ser declaradas em diferentes comandos **assign**. Todas as funções executam de forma paralela, ou seja, concorrentemente.

```
module circuito
   (
      input wire A, B, C, // Linhas de entrada
      output wire Y // Linha da saída
   );

   assign Y = (~A & B) | C;

endmodule
```

## 2.4. Descrição Comportamental

A descrição comportamental apresenta o comportamento esperado para o circuito. É um nível mais abstrato e de mais alto nível de descrição. Veja a descrição para o circuito referente à mesma expressão anterior *Y = ~AB + C*.

```
module circuito
   (
      input wire A, B, C, // Linhas de entrada
      output wire Y // Linha da saída
   );

   always@(A, B, C) begin
      Y = (~A & B) | C;
   end
endmodule
```

### 3. Verilog e o software Quartus Prime

O software Quartus Prime permite a utilização da linguagem Verilog para a implementação de circuitos. Para isso, em vez de usar o diagrama de blocos, deve-se criar um arquivo em Verilog. A seguir, é descrito o passo a passo para isso, apenas ressaltando as diferenças em relação ao projeto desenvolvido na Prática 1.

a. Crie um projeto no Quartus Prime para o kit DE10-Lite (em caso de dúvida, consulte o tutorial da Prática 1). **Após criar o projeto, não crie o arquivo de diagrama de blocos.**
b. Observe que já foi criado, juntamente com o projeto, um arquivo em Verilog chamado **DE10_LITE_Golden_Top.v**. Abra este arquivo em *File->Open...* e veja que ele já faz a declaração de todas as entradas (*inputs*) e saídas (*outputs*) do kit DE10_LITE.
c. Para criar mais arquivos em Verilog, clique em *File->New...* e escolha *Verilog HDL File*. Será aberto um editor de texto para que seja digitada a descrição de hardware na linguagem Verilog. Porém, nesta prática, não será necessário criar mais arquivos.
d. Dicas ao implementar seu projeto usando o arquivo em Verilog:
    - Complete o código DE10_LITE_Golden_Top.v adicionando a expressão lógica no local apropriado de acordo com o problema descrito na Seção 4.
    - Clique em *Processing->Start->Start Analysis & Elaboration*. Como a compilação é demorada, o Quartus Prime permite realizar compilações parciais. Essa opção é somente para verificar se há erros da linguagem no código, como falta de ponto-e-vírgula, declarações de identificadores etc. Use-a enquanto estiver desenvolvendo seu código, pois é uma compilação mais rápida.
    - Em seguida, clique em *Tools->Netlist Viewers->RTL Viewer* e analise o resultado. Essa ferramenta permite que seja visualizado pelo projetista qual foi o circuito “entendido” pelo Quartus Prime a partir da descrição do hardware. É uma ferramenta muito importante para identificação e correção de erros no projeto.
    - Outra ferramenta interessante que permite a visualização do circuito é acessada em *Tools->Netlist Viewers->Technology Map Viewer (Post-Mapping)*. Enquanto o *RTL Viewer* é apenas uma

representação gráfica, essa ferramenta só é acessível após a compilação completa e mostra como o circuito será de fato implementado no FPGA.

e. Após finalizar a descrição do circuito em Verilog, faça a compilação completa clicando em *Processing->Start Compilation*.
f. A etapa de configuração (programação) da FPGA é exatamente como descrito na Prática 1.

## 4. Proposta de atividade

Um sistema de reservatórios tem 4 sensores de nível, representados por 4 chaves da placa DE10_LITE (SW[3], SW[2], SW[1] e SW[0]). A bomba que alimenta esses reservatórios deve ser acionada se nenhum dos sensores estiver acionado ou se somente um dos sensores estiver acionado. Obtenha a expressão lógica que determine o acionamento da bomba. Implemente em Verilog e programe a FPGA usando o software Quartus Prime. Considere que o acionamento da bomba é representado por um dos LEDs vermelhos (LEDR[0]).