Unidade 3.1

Laboratório – Simuladores
(Introdução)

Prof. Rogério D. Dantas

Curso: *Engenharia de Computação*

Agenda deste labotarório:

- Preparação do ambiente de programação no Windows (Instalações);
  - VSCode e extensão (DigitalJS e Verilog HDL) ;
  - Icarus Verilog;
  - GTKWave;

- Uma linguagem desse tipo expressa as noções primárias de hardware: sua sintaxe e semântica permitem descrever o fluxo de dados e a temporização dos circuitos digitais, bem como a manipulação de fios para a conexão, construção de hierarquias e exploração da concorrência entre diversos circuitos digitais.

- Existem diversas linguagens de descrição de hardware: VHDL, Verilog, AHDL, SDL, SystemC, Handel-C, Esterel etc. As mais empregadas no mundo, atualmente, são o VHDL e o Verilog.

Preparação do ambiente de programação:

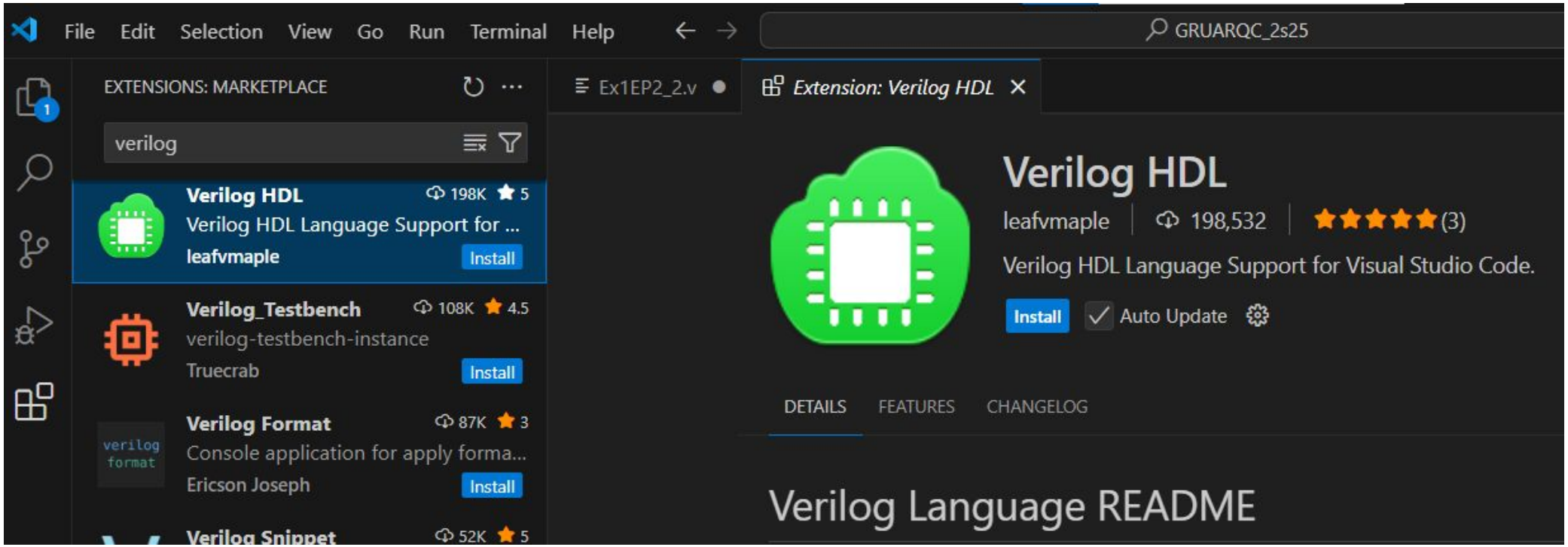
- Para as próximas aulas, precisamos instalar alguns softwares para desenvolver a descrição do Hardware:

- 1) ***VsCode***, que é uma IDE (Interface Integrada de Desenvolvimento), que servirá para digitar e sintetizar o código em Verilog;
- 2) ***Icarus Verilog*** é um compilador de Verilog que permite compilar e simular circuitos descritos em Verilog. O Icarus Verilog é uma ferramenta de código aberto e pode ser utilizada em sistemas operacionais Windows, Linux e MacOS.

- 3) ***GTKWave*** é uma ferramenta open source para visualização de formas de onda (waveforms), muito usada em projetos de hardware digital (como em Verilog ou VHDL)

Preparação do ambiente de programação (*Instalando o DigitalJS e Verilog HDL*):

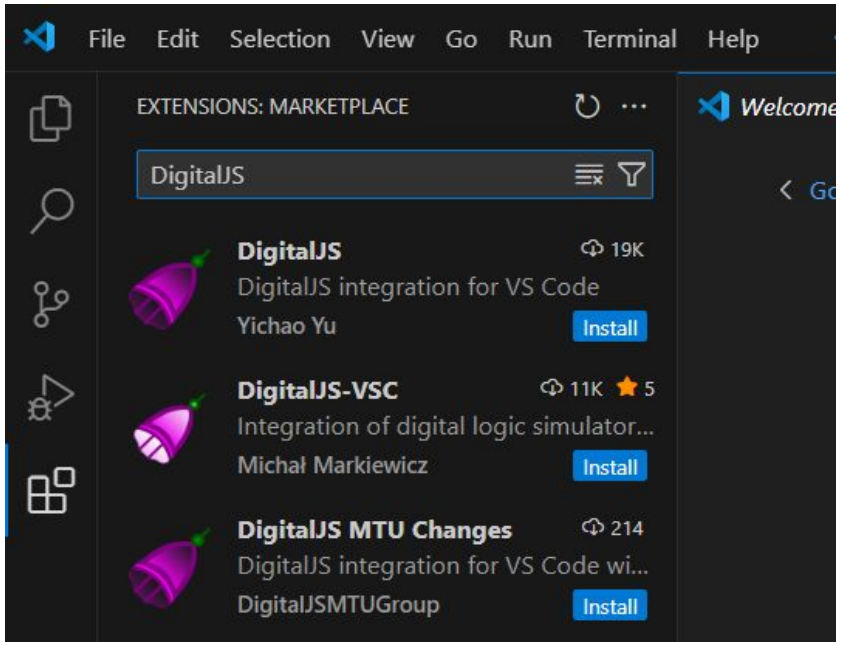
- Ainda sobre o VsCode, vamos instalar outro plugin, o Verilog HDL que nos ajuda



Preparação do ambiente de programação (*Instalando o DigitalJS e Verilog HDL*):

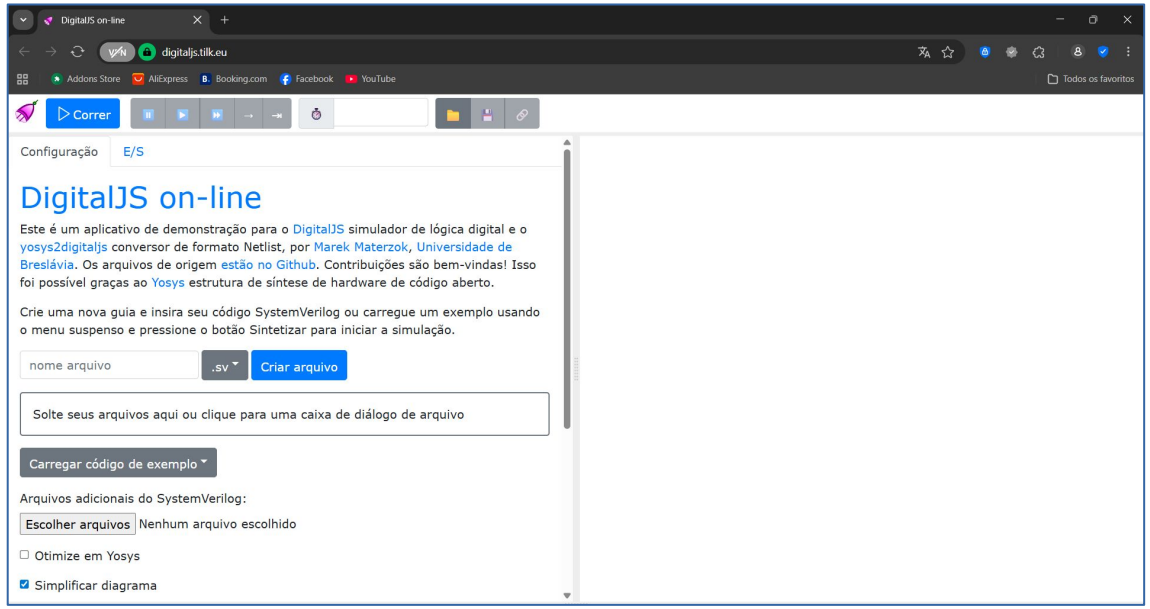
- Você pode utilizar o **DigitalJS** diretamente no seu navegador, sem necessidade de instalação através da versão ***online do DigitalJS***, entretanto, para essa disciplina recomendamos utilizar um plugin do VSCode que deve ser instalado conforme a figura ao lado:

- Clique no botão Instalar e pronto. O **DigitalJS** estará disponível no seu VSCode. Note que o ícone da figura acima aparecerá no lado direito das abas dos editores de texto e essa é a forma mais simples de abrir um editor.



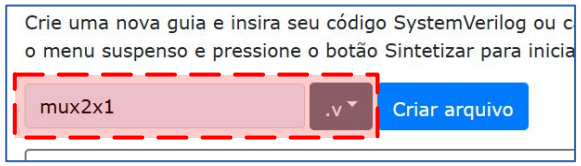
Testando a versão Online do DigitalJS:

- Basta acessar o link ***online do DigitalJS***:

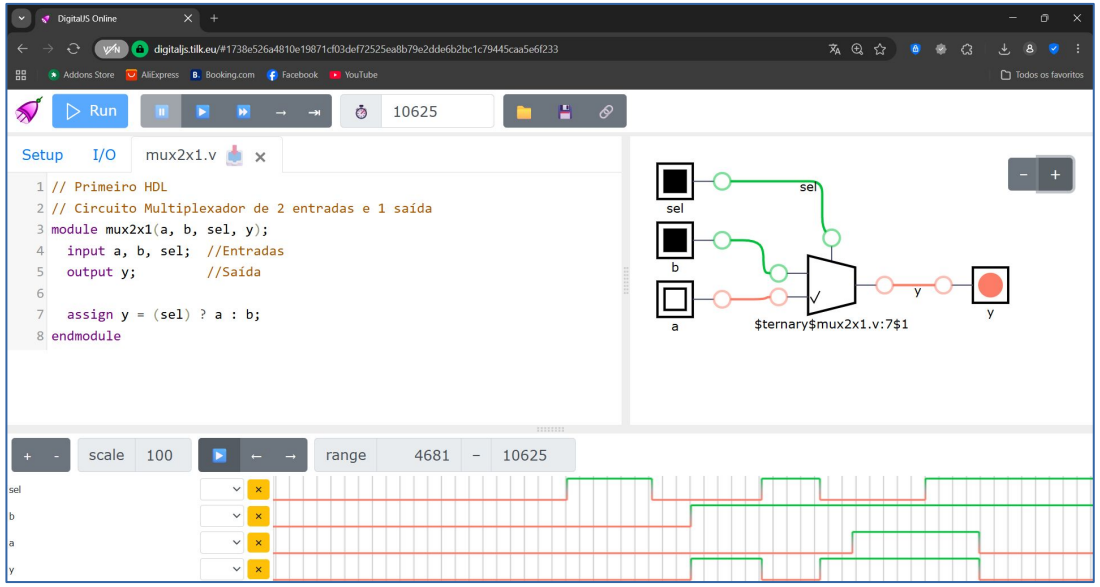


Basta digitar o código Verilog, e clicar em no botão ***Run***!

Ao abrir o site, basta escreve o **nome do arquivo *mux2x1*** e mudar e extensão para ***.v***:

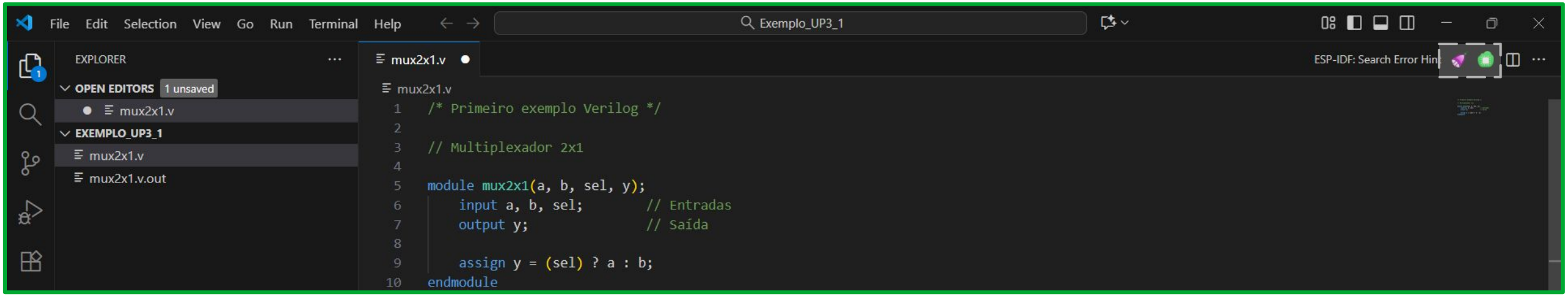


Clicando em ***Criar Arquivo***.



Preparação do ambiente de programação (*Instalando o DigitalJS e Verilog HDL*):

- Após a instalação dos dois plugins (***DigitalJS*** e ***Verilog HDL***), os ícones ficaram disponíveis na parte superior esquerda do VsCode.



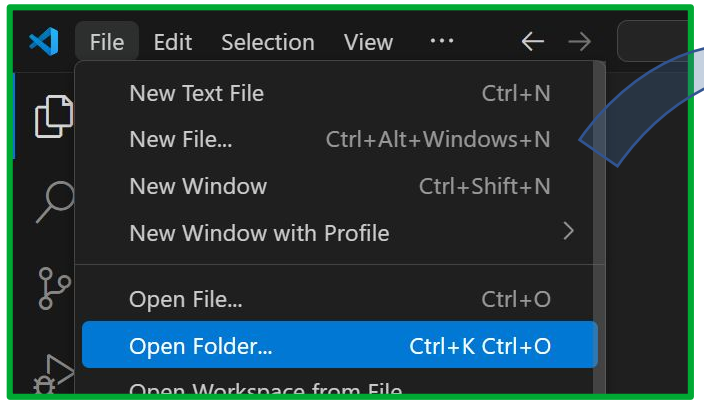
Agora basta abrir o VsCode!

Testando o DigitalJS com o Verilog HDL no VsCode:

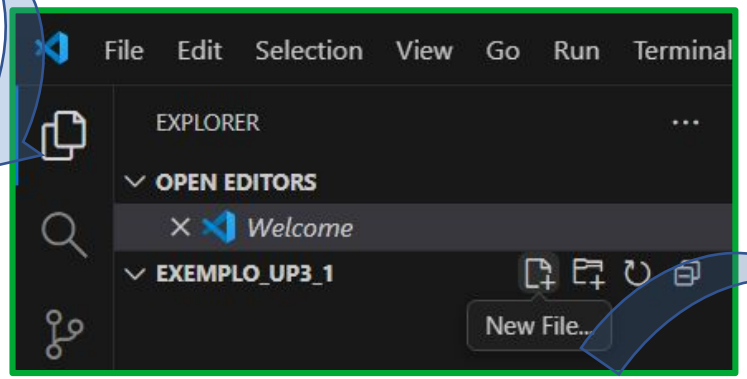
```
/* Primeiro exemplo Verilog */

// Multiplexador 2x1

module mux2x1(a, b, sel, y);
    input a, b, sel;        // Entradas
    output y;               // Saída

    assign y = (sel) ? a : b;
endmodule
```

Apenas como referência para nossas aulas, este será nosso primeira descrição de hardware a ser feito o ***mux2x1.v***. Nas próximas aulas iremos estudar mais detalhes sobre o código.

Testando o DigitalJS com o Verilog HDL no VsCode:

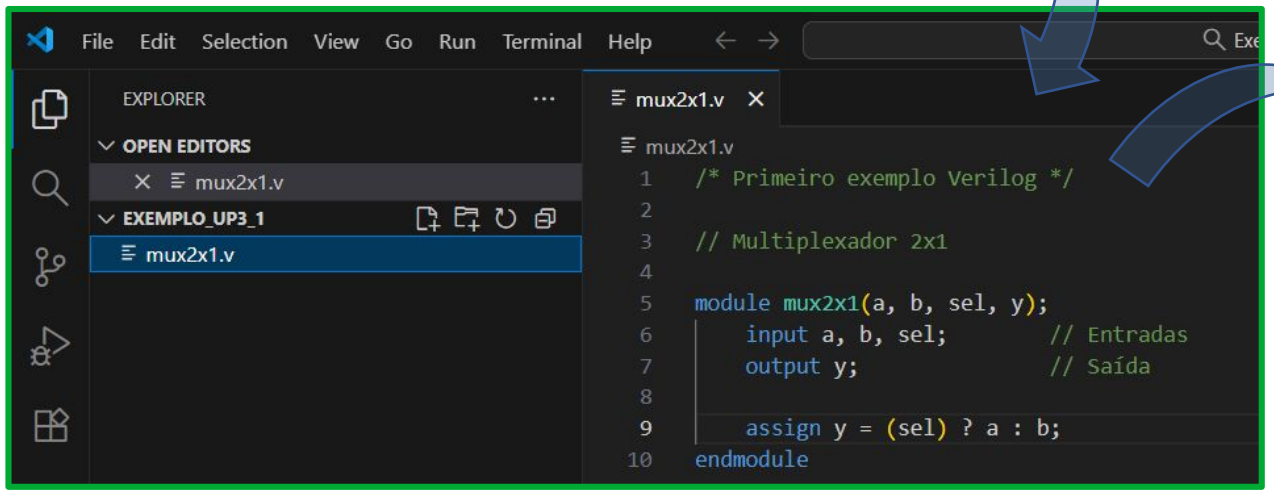


Abra a pasta onde você irá criar seu projeto!

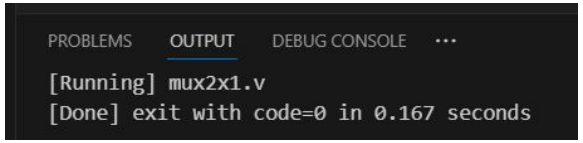


Crie o arquivo mux2x1.v

Digite as descrição do Hardware, e compile usando o botão

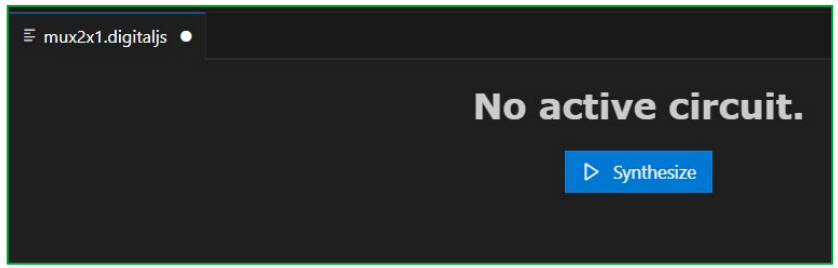


Saída após compilação

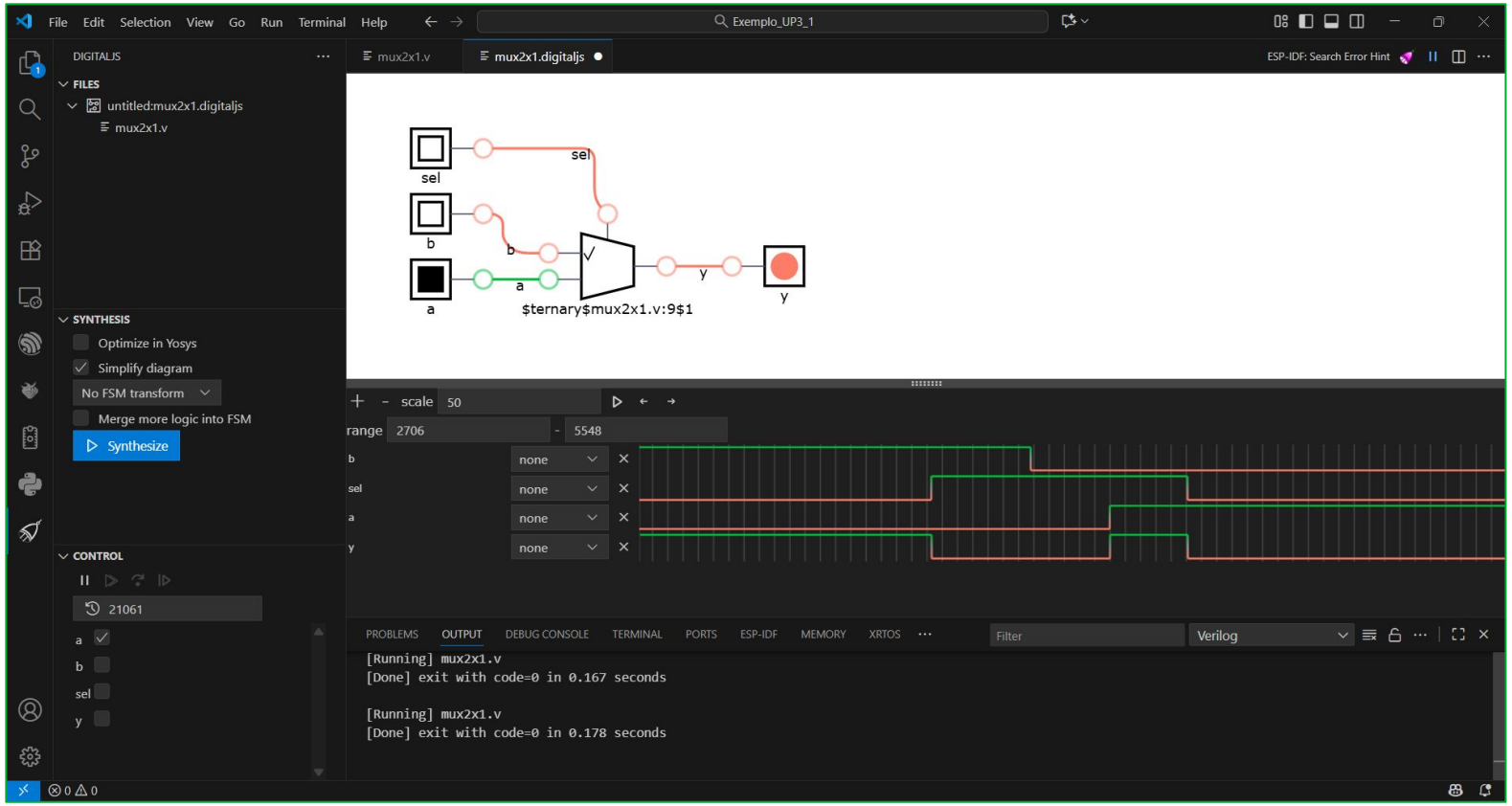


Testando o DigitalJS com o Verilog HDL no VsCode:

Agora vamos abrir o **DigitalJS** clicando em 



A tela a cima será aberta, basta clicar em ***Synthesize***, e aparecerá o circuito sintetizado pelo código Verilog do ***mux2x1.v***



Preparação do ambiente de programação (*Icaro Verilog*):

O Icarus no seu computador, segue as instruções conforme o seu Sistema Operacional:

- Windows → Baixe o arquivo de instalação do ***Icarus Verilog*** e instale normalmente, Abra o terminal do Windows e digite ***iverilog*** e pressione Enter. Se o Icarus estiver instalado corretamente, você verá uma mensagem de erro indicando que nenhum arquivo de entrada foi especificado.
- Linux (Ubuntu) → Abra o terminal e digite ***sudo apt-get install iverilog*** e pressione Enter. Após a instalação, digite ***iverilog*** e pressione Enter. Se o Icarus estiver instalado corretamente, você verá uma mensagem de erro indicando que nenhum arquivo de entrada foi especificado.
- MacOS (Homebrew) → Abra o terminal e digite ***brew install icarus-verilog*** e pressione Enter. Após a instalação, digite ***iverilog*** e pressione Enter. Se o Icarus estiver instalado corretamente, você verá uma mensagem de erro indicando que nenhum arquivo de entrada foi especificado.

Preparação do ambiente de programação (*GTKWave*):

O GTKWave é um visualizador de formas de onda utilizado para analisar os resultados de simulações de código em Verilog ou VHDL.

- Windows → Baixe o arquivo de instalação do ***GTKWave*** e instale normalmente, Execute o instalador e siga as instruções padrão de instalação (“Next”, “Install”, “Finish”). Após a instalação, abra o Prompt de Comando (CMD) e digite: ***gtkwave***
- Linux (Ubuntu) → Abra o terminal e digite ***sudo sudo apt install gtkwave*** e pressione Enter. Após a instalação, digite ***gtkwave*** e pressione Enter. Se o gtkwave estiver instalado corretamente, você verá a tela do GTKWave.
- MacOS (Homebrew) → Abra o terminal e digite ***brew install gtkwave*** e pressione Enter. Após a instalação, digite ***gtkwave*** e pressione Enter. Se o gtkwave estiver instalado corretamente, você verá a tela do GTKWave.

Testando do Icaro Verilog no VsCode:

- Abra o VsCode e vamos digitar dois códigos em Verilog, o primeiro será o ***mux2x1.v*** (já feito anteriormente no DigitalJS) e o segundo será o ***mux2x1_tb.v*** que é o código de testbench, código que serve para gerar os sinais de estimulos nas entradas do circuito a ser sintetizado e verificar o das saídas).

```
// mux2x1_tb.v
`include "mux2x1.v"
`timescale 1ns/100ps   // define unidade e precisão de tempo

module mux2x1_tb;
  reg a0, b0, sel0;     // sinais de entrada para o módulo
  wire y0;              // saída do módulo

  // instanciação do DUT (Device Under Test)
  mux2x1 uut (
    .a(a0),
    .b(b0),
    .sel(sel0),
    .y(y0)
  );
```

Código continua no próximo slide!

Testando do Icaro Verilog no VsCode:

- Continuação do código:

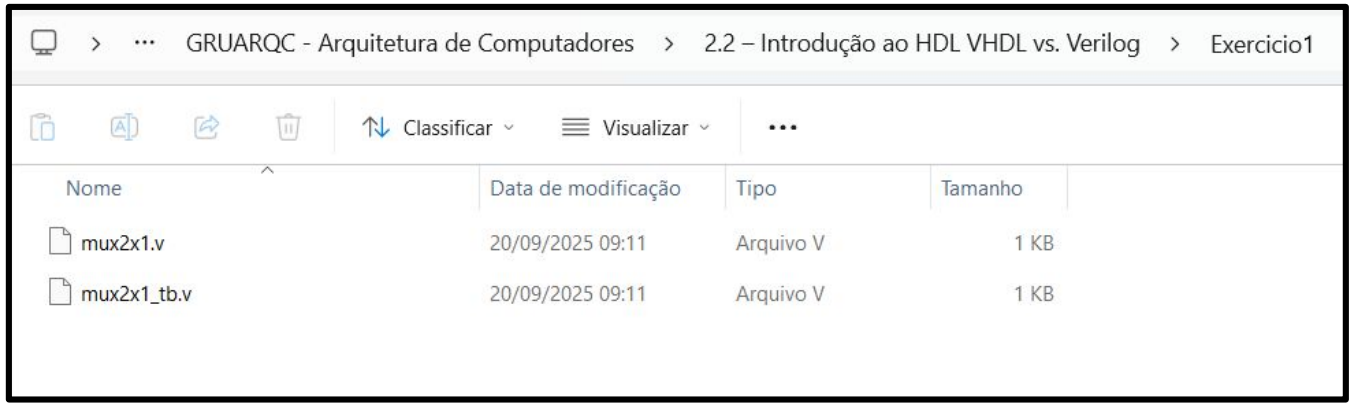
```
initial begin
    // cria arquivo para waveform (para GTKWave)
    $dumpfile("mux2x1.vcd"); // Arquivo onde serão salvas os dados para simulação
    $dumpvars(0, mux2x1_tb); // Salva as variáveis do simulador

    // Estímulos (Tabela verdade com todas as condições possíveis)
        a0 = 0; b0 = 0; sel0 = 0;
    #10 a0 = 0; b0 = 1; sel0 = 0;   // Após 10ns depois muda os valores
    #10 a0 = 0; b0 = 1; sel0 = 1;
    #10 a0 = 1; b0 = 0; sel0 = 0;
    #10 a0 = 1; b0 = 0; sel0 = 1;
    #10 a0 = 1; b0 = 1; sel0 = 0;
    #10 a0 = 1; b0 = 1; sel0 = 1;

    #20 $finish;                    // Após 20ns finaliza a simulação
  end

  // monitor para imprimir no console
  initial begin
    $monitor("Tempo=%0t | a0=%b b0=%b sel0=%b => y0=%b", $time, a0, b0, sel0, y0);
  end
```

Testando do Icaro Verilog no VsCode:

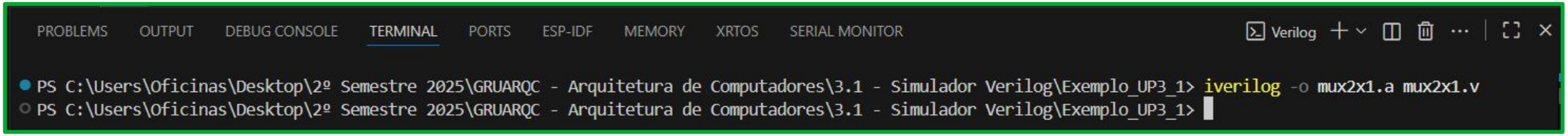
- **Conceitos importante que iremos utilizar agora:**
- **.v** → arquivo-fonte Verilog (módulo ou testbench).
- **iverilog** → compilador para simulação (Icarus Verilog). Ele não produz hardware sintetizável; converte seu código Verilog num executável/simulador.
- **.vvp** (ou qualquer nome de saída) → binário/objeto de simulação gerado por iverilog.
- **vvp** → executa o arquivo gerado pelo iverilog (roda a simulação).
- **.vcd** → arquivo de dump de formas de onda (***Value Change Dump***) que o testbench pode gerar; abre-se com **gtkwave**.
- **gtkwave** → visualizador de formas de onda.
- **Observação importante:** alguns textos chamam "compilar para síntese" — correção: iverilog compila para simulação; síntese é feita por ferramentas específicas (e.g. Xilinx Vivado, Intel Quartus).

Testando do icaro Verilog no VsCode:

- Com os dois arquivos criados e salvos:



- Agora no VSCode, abra o **Terminal (Ctrl+Shift)** e digite o seguinte comando:
  - **iverilog -o <nome_do_arquivo_saida.a> <arquivos_verilog.v>**
  - **» iverilog -o mux2x1.a mux2x1.v**
  - Obs.: Não esqueça que no terminal você precisa esta no diretório onde salvo os códigos!

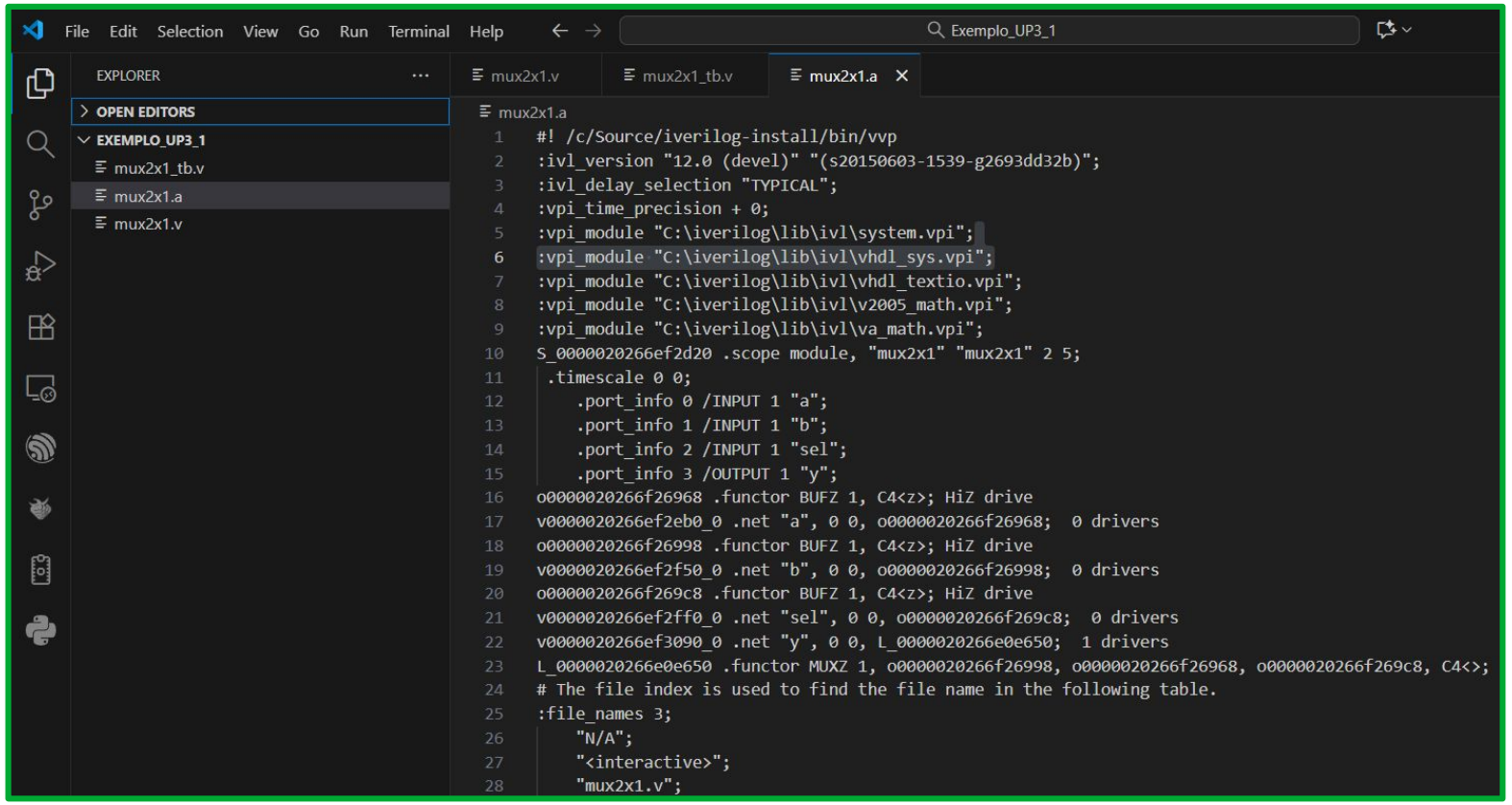


Testando do icaro Verilog no VsCode:

- Após executar o comando anterior, teremos o ***mux2x1.a***:

O que acontece quando você executa o comando:

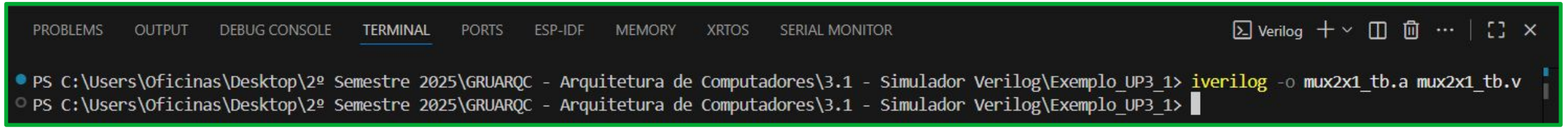
- O **Icarus Verilog** lê o arquivo **mux2x1.v**;
- Verifica sintaxe, módulos, conexões e tipos;
- Gera um arquivo objeto (por padrão, chamado a.out) contendo o circuito compilado;
- Esse arquivo pode ser executado com o comando v:



Testando do icaro Verilog no VsCode:

- Vamos realizar a compilação para o arquivo de testbench:
  - **iverilog -o <nome_do_arquivo_saida_tb.a> <arquivos_verilog_tb.v>**
  - **» iverilog -o mux2x1_tb.a mux2x1_tb.v**

  - Obs.: Não esqueça que no terminal você precisa esta no diretório onde salvo os códigos!

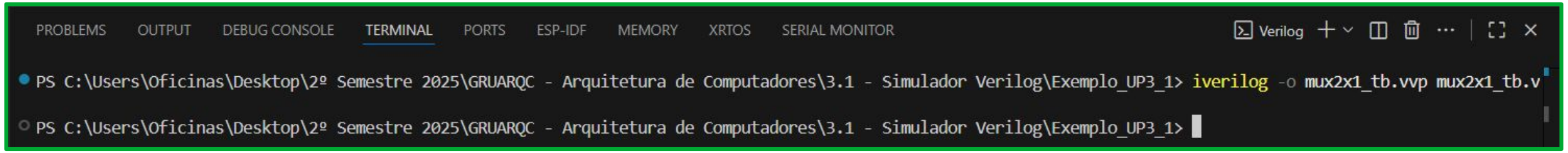


Como resultado, tem-se o arquivo **mux2x1_tb.a**

Testando do icaro Verilog no VsCode:

- Vamos realizar a compilação para o arquivo de testbench:
  - **iverilog -o <nome_do_arquivo_saida_tb.vvp <arquivos_verilog_tb.v>**
  - **» iverilog -o mux2x1_tb.vpp mux2x1_tb.v**

  - Obs.: Não esqueça que no terminal você precisa esta no diretório onde salvo os códigos!



Como resultado, tem-se o arquivo **mux2x1_tb.vvp**

Testando do icaro Verilog no VsCode:

- Agora será necessário gerar o arquivo de simulação com vvp:
  - **vvp <arquivos_verilog_tb.vvp>**
  - **» vvp mux2x1.vpp**

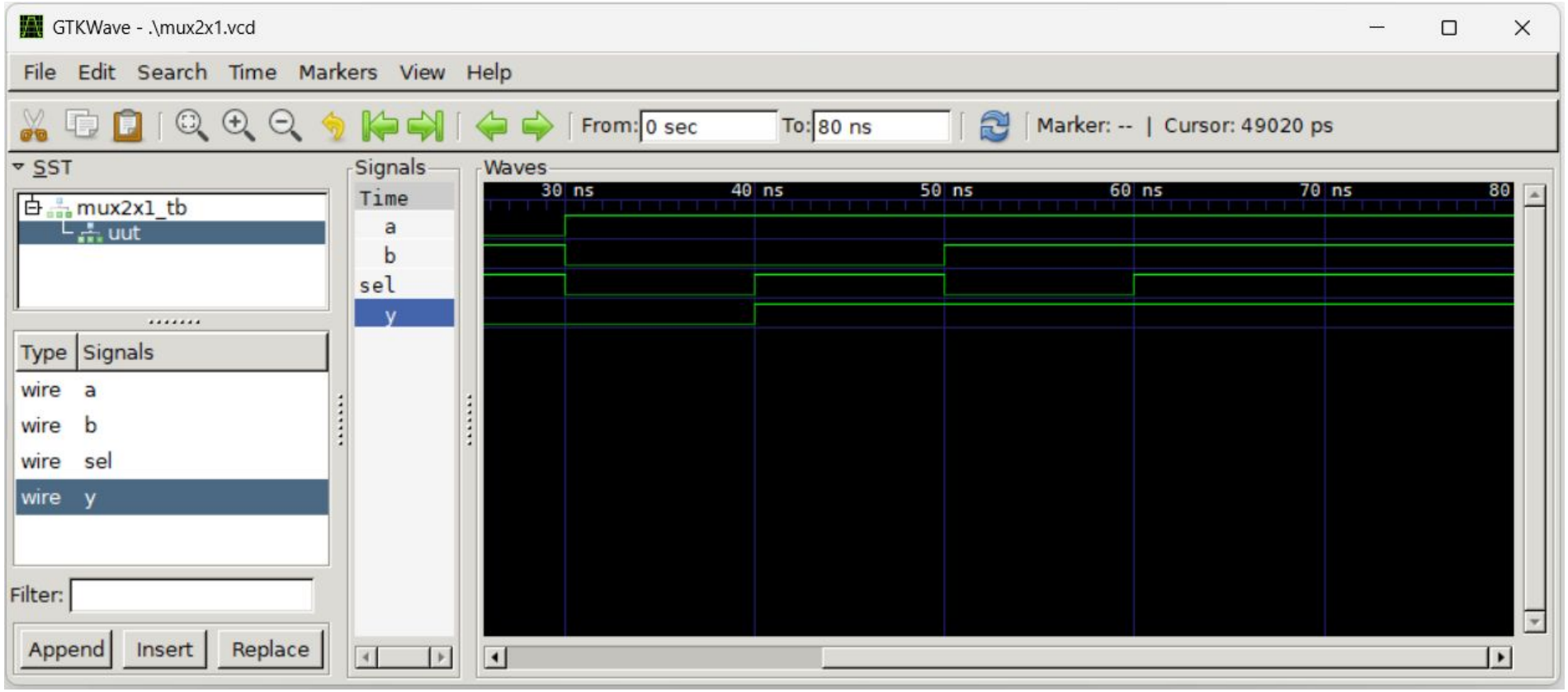
  - Obs.: Não esqueça que no terminal você precisa esta no diretório onde salvo os códigos!

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ESP-IDF  MEMORY  XRTOS  SERIAL MONITOR        Verilog

PS C:\Users\Oficinas\Desktop\2º Semestre 2025\GRUARQC - Arquitetura de Computadores\3.1 - Simulador Verilog\Exemplo_UP3_1> vvp mux2x1.vvp
VCD info: dumpfile mux2x1.vcd opened for output.
Tempo=0 | a0=0 b0=0 sel0=0 => y0=0
Tempo=100 | a0=0 b0=1 sel0=0 => y0=1
Tempo=200 | a0=0 b0=1 sel0=1 => y0=0
Tempo=300 | a0=1 b0=0 sel0=0 => y0=0
Tempo=400 | a0=1 b0=0 sel0=1 => y0=1
Tempo=500 | a0=1 b0=1 sel0=0 => y0=1
Tempo=600 | a0=1 b0=1 sel0=1 => y0=1
mux2x1_tb.v:31: $finish called at 800 (100ps)
PS C:\Users\Oficinas\Desktop\2º Semestre 2025\GRUARQC - Arquitetura de Computadores\3.1 - Simulador Verilog\Exemplo_UP3_1>
```

Como resultado, tem-se o arquivo **mux2x1.vvp**

Testando do icaro Verilog no VsCode:

- Agora vamos abrir o GTKWave para visualizar a carta de tempo do circuito descrito em Verilog:
  - **gtkwave <arquivos_verilog.vdc>**
  - » **gtkwave mux2x1.vdc**



Compilando o código em Verilog:

- Resumindo, no VSCode será digitado os seguintes comandos para o dois arquivos criados para o Multiplexador de duas entradas e uma saída o Mux2x1 (mux2x1.v e mux2x1_tb.v):

- » **iverilog -o mux2x1.v**
  - **Realizou a compilação do código de síntese do circuito multiplexador**
- » **iverilog .\mux2x1_tb.v**
  - **Realizou a compilação do código de simulação do circuito multiplexador**
- » **iverilog -o .\mux2x1.vvp .\mux2x1_tb.v**
  - **Gera o arquivo para simulação vvp.**
- » **vvp .\mux2x1.vvp**
  - **Gera o arquivo .gvd para simulação no GTKWave.**
- » **gtkwave .\mux2x1.vcd**
  - **Abre o simulado GTKWave.**

Conceitos Básicos de Eletrônica Digital:

- ***Portas Lógicas:*** Apresente as portas fundamentais (AND, OR, NOT, XOR, NAND, NOR).
  - Exemplo: Porta AND – Saída é 1 apenas se todas as entradas forem 1.
- ***Circuitos Combinacionais:*** Saídas dependem apenas das entradas atuais (ex.: multiplexador, somador).

Níveis de Abstração em Verilog:

- **Estrutural:** Descreve o circuito como interconexões de portas ou módulos primitivos (como um diagrama esquemático em código). Usa instâncias de módulos.
- **Comportamental:** Descreve o que o circuito faz, sem detalhes de portas. Usa ***assign***, ***always*** ou ***expressões funcionais***. Mais abstrato e fácil para lógica complexa.

Diferenças: Estrutural é baixo nível (próximo ao hardware), comportamental é alto nível (próximo à programação). Ambas podem ser sintetizadas.

Conceitos Básicos de Eletrônica Digital:

- **Níveis de Abstração em Verilog:**

Forma	Característica	Nível de abstração	Usado para
Estrutural	Conexão entre componentes (portas, módulos)	Baixo	Implementação física
Dataflow	Usa expressões lógicas (operadores `&`, ` | `, `^`)	Médio	
Comportamental	Usa blocos `always`, `if`, `case` e atribuições sequenciais	Alto	Modelagem funcional

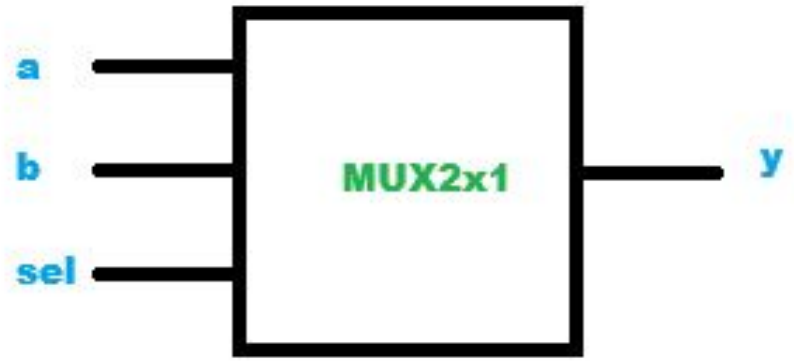
Exemplo de descrição em Verilog:

- **Multiplexador 2x1 (Mux2x1)**
- Conceito: Circuito que seleciona uma de duas entradas baseado em um sinal de seleção (sel). Útil para roteamento de dados.

- Forma Estrutural (mux2x1_struct.v): Usa portas primitivas.

Ex_estrutural.v

```verilog
module mux2x1_struct(a, b, sel, y);
    input a, b, sel;
    output y;
    wire not_sel, and_a, and_b;

    not u1 (not_sel, sel);      // Inversor
    and u2 (and_a, a, sel);     // AND para entrada a
    and u3 (and_b, b, not_sel); // AND para entrada b
    or u4 (y, and_a, and_b);    // OR para saída
endmodule
```



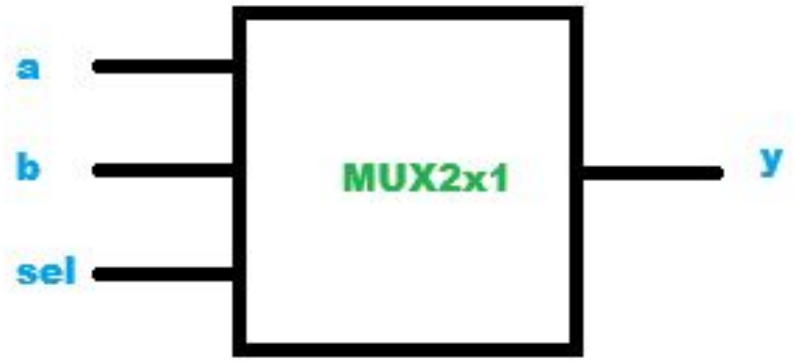
Exemplo de descrição em Verilog:

- **Multiplexador 2x1 (Mux2x1)**
- Conceito: Circuito que seleciona uma de duas entradas baseado em um sinal de seleção (sel). Útil para roteamento de dados.

- Forma Comportamental (mux2x1_behav.v): Usa operador condicional

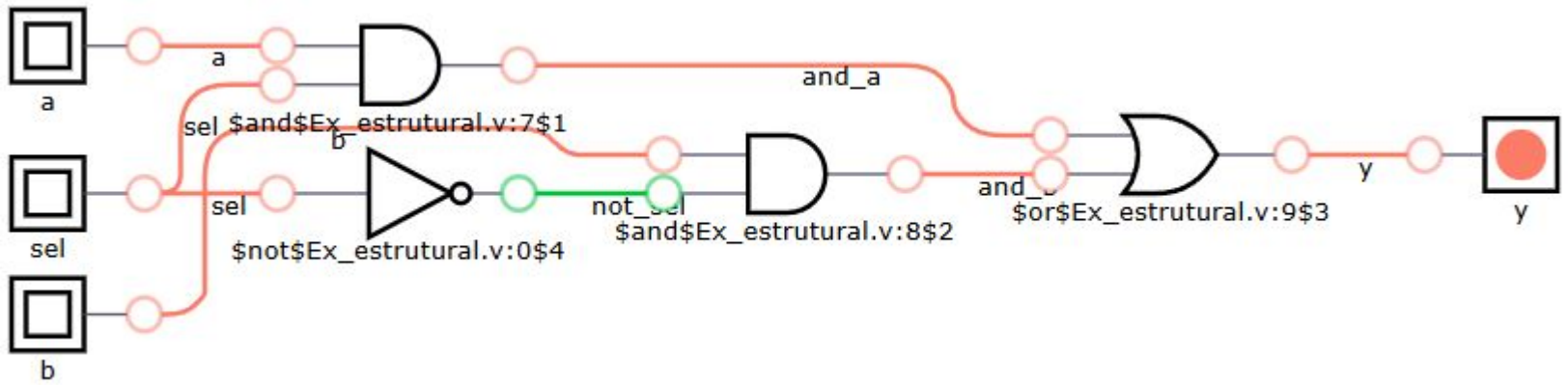
Ex_Comportamental.v

```
1  module mux2x1_behav(a, b, sel, y);
2      input a, b, sel;
3      output y;
4      assign y = (sel) ? a : b;
5  endmodule
6
```

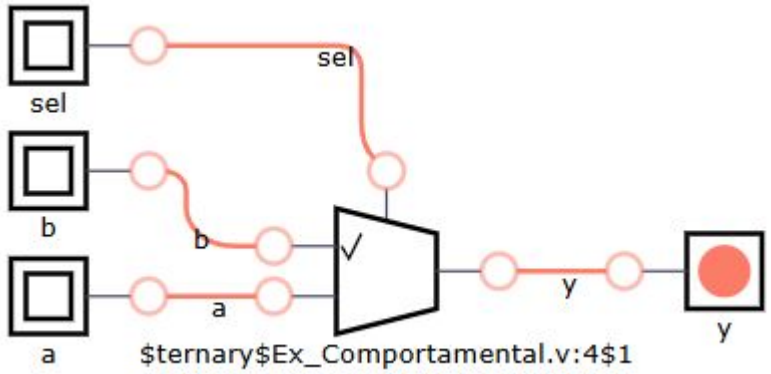


Exemplo de descrição em Verilog:

- **Multiplexador 2x1 (Mux2x1)**
  - **Forma Estrutural:**



  - **Forma Comportamental:**



Exemplo de descrição em Verilog:

- **Multiplexador 2x1 (Mux2x1)**
  - **Textbench:**

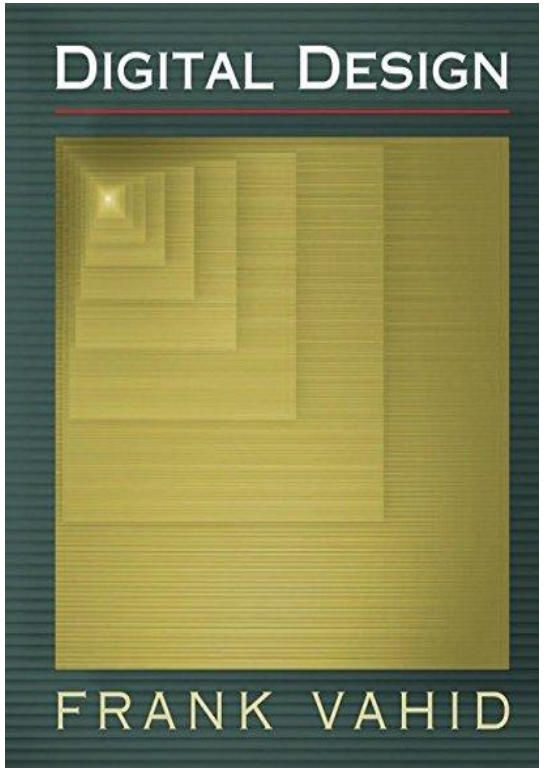
Ex_Testbench.v

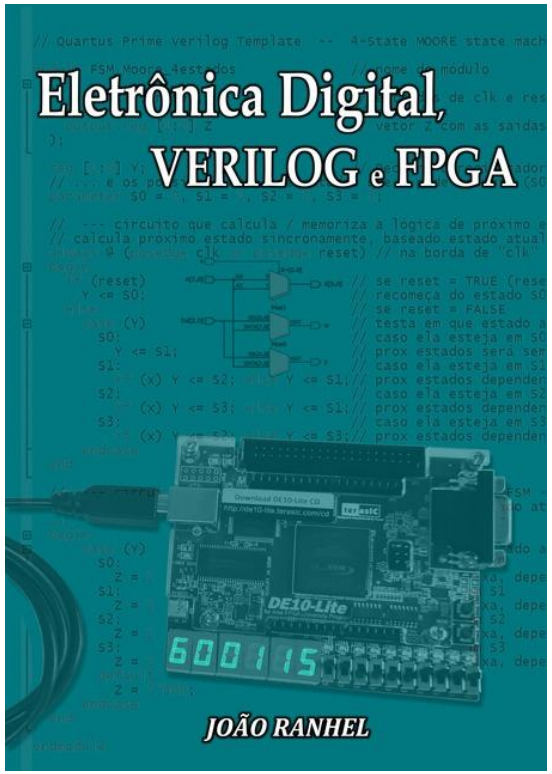
```
1   `include "Ex_estrutural.v"  // Ou Ex_Comportamental.v
2   `timescale 1ns/100ps
3   module mux2x1_tb;
4       reg a0, b0, sel0;
5       wire y0;
6       mux2x1_struct uut (.a(a0), .b(b0), .sel(sel0), .y(y0));  // Ajuste para behav se necessário
7       initial begin
8           $dumpfile("mux2x1.vcd");
9           $dumpvars(0, mux2x1_tb);
10          a0=0; b0=0; sel0=0; #10;
11          a0=0; b0=1; sel0=0; #10;
12          a0=1; b0=0; sel0=1; #10;
13          $finish;
14      end
15      initial $monitor("Tempo=%0t | a=%b b=%b sel=%b => y=%b", $time, a0, b0, sel0, y0);
16  endmodule
```

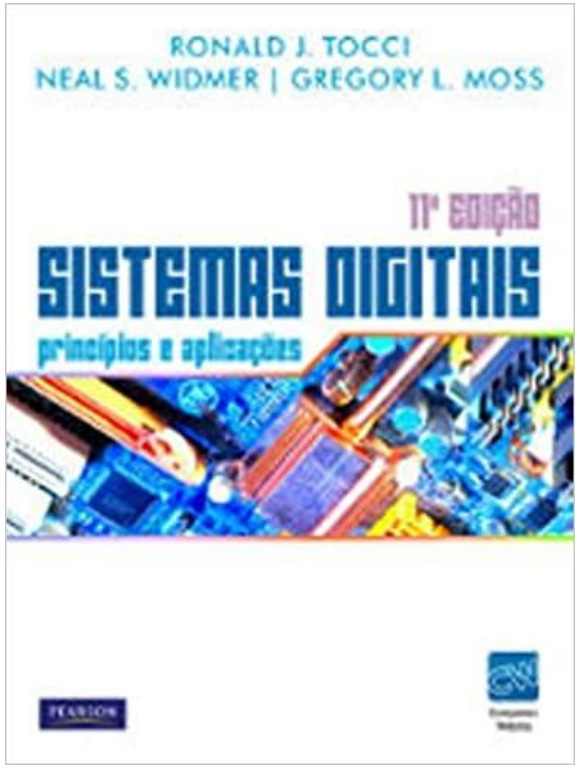
- Compile com iverilog -o mux2x1_tb.vvp mux2x1_tb.v, execute vvp mux2x1_tb.vvp e visualize em GTKWave.

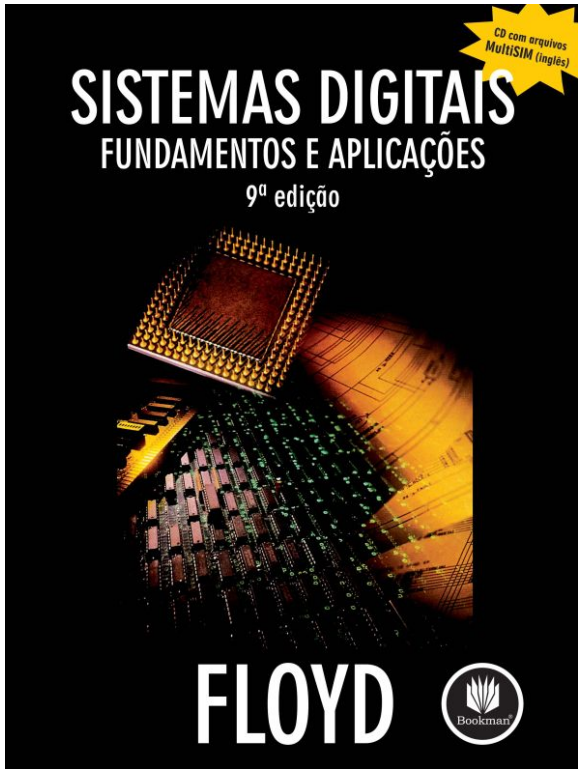
Bibliografia

Bibliografias Básicas:









Obrigado, e até a próxima aula!

Prof. Rogério D. Dantas
e-mail: rogerio.dantas@ifsp.edu.br