# Lab 0: Introdução à Plataforma de Desenvolvimento FPGA

**Universidade Estadual de Feira de Santana**
**Departamento de Tecnologia, Área de Eletrônica e Sistemas**
TEC 499 - MI - Sistemas Digitais

2016.1
rev 2.0

## Sumário



## Antes de Começar

Este laboratório é voltado tanto para introdução ao curso TEC 499 quanto para a condução das atividades administrativas. As atividades serão conduzidas e avaliadas no Laboratório de Hardware (LabHard). Como discente do curso, você tem acesso irrestrito aos recursos destinados aos estudantes (equipamentos, componentes, computadores, etc.). Atente para a página do curso, no sentido de se manter atualizado quanto ao horário de funcionamento do laboratório no período extra classe. Estas informações estão disponíveis na seção Infra-estrutura do site. Caso necessite de

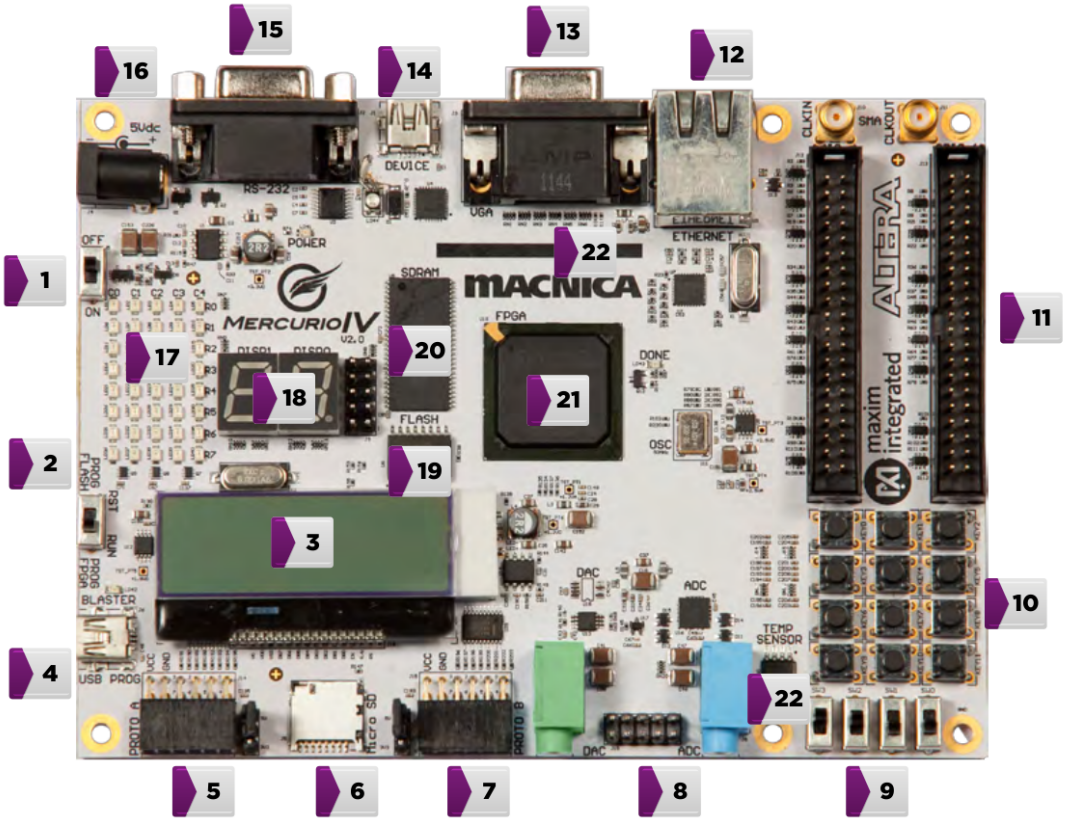
acesso em horários fora daqueles disponibilizados, verifique junto ao seu professor a possibilidade de acompanhamento durante um intevala específico.

Por medidas de segurança, os kits de desenvolvimento só serão fornecidos durante a sessão de laboratório. Como contamos com apenas dois kits, cada grupo deve utilizá-lo de forma a otimizar o tempo no sentido de não atrapalhar o trabalho das outras equipes. Para acesso aos computadores, utilize o login padrão utilizados pelos alunos do curso para o usuário `aluno`. Por fim, assumindo que isso não será um problema, os comentários ao longo dos códigos Verilog, assim como nomes de módulos, portas, sinais e instâncias estão escritos em inglês.

## 1. Plataforma de Desenvolvimento

TEC 499 usa o plataforma de desenvolvimento FPGA Mercurio IV. A imagem a seguir apresenta uma imagem da placa e a listagem a seguir identifica as suas partes mais importantes. O kit pode ser adquirido através do site da empresa MACNICA (http://www.macnicadhw.com.br). Descontos para estudantes são aplicáveis.

A Figura abaixo apresenta um perfil da placa, indicando os componentes presentes no kit. A seguir uma lista, elencando os seus componentes principais so o ponto de vista de TEC 499.




**1.** Chave Liga / Desliga

**3.** Display LCD

**4.** Programador USB Blaster

**9.** Switches

**10.** Teclado numérico

**11.** GPIO

**12.** VGA

**14.** Interface Serial

**17.** Matriz de LEDs

**18.** Displays de 7 segmentos

**19 e 20.** Memórias Flash e RAM

**21.** FPGA Cyclone IV

## 2. Introdução ao Verilog Estrutural

Ao longo deste semestre, você irá projetar sistemas de complexidade incremental utilizando Verilog, uma Linguagem de Descrição de Hardware (HDL) largamente utilizada na indústria e academia. Neste laboratório, você aprenderá Verilog estrutural, um subconjunto limitado de elementos da linguagem que possibilitam a você descrever circuitos em termos de sinais (*wires*), portas (*gates*) e módulos (*modules*) [1]. Para um estudo mais aprofundado (do qual fatalmente você não conseguirá fugir), sugerimos os tutoriais presentes em ASIC World.

### 2.1 Wires

*Wires* em Verilog estrutural são análogos a fios em um circuito construído manualmente: eles são usados para transmitir valores, na forma de sinais digitais, entre entradas e saídas ou outros *wires* internos.

*Wires* devem ser declarados antes de serem usados:

```
wire a;
wire b, c; // declara mais de um wire usando virgula
```

Os *wires* acima são escalares (i.e. representam 1 bit). Entretanto, eles também podem ser vetores:

```
wire [7:0]  d; // declara wire de 8 bits
wire [31:0] e; // declara wire de 32 bits
```

*Wires* podem ser atribuídos a outros *wires*, concatenados e indexados:

```
wire [31:0] f;
assign f = {d, e[23:0]}; // concatena d com os 24 bits menos
                         // significativos de e
```

Na linha acima, os colchetes `[]` são usados para indexar os 24 bits de e as chaves `{}` concatenaM *wires* separados por vírgula. Convenções de notação determinam que os bits mais significativos são especificados primeiro, seguido de uma vírgula e então os bits menos significativos.

## 3. Gates (Primitivas Estruturais)

Neste laboratório, você pode utilizar as seguintes primitivas: `and`, `or`, `xor`, `not`, `nand`, `nor`, `xnor`. Em geral, a sintaxe é:

```
<operador> (saida, entrada1, entrada2);  // para portas de duas entradas
<operador> (saida, entrada);             // para porta not
```

Por exemplo, o código Verilog a seguir implementa a equação Booleana `F = a + b`:

```
wire a, b, F;
/* ... algum codigo que atribui valores para a e b */
or (F, a, b);
```

[1] Nota: Este laboratório assume que você possui familiaridade com portas lógicas e álgebra Booleana.

Funções lógicas complexas podem ser implementadas usando *wires* intermediários entre estas portas primitivas. Este processo pode facilmente se tornar tedioso e extremamente complexo; nos próximo laboratório iremos explorar formas menos maçantes de implementação envolvendo sistemas complexos.

## 4. Módulos (module)

Módulos (`module`) proporcionam um meio de abstração e encapsulamento para seu projeto. Eles consistem de uma declaração de portas e código Verilog para implementar uma funcionalidade desejada. Por exemplo, considere um módulo que calcula `y = (a + b)(c + d)`:

```
module example_module (
  // declaracao de portas e wires
  input a, b, c, d;
  output y;
);
  wire a_or_b, c_or_d;
  // logica
  or  (a_or_b, a, b);
  or  (c_or_d, c, d);
  and (y, a_or_b, c_or_d);
endmodule
```

Neste exemplo podemos verificar alguns pontos importantes:

- As portas são consideradas internamente de forma equivalente a *wires* de entrada ou saída, mas podem ser tratadas como canais de ligação dentro de um módulo.
- *Wires* declarados dentro de um módulo (como `a_or_b`) possuem o escopo limitado àquele módulo.
- Módulos podem ser criados em um arquivo Verilog (`.v`) e recomenda-se que o nome do arquivo coincida com o nome do módulo (então, o exemplo acima pode ser encontrado em `example_module.v`).

Uma vez criado o módulo, você pode instanciá-lo em outros módulos:

```
wire a_in, b_in, c_in, d_in, result;

example_module unique_name(
  .a(a_in),   // conecta o wire em uma porta
  .b(b_in),
  .c(c_in),
  .d(d_in),
  .y(result)
);
```

Neste exemplo, note que as portas de entrada `a_in`, `b_in`, `c_in`, e saída `result` são *wires* válidos no módulos que esta instanciação ocorre. Além disso, assumimos que `unique_name` é um identificador global único.

A sintaxe `.<input/output port>(<wire>)` é usada para conectar explicitamente *wires* às entradas e saídas corretas de um módulo.

Você pode também escrever:

```
wire a_in, b_in, c_in, d_in, result;
// ordem correta
example_module unique_name(a_in, b_in, c_in, d_in, result);
```

A declaração acima é também perfeitamente válida, mas não é recomendada, uma vez que é possível misturar a ordem dos *wires*. A primeira forma também é de mais fácil compreensão e leitura. A declaração a seguir é um exemplo de declaração incorreta, uma vez que a ordem correta dos parâmetros foi violada.

```
wire a_in, b_in, c_in, d_in, result;
// ordem incorreta
example_module unique_name(result, a_in, b_in, c_in, d_in);
```

# 5. Procedimento de Laboratório

Siga as etapas a seguir e certifique-se de salvar seu trabalho como forma de verificação do seu progresso. Se algo der errado durante os procedimentos (e.g. seu projeto não funciona na placa, erros de compilação), solicite ao professor que olhe o seu código. Todavia, note que você será o responsável por qualquer falha identificada, ou não pelo professor.

## 5.1 Adquirindo os arquivo de laboratório

Os arquivos de laboratório estão disponíveis no quadro do Trello correspondente à sua turma junto à esta descrição.

Localize o arquivo de laboratório `lab0.zip` e salve-o no seu computador.

Uma vez que você esteja de posse do arquivo `zip`, execute o seguinte comando:

```
unzip lab0.zip
```

Este comando deve descompactar os arquivo no diretório `lab0`

## 5.2 Funcionalidades Existentes

Dentro do diretório `verilog`, abra o arquivo `ml505top.v` no seu editor de textos favorito. O arquivo `ml505top.v` é o módulo "top level", no qual as portas de entrada e saída representam a conexão física com a placa. Todos os submódulos neste laboratório serão instanciados como parte deste módulo top level. Em outras palavras, o módulo `ml505top` é o módulo de maior ordem na hierarquia do circuito.

A definição de portas para este módulo possui uma entrada de 4 bits para as chaves DIP *switch* (#9 no diagrama) duas saídas de 5 e 8 bits para a matriz de LEDs (#17 no diagrama). Além disso, o circuito apresenta uma entrada de 12 bits para os botões (#10 no diagrama). Por fim, 3 bits são usados para representar os sinais `RED`, `GREEN` e `BLUE` do LED RGB (não destacado no diagrama). A entrada de clock não será discutida neste laboratório, pois é usada apenas para validação interna e você não deve se preocupar com ela.

A estrutura de diretórios e arquivos fornecida contém um simples exemplo de porta AND. Esta porta recebe como entradas os DIP *switches* `SW0` e `SW1`, e representa a operação AND entre estas duas no LED (`R0`,`C0`).

Antes de você implementar o resto do laboratório, verifique estas funcionalidades em hardware:

1. Abra uma janela do terminal e entre no diretório `lab0`.
2. Execute o comando '`make -C fpga`'. Este comando executará um conjunto de processos de transformação do seu código, produzindo, no final, um *bitstream* de configuração para a placa. Você irá entender estas etapas ao longo dos próximos laboratórios.
3. Conecte o cabo USB do kit na entrada USB Blaster (#4 no diagrama) e em uma das portas USB do computador. Note que o mesmo canal de alimentação será usado para a programação do dispositivo.
4. Verifique que o LED power, localizado próximo ao conector de saída serial RS232 da placa, está aceso.
5. Após o `make` ser concluído, execute o comando '`make -C fpga program`' a partir do mesmo diretório. Isso fará com que o FPGA seja programado com o arquivo de *bitstream* criado no passo 1.
6. O projeto deve então estar na placa. Verifique que o LED da matriz (`R0`,`C0`) representa a AND das chaves `SW0` e `SW1`. Note que, neste momento, os demais LED de `R0` permanecerão apagados e o LED RGB acenderá em vermelho.

Não se preocupe com os alertas (*warning*) que surgirem na tela por enquanto. Todavia, é recomendado que você comece a se identificar com essas mensagens, pois elas te seguirão até o final do curso. Apesar de boa parte delas apenas indicarem algo que você provavelmente já sabe, muitas vezes elas podem apontar, indiretamente, o motivo de uma falha funcional do seu circuito.

## 5.3 Multiplexador 2-1

Em seguida ao exemplo da porta AND, você verá ainda uma instância de `Mux2_1` em `ml505top.v`. Implemente este módulo em `Mux2_1.v` (de modo que ele funcione conforme os comentários no arquivo) e siga os passos da Seção 5.2 para testar sua implementação no hardware.

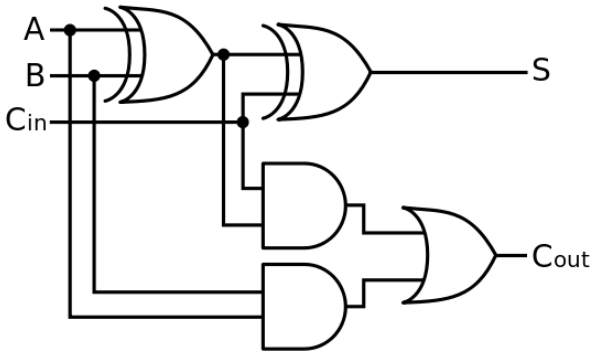
As duas entrada `A` e `B` correspondem, novamente, às chaves `SW0` e `SW1` da placa. A entrada de seleção do multiplexador corresponde à chave `SW2` e a saída aparece no LED (`R0`,`C1`) da matriz. Você pode encontrar estas conexões às portas no arquivo `ml505top.v` onde o módulo `Mux2_1` é instanciado. Note que usamos as mesmas chaves aplicadas às entradas da porta AND. Caso ache necessário, comente a porção de código correspondente à operação AND.

## 5.4 Somador Completo

Em seguida, examine as definições de porta presentes no arquivo `FA.v`. Seguindo as instruções nos comentários de `ml505top.v` e os exemplos apresentados neste roteiro, escreva a instanciação de `FA` no módulo `ml505top`. Você pode achar conveniente olhar a instanciação de `Mux2_1` como um exemplo.

Finalmente, usando apenas Verilog estrutural, implemente um circuito somador completo em `FA.v`. Se você não está familiarizado com a funcionalidade de um somador completo, a tabela verdade e uma possível implementação em nível de porta lógica é apresentada a seguir.

Novamente, você deve ser capaz de seguir os passos descritos na Seção 5.2 para testar o somador no hardware. Até aqui, o arquivo `ml505top.v` não possui a instância para o módulo `FA.v`, que deve ser feita por você. Este processo deve ser muito semelhante ao do módulo `Mux2_1`. Certifique-se de ler os comentários no código para especificações de porta. Lembre-se que você pode usar qualquer umas das chaves de entrada, representadas por `SW0..3`.




| **A** | **B** | $C_{in}$ | **S** | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## 5.5 Expandindo o Projeto

Agora que finalizamos a implementação do somador completo, nós iremos tentar algo um pouco menos entediante e criar um somador *ripple*. No diretório de arquivos fonte deste laboratório, você encontrará um arquivo chamado `Adder.v`, onde você deve instanciar os somadores completos que você descreveu durante o procedimento anterior. Desconsidere o arquivo `BehavioralAdder.v` por enquanto; nós iremos apenas lidar com Verilog estrutural neste laboratório.

Nós queremos que você construa um somador *ripple* de 8 bits, o que quer dizer que seu circuito conterá 8 instâncias do somador completo que você já implementou. Ao invés de descrever 8 instâncias do somador completo, Verilog possui a estrutura `generate` que torna a nossa vida mais fácil, uma vez que permite a você (além de outras coisas) gerar múltiplas cópias de um mesmo módulo.

Para usar o comando `generate` nós também precisamos de um parâmetro o qual é definido por você no arquivo `Adder.v`. O comando `generate` e os parâmetros são declarados como segue:

```
module foo(
  input  [N-1:0] x;
  output [N-1:0] y;
```

```
);
  // declaracao do parametro N
  parameter N = <default_value>;
  // o parametro pode ser usado para especificar
  // tamanho de sinais
  wire [N-1:0] z;
  // variavel a ser usada no comando generate
  genvar i;
  // declaracao de um comando generate
  generate for( i = 0; i < N; i = i + 1)
    begin:<unique_name>
      /* Aqui, o modulo baz tem um parametro chamado width */
      baz
        #( .width(N) )
        baz_in (
          .a(x[i]),
          .b(y[i]),
          .c(z[i])  );
    end
  endgenerate
endmodule
```

Sua tarefa é usar o comando `generate` para instanciar um conjunto de somadores e completar o módulo `Adder`. Se você ficou confuso no que diz respeito ao comando `generate`, pergunte ao seu professor ou consulte os tutoriais disponíveis no site do curso.

Quando você completar o módulo `Adder`, execute a sequência de comandos:

```
make -C fpga clean
make -C fpga
make -C fpga program
```

Quando você terminar de programar a placa, se você descreveu corretamente seu arquivo `Adder.v`, o LED RGB deve acender na cor verde. Do contrário, o LED RGB permanecerá vermelho e você terá que corrigir o seu somador.

## 6. Acompanhamento

Para questões de acompanhamento, você deverá apresentar sua implementação em hardware, assim como as respostas para as questões de acompanhamento.

Por favor, mantenha as respostas para as questões de acompanhamento abertas em um editor de texto ou escritas em papel (elas não são coletadas, então não há necessidade de imprimi-las se você digitar).

Requisitos de Acompanhamento:

1. Mostre os três arquivos Verilog que você modificou.
2. Quantas portas lógicas o seu somador precisa para ser implementado?
3. Mostre o multiplexador funcionando na placa.

4. Mostre a implementação do arquivo `Adder.v` e execute '`make -C program`' na placa.