Lab 2: Síntese em Verilog e FSM

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

2016.1

Sumário

Introdução

Neste laboratório iniciaremos as primeiras implementações utilizando Verilog comportamental. Além disso, iniciaremos o contato com uma das mais importantes tarefas das ferramentas de EDA: Síntese Lógica. Completar este laboratório dará a você experiência para descrever seu circuito de forma rápida, e sob a perspectiva de um nível mais alto de abstração. Além disso, você irá analisar a lógica produzida pela ferramenta de EDA, e possíveis ineficiências apresentadas após a tradução da descrição em Verilog para implementação em FPGA.

Nós preparamos um *framework* para o projeto de uma trava construída a partir da combinação de dois dígitos. Seu trabalho será implementar uma máquina de estados (FSM) para completar o sistema. As entradas da sua trava consistem de quatro chaves DIP e dois *push buttons*. As chaves, representadas pelo sinal `Digit[3:0]`, são usadas para selecionar uma combinação de dígitos de entrada. O *push button* (KEY11), representado pela entrada `Enter` é usado para confirmar o dígito selecionado. O LED RGB é utilizado para indicar que a trava foi aberta, e também que uma combinação incorreta foi introduzida.

1. Pré-laboratório

Nota: Por favor, certifique-se de completar os itens nesta seção antes de ir ao laboratório. Você provavelmente não terminará o laboratório durante a sessão se você não completar o pré-lab.

No pré-lab, complete as seguintes tarefas:

1. Leia atentamente este roteiro. Em particular, a Seção 2.: **Detalhes do Projeto** lhe dará uma série de informações valiosas para o circuito que você irá construir. Por favor, também considere as recomendações presentes no documento **The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates**, disponível no portal do curso. Este material será útil para entender as metodologias de projeto eficiente da FSM que você usará neste laboratório e nos próximos.
2. Faça o download dos arquivos do laboratório. Como sempre os arquivos são distribuídos no seu quadro no Trello.
3. Escreva o seu código Verilog antes. Você é responsável por implementar os módulos `LevelToPulse` e `Lab2Lock`.

2. Detalhes do Projeto

2.1 Visão Geral

A Figura a seguir apresenta um diagrama do circuito que desejamos implementar.

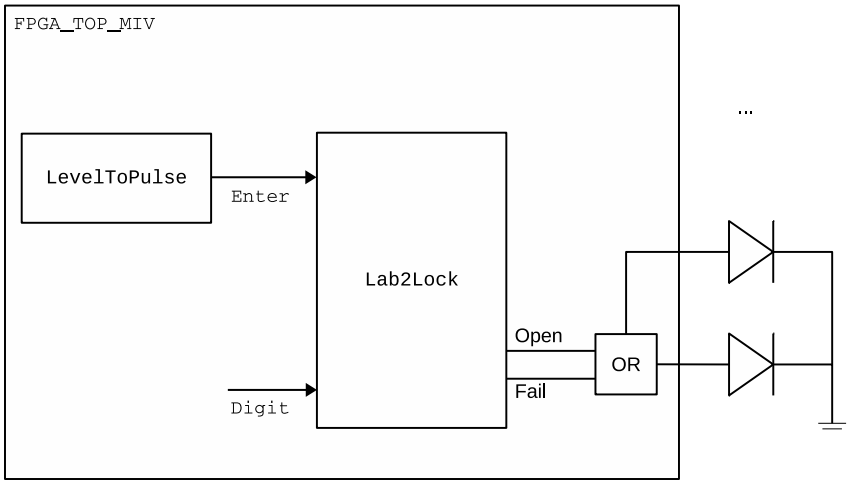


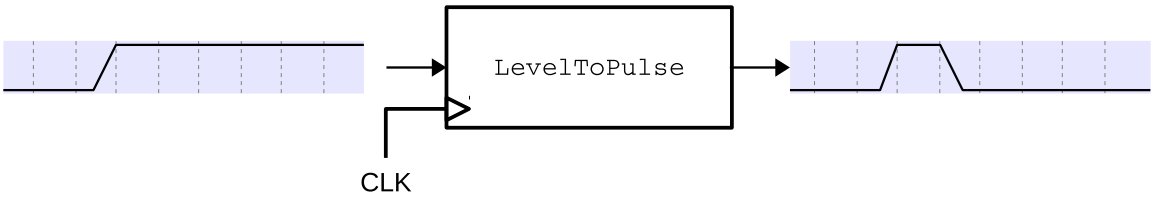
Figura 1: Diagrama de blocos do módulo FPGA_TOP_MIV.

2.2 LevelToPulse

Entradas oriundas de *push buttons* são geralmente associadas à presença de ruídos constantes, com oscilações em pequenos espaços de tempo antes de estabilizar em um valor. Por sorte a plataforma Mercurio IV implementa um filtro capaz de eliminar a presença deste ruído, gerando um sinal

"limpo" e de nível constante nas entradas dos pinos associados a cada um dos botões. Para mais detalhes, consulte o Manual de Usuário da placa Mercurio IV.

Este módulo captura um contato no *push button* e limita a entrada a um pulso que ocupa apenas 1 ciclo de clock, a partir do momento em que o usuário pressiona o botão (borda ascendente). O circuito deve funcionar como um detector síncrono de borda ascendente A figura a seguir descreve o funcionamento conceitual do `LevelToPulse`.



Uma das formas de implementar este módulo é a partir do projeto de uma FSM. Para isso, você deve atender aos seguintes requisitos:

1. Levando em conta uma FSM de Mealy ou Moore qual delas apresenta o menor número de estados?
2. Escolha um tipo de FSM e desenhe o diagrama de transição de estados para o módulo `LevelToPulse`.

A tabela a seguir apresenta a especificação de portas de E/S para o módulo `LevelToPulse`.

Sinal	**Tamanho**	**Direção**	**Descrição**
`Clock`	1	In	O clock
`Reset`	1	In	Reset do da FSM do `LevelToPulse`
`Level`	1	In	Entrada direta do *push button*
`Pulse`	1	Out	Saída de um único pulso

2.3 Lab2Lock

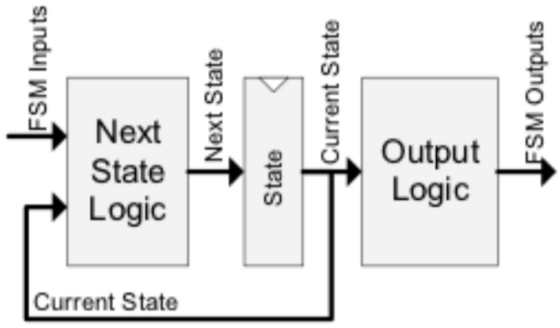
Este módulo é responsável por manter o estado da trava e gerar as saídas que indicam o seu status através do LED RGB. Sendo assim, ele deve detectar que `DIGIT1` e `DIGIT2` foram introduzidos na ordem correta. Você deve usar o comando em Verilog `localparam` para definir os valores destas constantes dentro do seu módulo. O seguinte trecho de código descreve como isso é feito:

```
/* Inside the Lab2Lock module */
localparam DIGIT1 = 4'h2;
localparam DIGIT2 = 4'h3;
```

O `localparam` define um parâmetro constante sob o escopo do módulo onde este foi declarado. Isso quer dizer que, diferente do comando `parameter`, o `localparam` só pode ser modificado dentro do módulo `Lab2Lock`. Utilize os valores acima para determinar a combinação da sua trava no sentido de facilitar os testes. Você pode modificar estes valores depois se quiser testar com outros valores. Entretanto, para o acompanhamento, utilize os valores fornecidos neste documento.

Você irá descrever este módulo na forma de uma FSM. Especificamente, você deve implementar uma máquina de **Moore** para completar esta tarefa. Uma máquina de Moore depende apenas

do estado atual para gerar suas saídas. Dessa forma, seu módulo `Lab2Lock` terá a seguinte organização em alto nível:



Adicionalmente, você pode optar por acrescentar registradores às saídas da sua FSM. Lembre-se das recomendações apresentadas no documento elencado acima no sentido de construir FSMs eficientes em Verilog.

A especificação de portas e o diagrama de transição de estados para este módulo é apresentado a seguir.

Sinal	**Tamanho**	**Direção**	**Descrição**
`Clock`	1	In	O clock
`Reset`	1	In	Reset da trava (retorna para o estado inicial)
`Enter`	1	In	Informa para a trava ler um novo dígito
`Digit`	4	In	Um dígito da combinação
`State`	3	Out	O estado da FSM, para propósitos de depuração
`Open`	1	Out	Indica que o sistema de trava está aberto
`Fail`	1	Out	Indica que o usuário introduziu uma combinação errada

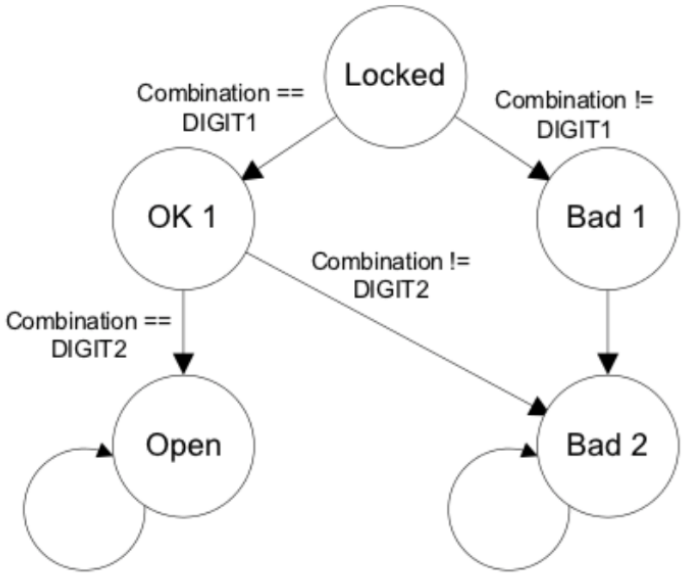


Figura 2: FSM do módulo Lab2Lock.

Quando estiver pronto para testar sua implementação, compile seu projeto e programe-o na placa. Primeiro tente introduzir a combinação correta e verifique se funcionou. Neste caso, uma implementação correta deve emitir a cor verde no LED RGB. Caso contrário LED RGB deve emitir a cor vermelha, indicando que houve uma falha. É importante salientar que este LED não deve acender de forma prematura, ao perceber apenas um dígito incorreto. Para critérios de depuração, o código binário correspondente ao estado da sua trava deve ser exibido nos LEDs `R0-R2` da primeira coluna da matriz de LEDs.

O sinal de reset da trava pode ser ativado pressionando o primeiro *push button* (KEY0). Este sinal deve ser utilizado para retornar a FSM para o seu estado inicial.

3. Dicas e Erros Comuns

Uma vez que consideramos ser esta o seu primeiro contato com lógicas sequencial e combinacional em Verilog, os módulos que você terá de implementar neste laboratório são simples e diretos. A seguir apresentamos algumas dicas e erros comuns que os estudantes apresentam durante suas primeiras investidas na descrição de circuitos em Verilog.

Combinacional *vs.* Sequencial

É sempre bom olhar para a interface que você precisa implementar antes de começar a descrever seu código. Nos módulos `LevelToPulse` e `Lab2Lock`, primeiro identifique quais sinais são relevantes para a região combinacional do módulo e quais sinais são necessários para determinar os valores para o próximo estado em seus registradores. Isso o ajudará a acompanhar quando, onde e quais os sinais devem ser utilizados.

Múltiplos Blocos `always @`

Neste laboratório você terá que usar múltiplos blocos `always @ (<port list>)`. Um destes blocos para lógica sequencial e outro será usado para atribuição dos valores correspondentes aos sinais de lógica combinacional. Um erro comum é atribuir um sinal (ex: `bad_signal`) em ambos os blocos `always @ (posedge clock)` e `always @ (*)`.

Por exemplo:

```
reg bad_signal;
always @ (posedge clock) begin
        bad_signal <= bad_inpt1;
end
always @ (*) begin
        bad_signal <= bad_input2;
end
```

Isso resultará em um sinal duplamente atribuído e as ferramentas de síntese para dispositivos FPGA não permitem que esse tipo de atribuição seja realizada. Sempre tente manter todas as atribuições para um sinal em um único bloco `always` para evitar este problema.

Atribuições Bloqueantes *vs.* Não-bloqueantes

Verilog possui dois tipos de atribuição denominadas bloqueante (=) e não-bloqueante (<=). Uma prática comum dos iniciantes em projetos de circuitos em Verilog é utilizar ambos os tipos de atribuição em um mesmo bloco `always`. Neste caso, a ferramenta de síntese adotará um critério arbitrário acerca do comportamento do seu circuito. Em geral, o uso de ambas as atribuições resulta no mal funcionamento do circuito final.

A explicação para esta característica é oriunda da própria definição da linguagem. Na atribuição bloqueante, a avaliação e atribuição do comando é realizada de forma imediata. Analise o exemplo a seguir.

```
always @(a or b or c)
begin
   x = a | b;      // avalia a | b, atribui resultado em x
   y = a ^ b ^ c; // avalia a ^ b ^ c, atribui resultado em y
   z = b & ~c;     // avalia b & (~c), atribui resultado em z
end
```

Por outro lado, atribuições não bloqueantes avaliam o lado direito de todas as expressões presentes em um bloco `always` para então realiza-las. Compare agora a análise anterior com o código a seguir.

```
always @(a or b or c)
begin
  x <= a | b;      // avalia a & b, defere atribuicao em x
  y <= a ^ b ^ c; // avalia a ^ b ^ c, defere atribuicao em y
  z <= b & ~c;     // avalia b & (~c), defere atribuicao em z
end                // atribui x, y e z com seus novos valores
```

Em geral, recomendamos a atribuição bloqueante para lógica combinacional e não bloqueante para lógica sequencial.

Representação dos Sinais de E/S

Outro obstáculo que muitos estudantes enfrentam é no que diz respeito à representação lógica dos sinais de E/S da plataforma de desenvolvimento. Lembre-se de que o botão é ativo em borda ascendente, ou seja, quando pressionado ele produz um sinal de nível lógico alto na entrada do circuito. Por outro lado, a matriz de LEDs e os sinais Red, Green e Blue do LED RGB são ativos em borda descendente, ou seja, para acendê-los você deve enviar um sinal de nível lógico baixo. Para maiores informações, consultar o Manual do Usuário da Placa Mercurio IV.

4. Acompanhamento

Neste momento, você já deve ter a sua trava em pleno funcionamento. Para compilar o seu circuito você não precisa especificar a implementação individual de cada unidade funcional. Em vez disso, você forneceu para a ferramenta de síntese uma representação textual de alto nível que descreve o comportamento que você deseja que o circuito apresente. Com isso, a ferramenta de síntese fez o melhor para atender à esta descrição. Você agora tem experiência suficiente para notar o quanto uma síntese de alto nível pode afetar o tempo de desenvolvimento.

Em uma folha de papel apresente:

1. As informações solicitados na Seção 2.2.
2. Uma análise da implementação RTL do seu circuito. Como a ferramenta interpretou a implementação das duas FSMs que você descreveu?
3. Colete as informações a seguir.
    - Número de LEs ocupados
    - Quantidade de LABs ocupados completamente ou parcialmente
    - Número de registradores (utilizados como *flip-flops*)
    - *Lembre-se que você pode fazer isso olhando os relatórios gerados ao final da compilação.*

Por fim, demonstre o funcionamento da sua trava através da introdução de uma combinação correta e outra incorreta. Demonstre ainda o funcionamento do reset da trava (retorno para um estado inicial).