Insper

Elementos de Sistemas - Projeto E - Ferramental

Rafael Corsi - rafael.corsi@insper.edu.br

Março, 2018

Início do projeto

1. Atualizar arquivo ScramMaster.txt
2. Atualizar repositório com o upstram
3. Adicionar o novo script de teste ao travis : E-LogicaSequencial/script/testeLogicaSequencial.py
4. Criar projeto no github
5. Atribuir tarefas e acompanhar o desenvolvimento

Process VHDL

Em VHDL quando desejamos fazer algo sequêncial é necessário usarmos uma estrutura chamada de process, que possui a declaração a seguir :

```
process (optional sensitivity list)
    declarations
begin
    sequential statements
end process;
```

Nesse **process** possuímos a lista de sensibilidade (*sensitivity list*) que indica quando o process será executado. Podemos pensar da seguinte maneira, sempre que algum sinal que está listado nessa lista de sensibilidade mudar de valor (0 -> 1, 1 -> 0) o processo será executado, vemos o exemplo a seguir :

```
process(A)
begin
  Q <= A;
end process;
```

> Sempre que o sinal A (sinal ou porta) alterar de valor o sinal Q será atribuído com o seu valor

Agora vamos criar um outro processo, esse estará errado :

```
process(A)
begin
  Q <= A and B;
end process;
```

A ideia por traz desse processo seria que o sinal Q receba o sinal A e B sempre que algum dos dois sofram alguma alteração, porém essa implementação não irá funcionar já que B não faz parte da lista de sensibilidade e se B mudar de valor o processo não será chamado, o sinal Q só será atualizado quando A mudar de valor.

Reescreva o módulo anterior para corrigir esse problema:

Clock

Para inserirmos um clock (um sistema síncrono) precisamos necessariamente usar um process, e a arquitetura é a seguinte :

```
process(clock)
begin
  if(rising_edge(clock)) then
     Q <= D;
  end if;
end process;
```

> Sempre que o clock sofrer variação (0 -> 1, 1 -> 0) o process é chamado e verifica-se se a transição foi de borda de subida (rising_edge) se for, atribui o sinal A ao sinal Q, caso contrário Q mantém seu último valor.

Não se deve usar no mesmo registrador em ambas as bordas : de subida (rising_edge) e de descida (falling_edge) pois **não será suportado por hardware**, salvo em registradores DDR (duble data rate) especiais, exemplo :

```
 process(clock)
begin
  if(rising_edge(clock)) then
     Q <= D;
  elsif(falling_edge(clock)) then
     Q <= D;
  end if;
end process;
```

Modifique o arquivo **src/rtl/FlipFlopD.vhd** que declara um Flip Flop do tipo D com o exemplo de código anterior.

RTL Viewer

O RTL Viewer é uma ferramenta do Quartus utilizada para visualizar a interpretação do compilador do hardware descrito em VHDL (ou outra linguagem de descrição de hardware HDL, tal como Verilog). Muito útil para análise de projeto, deve ser bastante utilizada, além de servir para documentação.

Para criarmos esse RTL Viewer de um módulo específico devemos no Quartus selecionar o componente em questão como TopLevel, para isso faça o seguinte :

1. Abra o projeto do quartus localizado em E-LogicaSequencial/Quartus/
2. No *Project Navigator* escolha por *Files*:

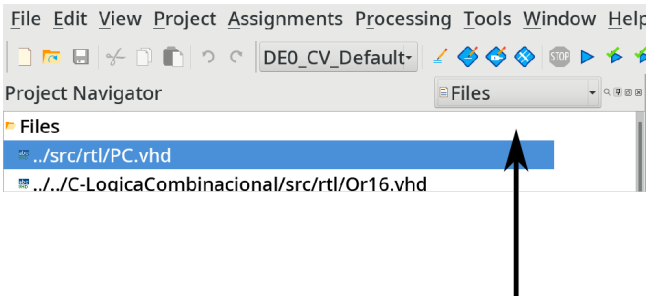

Figure 1: Files

3. Encontre o arquivo que deseja configurar como TopLevel :

Figure 2: FlipFlopD

Clique com o botão direito no arquivo e selecione a opção : **Set as Top Level Entity**

Agora o quartus irá considerar esse módulo como sendo o "top" do projeto (podemos pensar como sendo o main), compile o projeto.

4. Compile o projeto
5. Abra o RTL Viewer (Tools -> NetList Viewers -> RTL Viewer)

Essa declaração em VHDL irá ser interpretada pelo compilador como um FlipFlop tipo D.

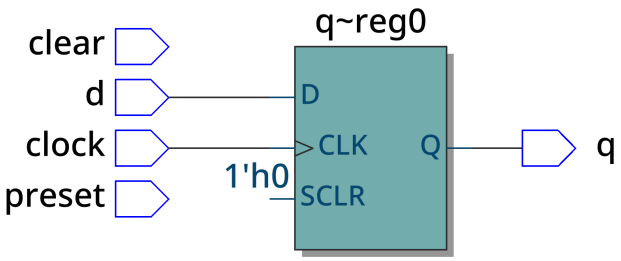

Figure 3: FF RTL

Reset e Preset

Flip Flops possuem normalmente duas outros sinais : Clear e Preset, usado respectivamente para zerar a saída do FF ou colocar um na sua saída. Vamos modificar o código anterior para suportar essas duas outras funcionalidades. Nesse caso possuímos duas opções :

- Set/Clear : síncrono
- Set/Clear : assíncrono

O modo síncrono seria que o set e o clear só pode ser executado na subida do clock e no assíncrono em qualquer momento que o sinal se set e clear chegar o FF irá responder.

Nesse caso, iremos implementar o FF com set e reset assíncrono, para isso utilize a seguinte estrutura que já define o clear e implemente para o preset:

```
process(clock, clear)
```

```
begin
  if (clear = '1') then
     Q <= '0';
  elsif(rising_edge(clock)) then
     Q <= D;
  end if;
end process;
```

Agora você pode executar o script de teste do projeto e verificar se a implementação está correta, analise também a forma de onda via modelsim.

Gere o RTL e analise o resultado.

Testando na FPGA

No Quartus atribua ao toplevel o arquivo TopLevel.vhd (mesmo passos anteriores), esse módulo irá mapear o FF recém criado para os pinos da FPGA:

```
Clock <= not KEY(0); -- os botoes quando nao apertado vale 1
                     -- e apertado 0, essa logica inverte isso
clear <= not KEY(1);
set   <= not KEY(2);

u0 : FlipFlopD port map (
        clock    => Clock,
        d        => SW(0),
        clear    => clear,
        preset   => set,
        q        => LEDR(0)
    );
```

! Esse exemplo é **proibido** de ser feito em projetos com FPGAs, não se deve gerar um clock a partir de um pino qualquer da FPGA, a FPGA possui pinos específicos para a geração do clock. Porém é a melhor maneira didática de mostrar um FF operando.
O correto seria colocarmos um pino de "enable" que ativaria ou não o clock do FF .