Trabalho Prático 3 – TP3 (EM DUPLAS)

O módulo a ser implementado é o ***traffic_filter***. O objetivo deste módulo é detectar determinados padrões em um fluxo de entrada de dados serial. Estes padrões correspondem a "ataques", devendo-se **notificar** o mesmo. O ***traffic_filter*** bloqueia a saída de dados em caso de detecção de igualdade entre o fluxo de entrada e um determinado padrão.

1 Orientações Gerais

Entrega do TP3	07/JUNHO/23 (qua) – 09h45

Este trabalho consiste nas seguintes tarefas:

- **[2,0 pontos]** Projetar a máquina de estados (FSM) que controla o circuito proposto, *traffic_filter*, desenhando-a e explicando o funcionamento da mesma. Este projeto é a **parte 1 do relatório** a ser entregue. Significa apresentar de forma textual a descrição da FSM, com o seu desenho, e explicação do funcionamento do circuito.
- **[1,0 pontos]** Modelar corretamente o módulo *compara_dado*. Este módulo é um componente do *traffic_filter*.
- **[4,0 pontos]** Desenvolver a modelagem do circuito proposto em linguagem VHDL. No material de apoio ao trabalho há o arquivo *tp3.vhd*, que deve receber o código desenvolvido. Os pontos serão divididos da seguinte maneira:
  - [0.5 ponto] correta modelagem da entidade
  - [0.5 ponto] correta modelagem dos sinais internos à arquitetura
  - [0.5 ponto] instanciação correta dos 4 módulos *compara_dado*
  - [1.0 ponto] correta modelagem da máquina de estados finita (FSM)
  - [1.0 ponto] correta modelagem dos registradores dependentes da FSM, e registrador de deslocamento
  - [0.5 ponto] correta modelagem dos circuitos combinacionais e atribuição das saídas
- **[3,0 pontos]** Apresentar no relatório **um cenário** de operação contendo:
  - [1.0 ponto] programação de 4 padrões diferentes
  - [1.0 ponto] detecção de ao menos 3 padrões
  - [1.0 ponto] reprogramação de novos padrões (pode ser apenas 1) – evento não mostrado na figura exemplo – e detecção do novo padrão programado

  Apresentar e explicar os vetores de teste e as formas de onda. A apresentação do cenário de teste é a **parte 2 do relatório** a ser entregue. Usar como modelo para o relatório a figura apresentada no final deste documento.

▶ **Entregar:** um arquivo em formato ZIP contendo **6 (seis)** arquivos:
1. modelagem em VHDL módulo compara_*dado* (*comp.vhd*)
2. modelagem em VHDL do circuito proposto (*tp3.vhd*)
3. *tb.vhd* (alterado em relação ao arquivo fornecido para contemplar a parte 2 do relatório)
4. *script* de simulação *sim.do* (fornecido)
5. arquivo *wave.do* (fornecido)
6. relatório em formato PDF

▶ **O cenário de teste só será avaliado se o resultado corresponder ao resultado da simulação, isto é, o código deverá ser compilado e o simulador gerar as formas de onda presentes no relatório.**

▶ **Plágio é impossível, pois cada grupo necessariamente fará cenários de teste diferentes.**

▶ **Plágio em qualquer parte das entregas zera a nota para os grupos em que se detectar esta grave infração.**

2 Descrição do módulo a ser implementado

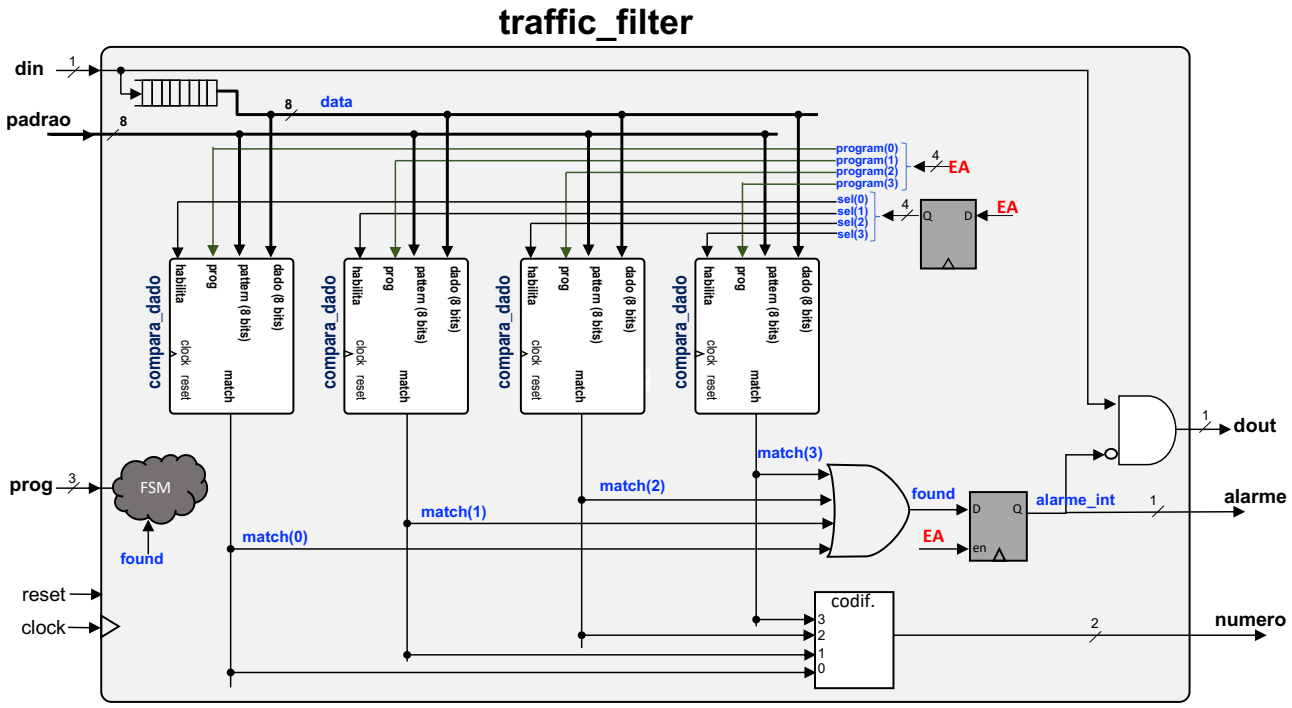
O ***traffic_filter*** em modo normal de operação transfere o fluxo serial de dados (***din***) para a saída (***dout***). O fluxo de dados é síncrono ao sinal de ***clock***, ou seja, em todo ciclo de ***clock*** há um bit de dados válido em ***din***.

Demais saídas:

- ***alarme*** – em caso de uma situação de ataque a saída ***alarme*** é ativada (nível lógico 1), e fluxo de dados de saída é bloqueado.
- ***número*** – indica o ***número*** do padrão que gerou o alarme.

Observar que ***din*** continuamente entra em um registrador de deslocamento de 8 bits, formando uma palavra da dados de 8 bits, denominado ***data***.

O ***traffic_filter*** possui 4 módulos *compara_dado*, podendo-se comparar o fluxo de entrada com até quatro padrões diferentes. O usuário pode selecionar em ***prog*** os valores {1,2,3,4} para programar e habilitar um determinado padrão.



Exemplos de codificação para os registradores e decodificador *(program)* que dependem do Estado Atual (EA)

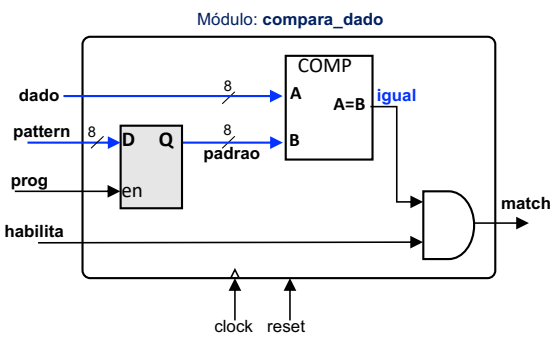
```
program(0) <= '1' when EA= xxxx else '0';
```

```
process(clock, reset)
 begin
  if reset = '1' then
     sel(0) <=  '0';
  elsif rising_edge(clock) then
    if EA=xxxx then
        sel(0) <= '1';
    elsif EA= xxxx then
        sel(0) <= '0';
    end if;
  end if;
```

```
process(clock, reset)
 begin
  if reset = '1' then
     alarme_int <=  '0';
  elsif rising_edge(clock) then
    if EA=xxxx then
        alarme_int <= found;
    elsif EA=xxxx or outras condições then
        alarme_in <= '0';
    end if;
  end if;
```

A primeira ação do usuário é **programar** os padrões a serem detectados no fluxo de dados. Os padrões ficam armazenados em um **módulo** denominado ***compara_dado*** (figura ao lado). Funcionamento do ***compara_dado***:

- Se *prog*='1' armazena-se a entrada *pattern* no registrador *padrao*;
- Se *habilita='1'* habilita-se a comparação entre o dado de entrada (*dado*) com o padrão armazenado (*padrao*);
- A saída do módulo ***compara_dado*** é o sinal *match*, o qual indica igualdade entre o valor armazenado (*padrao*) e o valor de entrada (*dado*), se o módulo está habilitado.



O barramento ***prog*** (3 bits) do ***traffic_filter*** indica o que o circuito deve fazer (comandos de entrada) para a FSM:

prog	**ação**
0	Não realiza nenhuma ação.
1	Armazena o padrão 1 no registrador interno do primeiro *compara_padrao* **e** ativa o sinal sel(0), o qual habilita a comparação no primeiro *compara_padrao*
2	Ação semelhante à *prog=1*, só que armazena e habilita a comparação com o padrão 2
3	Ação semelhante à *prog=1*, só que armazena e habilita a comparação com o padrão 3
4	Ação semelhante à *prog=1*, só que armazena e habilita a comparação com o padrão 4
5	Ativa o estado ***buscando.*** Caso ocorra a ocorrência de igualdade com algum padrão (***found=1***), a FSM deve ir para o estado ***bloqueio***
6	Zera *alarme_int*, fazendo a FSM voltar para ao estado ***buscando*** (só aceitar este valor estando no estado bloqueio)
7	Faz com que a FSM vá para um estado de ***zerar*** os registradores *alarme_int* e *sel* (só aceitar este valor estando nos estados ***buscando*** ou ***bloqueio***), e posteriormente ao zeramento dos registradores a FSM deve voltar para o estado inicial.

Após a programação ocorrer, o usuário seleciona ***prog=5***, indo para o estado ***buscando***. Neste estado o registrador *alarme_int* (e consequentemente a saída ***alarme***) passa a armazenar o valor da comparação *found*. Se *alarme_int* receber '1' o alarme é acionado e a saída é bloqueada (valor '0'). Acontecendo alarme (*found=1*), a FSM vai para o estado ***bloqueio***, e o usuário pode selecionar:

1. ***prog=6*** → a FSM voltar ao estado ***buscando***.
2. ***prog=7*** → a FSM vai para um estado de zerar os registradores *alarme_int* e *sel*, e posteriormente ao zeramento a FSM deve voltar para o estado inicial.

3 Configuração do Test Bench

- Para a criação de um **cenário de teste diferente** deve-se modificar no *test bench* o padrão de teste, onde: ***t*** indica o tempo em períodos de *clock*, ***prog*** indica o modo de configuração, e ***padrão*** o valor do padrão a ser armazenado, como mostrado abaixo:

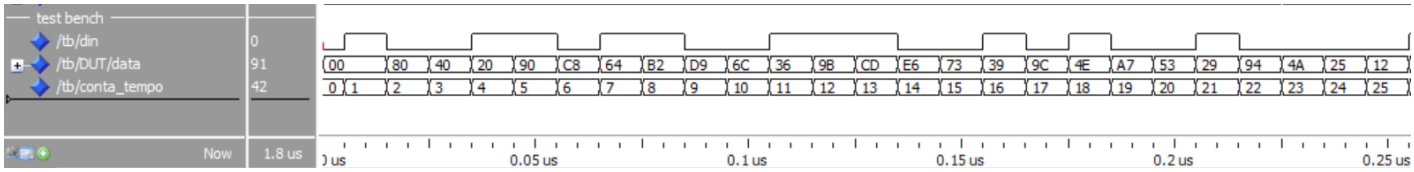
```
type padroes is array(natural range <>) of test_record;
 constant padrao_de_teste : padroes := (
  (t =>   4, prog => "001", padrao => x"24"),
  (t =>  10, prog => "010", padrao => x"B8"),
  (t =>  15, prog => "011", padrao => x"D4"),
  (t =>  20, prog => "100", padrao => x"65"),
  (t =>  35, prog => "101", padrao => x"FF"),    -- ATIVA A COMPARAÇÃO (5)
  (t =>  55, prog => "110", padrao => x"FF"),    -- reinicia a comparação (6)
  (t =>  70, prog => "110", padrao => x"FF"),    -- reinicia a comparação (6)
  (t => 105, prog => "111", padrao => x"FF")     -- reinicializa (7)
);
```

- A sequência de dados ***din*** é gerada de forma pseudoaleatória. Para alterar a sequência deve-se alterar a semente de geração de dados, isto é, o valor *seed*, como mostrado abaixo:

```
constant seed : std_logic_vector(GP-1 downto 0)      := "1101101110110110010";
```

- Como os valores gerados são pseudoaleatórios, sugere-se observar na forma de onda os valores gerados para escolher os padrões a serem comparados.

– A figura abaixo mostra o *din* sendo gerado, o *data* que é resultado do registrador de deslocamento, e o "conta_tempo" que auxilia a determinar o ***t*** *nos vetores de teste.*



4 Estrutura Sugerida para a Modelagem do *traffic_filter*

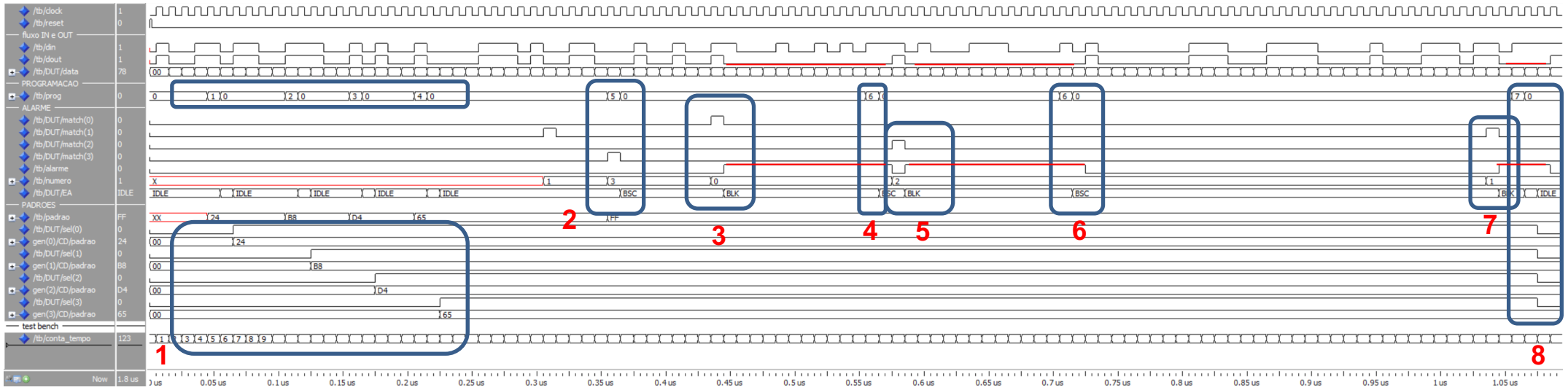
– A partir do desenho do *traffc_filter*, o código deve possuir:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity tp3 is
  port (clock   : in std_logic;
        reset   : in std_logic;
        . . .
        );
end entity;


architecture tp3 of tp3 is
  type state is ( . . .);
  signal EA, PE: state;
  . . .
```

```
begin
  --     REGISTRADOR DE DESLOCAMENTO QUE RECEBE O FLUXO DE ENTRADA
  --     4 PORT MAPS PARA OS compara_dado
  found   <=  . . .
  program(0) <= . . .
  program(1) <= . . .
  program(2) <= . . .
  program(3) <= . . .
  --     registradores para ativar as comparações (sel)
  --     registrador para o alarme interno
  --     MÁQUINA DE ESTADOS (FSM)
  --SAIDAS:
  alarme <= . . .
  dout   <= . . .
  numero <=  . . .

end architecture;
```

5 Exemplo de Simulação

Exemplo de simulação. Produzir no relatório um cenário <u>diferente deste</u>, mas este cenário deve funcionar na modelagem



<u>Eventos destacados:</u>

1. *prog* {1,2,3,4} programando os padrões e habilitando a comparação – observar que {sel(0), sel(1), sel(2), sel(3)} só voltam a zerar com o comando 7 (prog=7, no evento 8, ao final desta simulação)
2. início da busca por ataque, estado ***buscando*** – (BSC na figura) → prog=5
3. *match* com o <u>primeiro</u> padrão, ***número=0,*** com a máquina indo para o estado ***bloqueio*** (BLK na figura), e com o tráfego de saída bloqueado (**dout=0**, destacado em vermelho) e o alarme ativado (destacado em vermelho)
4. Prog=6, permitindo comparar novamente. Logo depois de liberar a comparação, houve um match com o padrão 2 (***número=2***)
5. Voltou-se novamente para o estado de ***bloqueio*** (BLK na figura) – tráfego bloqueado e alarme acionado
6. Prog=6, permitindo comparar novamente, estado ***buscando*** – (BSC na figura)
7. Match com o primeiro padrão, ***número=1,*** com a máquina indo para o estado ***bloqueio*** (BLK na figura) – tráfego bloqueado e alarme acionado
8. Circuito vai para o estado de ***zerar***, deixando passar o tráfego de entrada, zerando {sel(0), sel(1), sel(2), sel(3)}, e depois para o estado ***IDLE***.