Trabalho Prático em Introdução aos Sistemas Lógicos
Vernam Cypher: Impossível de Ser Quebrado

Gustavo Freitas Cunha

Matrícula: 2020054498

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brazil

gustavocunha@dcc.ufmg.br

Introdução

Nesta documentação serão apresentados aspectos e ideias da implementação das estruturas propostas, bem como os códigos estruturais, comportamentais, testbench, diagramas de tempo e resultados obtidos ao executar os códigos na plataforma 'EDA Playground'.
Todos os códigos possuem comentários de forma informativa e concisa, visando a contribuir para o entendimento da ideia da implementação.
Os arquivos de código também foram submetidos, mas recomenda-se que esta documentação seja lida, para maiores detalhes.

Atividade 1. Flip-Flop Tipo D

- Flip Flop Tipo D: Especificação Estrutural

```
//implementacao estrutural  do flip flop tipo d disparado por borda
ascendente

//latch basica com portas nor
module basic_latch (q, ql, s, r);
  //entradas e saidas
  input s,r;
  output q, ql;

  nor (ql, s, q);
  nor (q, ql, r);

endmodule

//flip flop composto de duas latches tipo d, com duas portas and e uma
latch basica cada
module d_flip_flop(q, ql, d, clk);
```

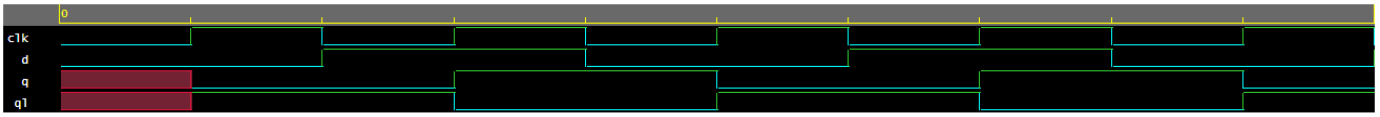
```
  //entradas e saidas
  input d, clk;
  output q, ql;

  //fios
  wire s, r, p, np, clkl, nd, s2, r2;

  not (clkl, clk);
  not (nd, d);

  //primeira latch do tipo d
  and (r, nd, clkl);
  and (s, d, clkl);
  basic_latch bl1 (p, np, s, r);

  //segunda latch do tipo d, cujas entradas sao saidas da primeira
  and(r2, np, clk);
  and(s2, p, clk);
  basic_latch bl2 (q, ql, s2, r2);

endmodule
```

- Flip Flop Tipo D: TestBench da Especificação Estrutural

```
module ff_testbench;
  reg d = 0;
  reg clk = 0;
  wire q, ql;

  //instanciacao do modulo
  d_flip_flop dff (.q(q), .ql(ql), .d(d), .clk(clk));

  //variacao do clock
  always #1
    begin
      clk = ~clk;
    end

  //variacao da entrada d, para teste
  always #2
    begin
```

```
      d = ~d;
    end

  //arquivo referente ao diagrama de tempo gerado
  initial begin
    $dumpfile("ff.vcd");
    $dumpvars(1);
    #10
    $finish;
  end

  //impressoes dos sinais no monitor para conferencia
  initial begin
    $monitor("time=%d: d= %b clk = %b q = %b,\n",$time,d,clk,q);
  end

endmodule
```

- Flip Flop Tipo D: Diagrama de Tempo da Especificação Estrutural



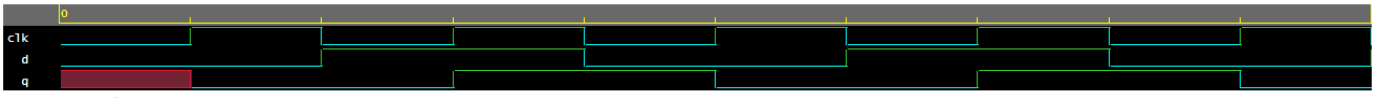
- Flip Flop Tipo D: Especificação Comportamental

```
//implementacao comportamental do flip flop tipo d disparado por borda
ascendente
module d_flip_flop(q, d, clk);
  input d, clk;
  output reg q;

  //sempre que ocorrer a borda positiva, o sinal de d eh amostrado na
saida
  always @(posedge clk)
    q <= d;

endmodule
```

- Flip Flop Tipo D: TestBench da Especificação Comportamental

```
module ff_testbench;
  reg d = 0;
  reg clk = 0;
  wire q;

  //instanciacao do modulo
  d_flip_flop dff (.q(q),.d(d),.clk(clk));

  //variacao do clock
  always #1
    begin
      clk = ~clk;
    end

  //variacao da entrada d, para teste
  always #2
    begin
      d = ~d;
    end

  //arquivo referente ao diagrama de tempo gerado
  initial begin
    $dumpfile("ff.vcd");
    $dumpvars(1);
    #10
    $finish;
  end

  //impressoes dos sinais no monitor para conferencia
  initial begin
    $monitor("time=%d: d= %b clk = %b q = %b,\n",$time,d,clk,q);
  end

endmodule
```

- Flip Flop Tipo D: Diagrama de Tempo da Especificação Comportamental



Atividade 2. Stream Cipher

Para esta parte do trabalho, foi implementado um registrador simples, utilizando os flip-flops tipo d implementados na Atividade 1. Seguem o código, testbench e diagrama de tempo do registrador.

- Registrador: Código da Implementação

```
//implementacao comportamental do flip flop tipo d disparado por borda
ascendente
module d_flip_flop(q, d, clk);
  input d, clk;
  output reg q;

  //sempre que ocorrer a borda positiva, o sinal de d eh amostrado na
saida
  always @(posedge clk)
    q <= d;

endmodule

module simple_register (q0, q1, q2, q3, d0, d1, d2, d3, clk);
  input d0, d1, d2, d3, clk;
  output q0, q1, q2, q3;

  //quatro flip flops controlados por um unico clock
  //cada flip flop recebe uma entrada e possui uma saida
  //entradas e saidas em paralelo
  d_flip_flop f1 (q0, d0, clk);
  d_flip_flop f2 (q1, d1, clk);
  d_flip_flop f3 (q2, d2, clk);
  d_flip_flop f4 (q3, d3, clk);

endmodule
```

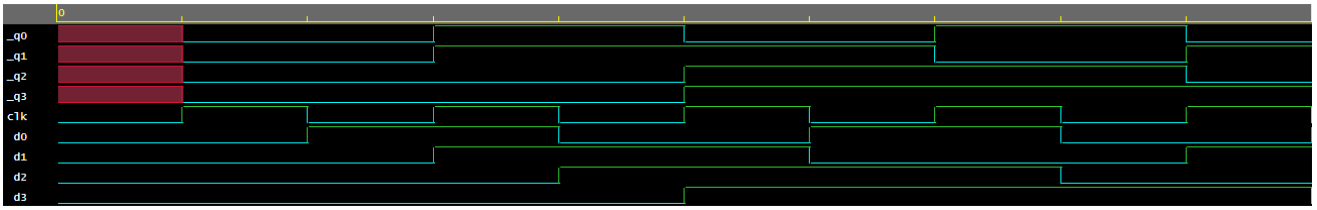
- Registrador: TestBench

```
module simple_register_testbench;
  reg d0 = 0;
  reg d1 = 0;
  reg d2 = 0;
  reg d3 = 0;
```

```
  reg clk = 0;
  wire _q0;
  wire _q1;
  wire _q2;
  wire _q3;

  //instanciacao do modulo
  simple_register sr (.q0(_q0), .q1(_q1), .q2(_q2), .q3(_q3), .d0(d0),
.d1(d1), .d2(d2), .d3(d3), .clk(clk));

  //variacao do clock
  always #1
    begin
      clk = ~clk;
    end

  //variacao das entradas, para teste
  always #2
    begin
      d0 = ~d0;
    end

  always #3
    begin
      d1 = ~d1;
    end

  always #4
    begin
      d2 = ~d2;
    end

  always #5
    begin
      d3 = ~d3;
    end

  //arquivo referente ao diagrama de tempo gerado
  initial begin
    $dumpfile("sr.vcd");
    $dumpvars(1);
    #10
    $finish;
```

```
  end

  //impressoes dos sinais no monitor para conferencia
  initial begin
    $monitor("time=%d: clk = %b q0 = %b q1 = %b q2 = %b q3 =
%b,\n",$time,clk,_q0,_q1,_q2,_q3);
  end

endmodule
```

- Registrador: Diagrama de Tempo



Agora, será apresentado o módulo que faz a cifragem e decifragem de uma mensagem através de uma chave aleatória. As cadeias de bits envolvidas são armazenadas em registradores. A mensagem, bem como a chave, têm quatro bits. Testes pseudo-aleatórios de chave foram gerados por meio de um software de computador e uma mensagem qualquer foi escolhida. Esses dados foram colocados como **entradas na testbench**, que chama, em um primeiro momento, o módulo para cifragem da mensagem e, em seguida, o módulo para decifragem da mensagem cifrada. As impressões dos resultados, cujos prints serão apresentados, mostram que a implementação funciona conforme o esperado.

- Cifrador e Decifrador: Código

```
//implementacao comportamental do flip flop tipo d disparado por borda
ascendente
module d_flip_flop(q, d, clk);
  input d, clk;
  output reg q;

  //sempre que ocorrer a borda positiva, o sinal de d eh amostrado na
saida
  always @(posedge clk)
    q <= d;

endmodule
```

```
module simple_register (q0, q1, q2, q3, d0, d1, d2, d3, clk);
  input d0, d1, d2, d3, clk;
  output q0, q1, q2, q3;

  //quatro flip flops controlados por um unico clock
  //cada flip flop recebe uma entrada e possui uma saida
  //entradas e saidas em paralelo
  d_flip_flop f1 (q0, d0, clk);
  d_flip_flop f2 (q1, d1, clk);
  d_flip_flop f3 (q2, d2, clk);
  d_flip_flop f4 (q3, d3, clk);

endmodule


module stream_cipher(msg_c0, msg_c1, msg_c2, msg_c3, ch_0, ch_1, ch_2,
ch_3, msg_0, msg_1, msg_2, msg_3, clk);

  input ch_0, ch_1, ch_2, ch_3, msg_0, msg_1, msg_2, msg_3, clk;
  output msg_c0, msg_c1, msg_c2, msg_c3;
  wire qc_0, qc_1, qc_2, qc_3;
  wire qm_0, qm_1, qm_2, qm_3;
  wire qmc_0, qmc_1, qmc_2, qmc_3;

  //armazenamento da chave em registrador
  simple_register c (qc_0, qc_1, qc_2, qc_3, ch_0, ch_1, ch_2, ch_3,
clk);

   //armazenamento da mensagem em registrador
  simple_register m (qm_0, qm_1, qm_2, qm_3, msg_0, msg_1, msg_2,
msg_3, clk);

  //cifragem dos bits da mensagem
  xor (qmc_0, qc_0, qm_0);
  xor (qmc_1, qc_1, qm_1);
  xor (qmc_2, qc_2, qm_2);
  xor (qmc_3, qc_3, qm_3);

   //armazenamento da mensagem cifrada em registrador
  simple_register mc (msg_c0, msg_c1, msg_c2, msg_c3, qmc_0, qmc_1,
qmc_2, qmc_3, clk);

endmodule
```

```
module stream_decipher(msg_0, msg_1, msg_2, msg_3, ch_0, ch_1, ch_2,
ch_3, msg_c0, msg_c1, msg_c2, msg_c3, clk);

  input ch_0, ch_1, ch_2, ch_3, msg_c0, msg_c1, msg_c2, msg_c3, clk;
  output msg_0, msg_1, msg_2, msg_3;

  wire qc_0, qc_1, qc_2, qc_3;
  wire qm_0, qm_1, qm_2, qm_3;
  wire qmc_0, qmc_1, qmc_2, qmc_3;

  //armazenamento da chave em registrador
  simple_register c (qc_0, qc_1, qc_2, qc_3, ch_0, ch_1, ch_2, ch_3,
clk);

   //armazenamento da mensagem cifrada em registrador
  simple_register mc (qmc_0, qmc_1, qmc_2, qmc_3, msg_c0, msg_c1,
msg_c2, msg_c3, clk);

  //decifragem dos bits da mensagem
  xor (qm_0, qc_0, qmc_0);
  xor (qm_1, qc_1, qmc_1);
  xor (qm_2, qc_2, qmc_2);
  xor (qm_3, qc_3, qmc_3);

   //armazenamento da mensagem decifrada em registrador
  simple_register m (msg_0, msg_1, msg_2, msg_3, qm_0, qm_1, qm_2,
qm_3, clk);

endmodule
```

- Cifrador e Decifrador: Testbench

```
module testebench_stream_cipher_decipher;
  //chave aleatoria
  reg ch_0 = 0;
  reg ch_1 = 1;
  reg ch_2 = 1;
  reg ch_3 = 1;
```

```
  //mensagem escolhida
  reg msg_0 = 0;
  reg msg_1 = 0;
  reg msg_2 = 0;
  reg msg_3 = 1;

  //mensagem (a ser) cifrada
  wire msg_c0, msg_c1, msg_c2, msg_c3;

  //clock iniciando em 0
  reg clk = 0;

  //instanciacao do modulo de cifragem
  stream_cipher cipher (.msg_c0(msg_c0), .msg_c1(msg_c1),
.msg_c2(msg_c2), .msg_c3(msg_c3),.ch_0(ch_0), .ch_1(ch_1), .ch_2(ch_2),
.ch_3(ch_3), .msg_0(msg_0), .msg_1(msg_1),.msg_2(msg_2), .msg_3(msg_3),
.clk(clk));

  //variacao do clock
  always #1
    begin
      clk = ~clk;
   end

  //arquivo referente ao diagrama de tempo gerado
  initial begin
    $dumpfile("ss.vcd");
    $dumpvars(1);
    #7
    $finish;
  end


  //mensagem (a ser) decifrada
  wire _msg_0, _msg_1, _msg_2, _msg_3;

  //instanciacao do modulo de decifragem
  stream_decipher decipher (.msg_0(_msg_0), .msg_1(_msg_1),
.msg_2(_msg_2), .msg_3(_msg_3), .ch_0(ch_0), .ch_1(ch_1), .ch_2(ch_2),
.ch_3(ch_3), .msg_c0(msg_c0), .msg_c1(msg_c1), .msg_c2(msg_c2),
.msg_c3(msg_c3), .clk(clk));
```

```
//impressoes dos sinais no monitor para conferencia da decifragem
  initial begin
    $monitor("time=%d: mensagem cifrada: %b %b %b %b mensagem
decifrada: %b %b %b %b clk = %b,\n",
    $time, msg_c0, msg_c1, msg_c2, msg_c3, _msg_0, _msg_1, _msg_2,
_msg_3, clk);
  end

endmodule
```

Note que, conforme foi dito, a chave (pseudo-aleatória) e a mensagem foram dadas como entradas na testbench. Veja, a seguir, o resultado da execução deste código, onde são impressos a mensagem cifrada (saída do módulo `stream_cipher`) e, na sequência, a mensagem decifrada (saída do módulo `stream_decipher`). Como se pode ver no código da testebench, os módulos são chamados em sequência, a fim de testar o funcionamento da implementação, a qual se mostrou plena, já que a mensagem decifrada é exatamente a mesma mensagem de entrada, settada na testbench.

- Cifrador e Decifrador: Saídas e Resultados Impressos

```
[2022-02-09 13:15:15 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile ss.vcd opened for output.
time=                    0: mensagem cifrada: x x x x mensagem decifrada: x x x x clk = 0,

time=                    1: mensagem cifrada: x x x x mensagem decifrada: x x x x clk = 1,

time=                    2: mensagem cifrada: x x x x mensagem decifrada: x x x x clk = 0,

time=                    3: mensagem cifrada: 0 1 1 0 mensagem decifrada: x x x x clk = 1,

time=                    4: mensagem cifrada: 0 1 1 0 mensagem decifrada: x x x x clk = 0,

time=                    5: mensagem cifrada: 0 1 1 0 mensagem decifrada: x x x x clk = 1,

time=                    6: mensagem cifrada: 0 1 1 0 mensagem decifrada: x x x x clk = 0,

time=                    7: mensagem cifrada: 0 1 1 0 mensagem decifrada: 0 0 0 1 clk = 1,

Done
```

Você pode testar o funcionamento do projeto settando outra mensagem na testbench e/ou gerando outra chave e settando-a, também, na testbench acima. Os arquivos de código também foram submetidos.
Note que há um delay devido à “camada” de abstração mais inferior possuir flip flops, mas o resultado é obtido com sucesso, após o tempo necessário de processamento e delay.