Universidade Federal do Rio Grande Do Norte

Campus Central (Natal)

Relatório Técnico

Ênfase em Mecatrônica

Aluno: Mateus de Assis Silva.

Professor: Samaherni Morais Dias

Disciplina: Circuitos Digitais - Laboratório

Implementação de Registrados Multifuncional em VHDL

Introdução:

Um dos elementos utilizados para adicionar memória nos nossos circuitos se chama *flip-flop*. Trata-se de uma combinação de portas lógicas capaz de manter informação binária enquanto estiver energizado e mudar seu estado quando sua entrada vai para nível lógico alto junto com um sinal de sincronia (*clock*). Para os nossos desenvolvimentos futuros, iremos considerá-los como foi definido [1]. O *flip flop* aqui utilizado terá as seguintes entradas: *J,K* (que manipula a mudança da saída), *preseting* (que força a saída pra um) e *clearing* (que força a saída para zero). Note que trata-se do *flip flop* JK.

Outro conceito importante é a de máquina de estados finitos. Essa é uma entidade abstrata capaz de alterar sua informação de saída dentre um conjunto de saídas possíveis quando uma dada condição de entrada é atendida. Assim, podemos adicionar um elemento temporal às nossas conhecidas tabelas-verdade [2].

Usando tais definições, podemos construir elementos que envolvam manter um sinal por tempo até serem solicitados que mudem. Assim sendo, desenvolveu-se um registrador de seis funções. Esse deverá suportar as funções em ordem de prioridade, sendo essa: manter informação, carregar informação, deslocar à direita, deslocar à esquerda, *set* síncrono e *clear* síncrono. A imagem do sistema completo se vê abaixo, conforme sugerido pelas recomendações da atividade.

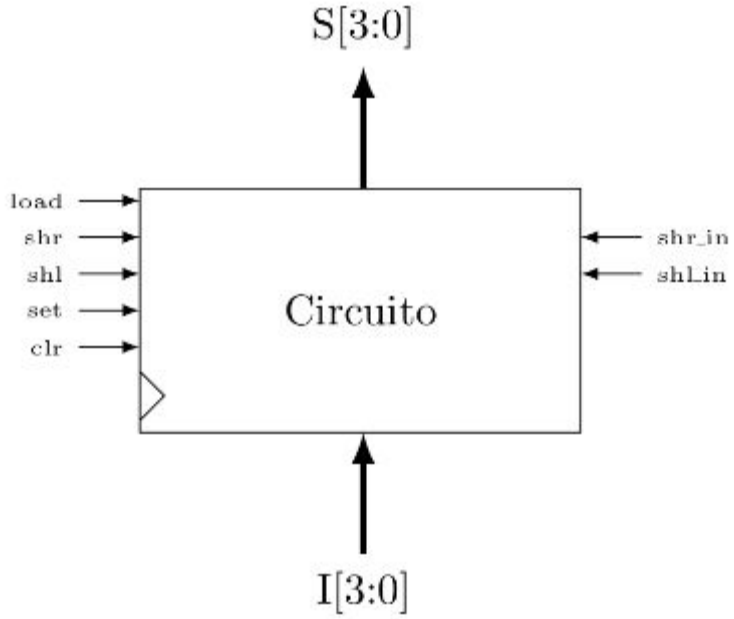


Figura 1: Registrador de seis funções.

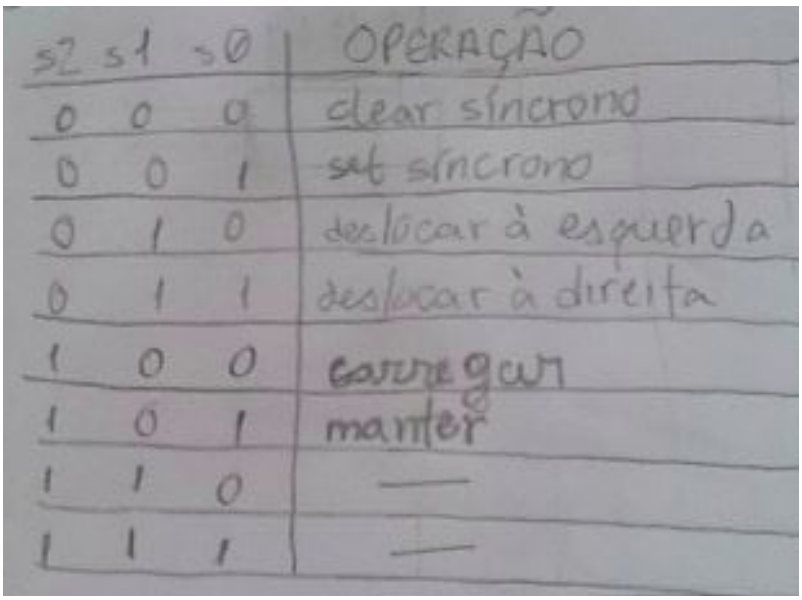
Para o desenvolvimento do sistema, decidiu-se colocar uma entrada a mais do que apresentado. Tal entrada será denominada “*keep*” e se refere à função de manter a saída. Escolheu-se pela adição desse grau de liberdade de forma a manter a prioridade da função, conforme exigido pela atividade.

Desenvolvimento:

Em primeira instância, é necessário determinar o tamanho do multiplexador a ser utilizado, de acordo com as funções desejadas. Ora, há seis funções, por ordem de prioridade: manter o valor atual, carregar, deslocar a direita, deslocar a esquerda, *set* síncrono (carregar a informação 1111 na próxima borda de relógio), e *clear* síncrono (carregar 0000 na próxima borda de relógio). Assim sendo, utilizaremos um multiplexador 8x1 para a implementação.

Agora que está definido qual multiplexador deve ser utilizado, definir-se-a a sua tabela de funções. Serão utilizadas as primeiras seis entradas do multiplexador de forma a implementar as aplicações. Vê-se, abaixo, a mencionada tabela.

Tabela 1: Funções do multiplexador 8x1.



s2	s1	s0	OPERAÇÃO
0	0	0	clear síncrono
0	0	1	set síncrono
0	1	0	deslocar à esquerda
0	1	1	deslocar à direita
1	0	0	carregar
1	0	1	manter
1	1	0	—
1	1	1	—

Com a tabela de funções definida, pode-se conectar as entradas do multiplexador. Por ilustração, vê-se abaixo o *n*-ésimo flip flop (FF) com as conexões já montadas.

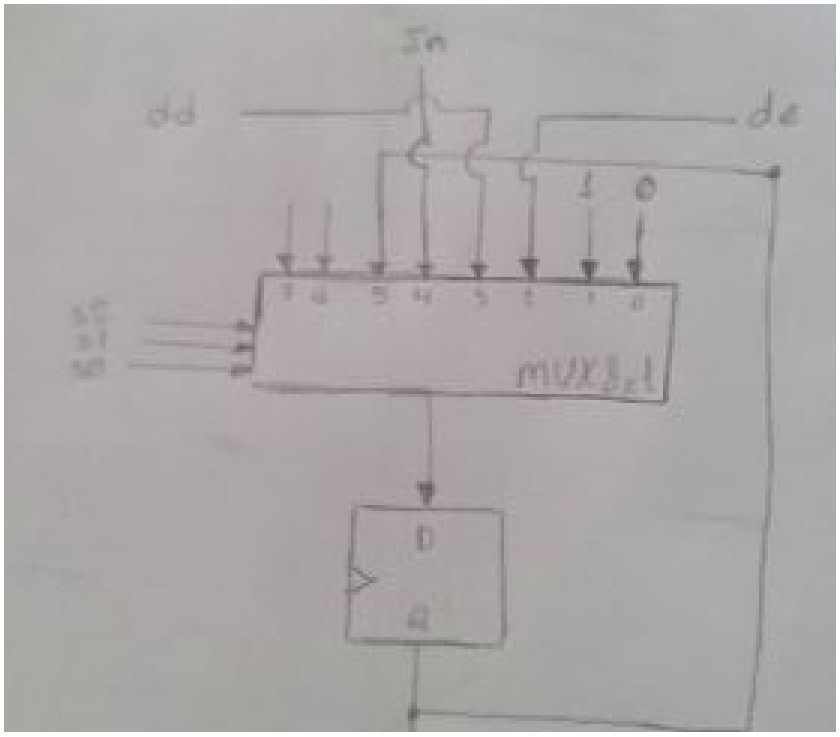


Figura 2: FF montado.

Agora que as conexões estão bem definidas, é necessário mapear as linhas de controle. Isso significa que é preciso definir uma lógica combinacional que leva as entradas de controle (relacionadas às 6 funções requeridas) para os controles do multiplexador (s2,s1 e s0). Para tanto, considere a tabela abaixo.

Tabela 2: Tabela Verdade para linhas de controle.

ENTRADAS						SAÍDAS
kp	Ld	dir	shL	sd	OPERAÇÃO	$\bar{n}\bar{n}R$
0	0	0	0	0	dr sinc.	000
0	0	0	0	1	sd sinc.	001
0	0	0	1	x	desl. esq.	010
0	0	1	x	x	desl. dir.	011
0	1	x	x	x	load	100
1	x	x	x	x	manter	101

A partir da tabela acima, é possível desenvolver as equações que controlam o multiplexador a partir das entradas do sistema. Tais equações são vistas abaixo.

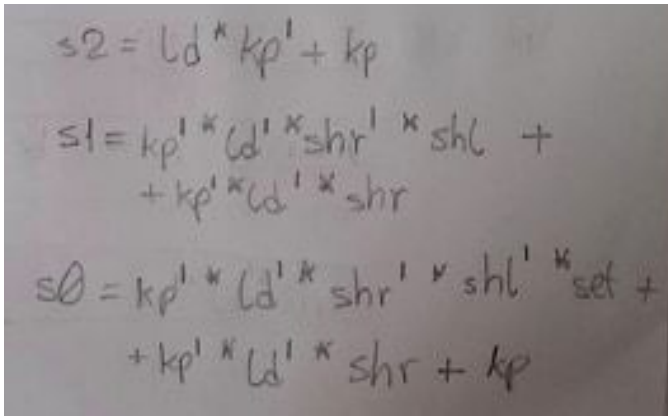


Figura 3: Equações de Mapeamento.

Com esses resultados em mãos, pode-se desenvolver o sistema completo. Entretanto, de forma a implementarmos as soluções previamente obtidas, considera-se que as entradas 6 e 7 dos multiplexadores também receberão o valor da saída do FF relacionado. Isso implica que, para *s2s1s0 = 111* e *s2s1s0 = 110* , a saída é mantida.

Considere também que a montagem apresentada na Figura 1, acima, pode ser abstraída como se vê abaixo. Para tal montagem, considera-se que as entradas *dd* e *de* se referem ao que foi transferido dos blocos direito e esquerdo. Além disso, foi-se omitidas as entradas unitária e nula do multiplexador, relativas ao *set* e *clear* síncronos.

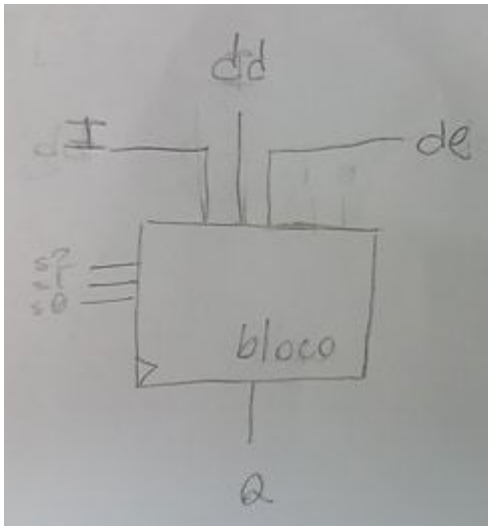


Figura 4: Bloco funcional.

Para a codificação dos comportamentos esperados, criou-se o multiplexador 8x1, seguido do bloco funcional. O flip flop D foi reaproveitado de projetos anteriores.

Os resultados das simulações se vêm abaixo, enquanto que os códigos empregados podem ser vistos no anexo A.

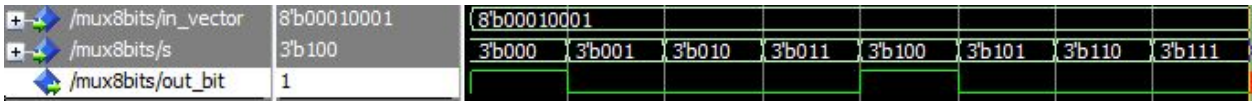


Figura 5: Simulação do Multiplexador 8x1.

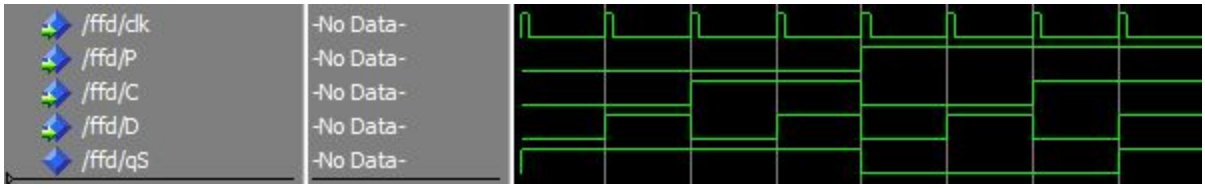


Figura 6: Simulação do *flip flop* D.

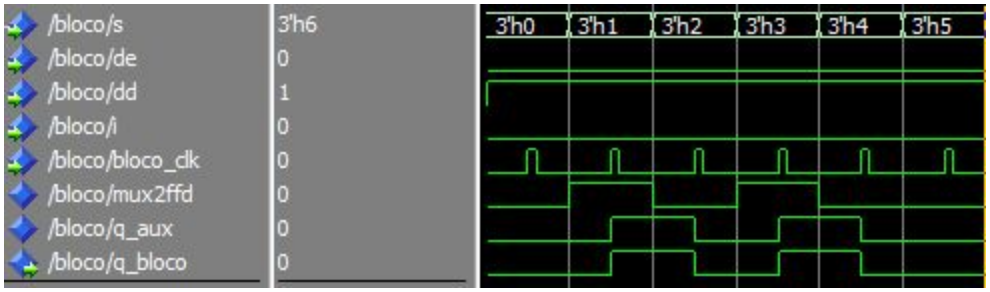


Figura 7: Simulação do Bloco Funcional.

Após a construção desses elementos básicos, deu-se início ao código do sistema completo. O diagrama desse modelo pode ser visto abaixo. Vê-se que as entradas estão acima, denominadas *I3*,*I2*,*I1*,*I0*. E abaixo estão as saídas. À direita, encontra-se o sinal *shr_in*, enquanto que à esquerda encontra-se o sinal *shl_in*.

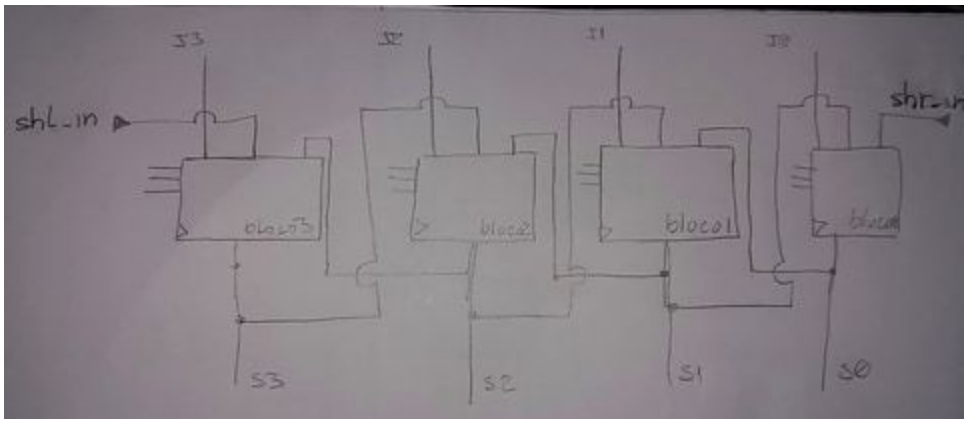


Figura 8: Sistema Completo.

Após a definição do sistema, passou-se à sua implementação em linguagem *vhdl*. O código se encontra em anexo, junto à sua simulação. O resultado dessa pode ser visto abaixo.

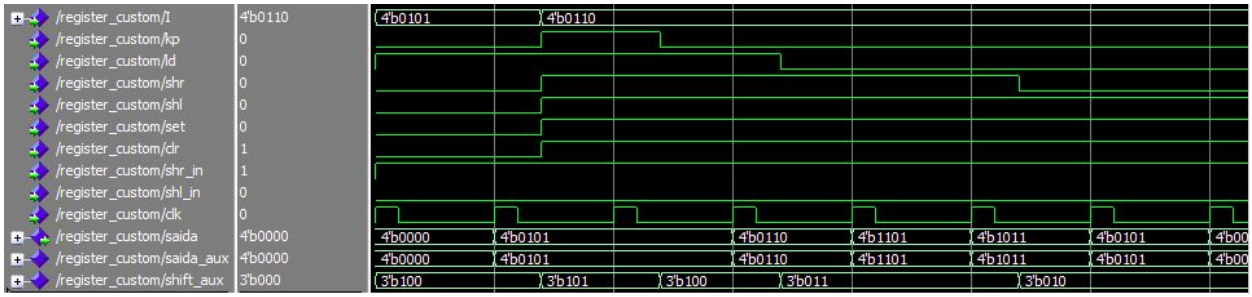


Figura 9: Resultados do sistema.

Para essa simulação, considera-se uma carga inicial, seguida de sinais unitários em todas as entradas do sistema. A partir daí, vai-se desligando as entradas uma a uma, seguindo a ordem de prioridade. Isso permite ver o sistema aguardando o sinal de relógio para aplicar a próxima função, além de tornar claro a precedência.

Conclusão:

O projeto tornou claro o uso de elementos de alto nível na confecção de sistemas mais complexos. Ao abstrairmos as funções de componentes eletrônicos, permite-se a construção de funcionalidades mais complicadas.

As simulações também tornaram claro o conceito de precedência. Ao utilizar o anulamento consecutivos de entradas prioritárias, viu-se a mudança no comportamento do circuito simulado. Isso, é claro, sincronizado pelo sinal de relógio.

Referências:

[1] Vahid, F. (2009). *Sistemas Digitais*. Bookman Editora. P.112 - 123.

[2] Vahid, F. (2009). *Sistemas Digitais*. Bookman Editora. P.129 - 151.

Anexo A: Códigos

```
-- MUX 8x1
-- For more information, go to github.com/mtxslv/DigitalCircuits
-- Code by Mateus de Assis Silva

entity MUX8bits is
      port(in_vector: in bit_vector(7 downto 0);
           s: in bit_vector(2 downto 0);
           out_bit: out bit);
end MUX8bits;

architecture MUX8bits_ckt of MUX8bits is
begin
      out_bit <= ( NOT(s(2)) AND NOT(s(1)) AND NOT(s(0)) AND in_vector(0) ) OR
                 ( NOT(s(2)) AND NOT(s(1)) AND s(0) AND in_vector(1) ) OR
                 ( NOT(s(2)) AND s(1) AND NOT(s(0)) AND in_vector(2) ) OR
                 ( NOT(s(2)) AND s(1) AND s(0) AND in_vector(3) ) OR
                 ( s(2) AND NOT(s(1)) AND NOT(s(0)) AND in_vector(4) ) OR
                 ( s(2) AND NOT(s(1)) AND s(0) AND in_vector(5) ) OR
                 ( s(2) AND s(1) AND NOT(s(0)) AND in_vector(6) ) OR
                 ( s(2) AND s(1) AND s(0) AND in_vector(7) );
end MUX8bits_ckt;
```

```
vsim MUX8bits

add wave *

force in_vector 11111111 0

force s(2) 0 0, 1 40
force s(1) 0 0, 1 20 -repeat 40
force s(0) 0 0, 1 10 -repeat 20

run 80
```

```
-- D flip flop
-- Code given by Samaherni M Dias
-- For more info, please access gihub.com/mtxslv/DigitalCircuits

ENTITY ffd IS
      port(clk, D, P, C: IN BIT;
                         q: OUT BIT);
END ffd;

ARCHITECTURE ckt OF ffd IS
SIGNAL qS: BIT;
BEGIN
  PROCESS(clk, P, C)
  BEGIN
    IF P = '0'    THEN qS <= '1';
    ELSIF C = '0' THEN qS <= '0';
    ELSIF clk='1' AND clk'EVENT THEN
      qS <= D;
    END IF;
  END PROCESS;
  q <= qS;
END ckt;
```

```
vsim ffd

add wave *

force P 0 0, 1 40 -repeat 80
force C 0, 1 20 -repeat 40
force D 0 0, 1 10 -repeat 20

force clk 1 0, 0 1 -repeat 10

run 80
```

```
-- functional block (bloco)
-- In order to work properly, make sure the files are in folder: MUX8bits.vhd
and ffd.vhd
-- This block connects a MUX8bits to a D flipflop (ffd). It supports the
operations (in order of multiplexing - most to least):
--    Keep the binary information (kp)
--    Load binary information (ld)
--    Shift rightward (shr)
--    Shift leftward (shl)
--    Synchronous setting (set)
--    Synchronous cleasing (clr)
-- Inputs: s(0, 1 and 2), dd(shr), de (shl) and in.
-- Outputs: q
-- wanna know more? Go to github.com/mtxslv/DigitalCircuits


entity bloco is
      port(s: in bit_vector(2 downto 0);
           dd,de,i, bloco_clk: in bit;
           q_bloco: out bit);
end bloco;

architecture bloco_ckt of bloco is

      component MUX8bits is
            port(in_vector: in bit_vector(7 downto 0);
                 s: in bit_vector(2 downto 0);
                 out_bit: out bit);
      end component;

      component ffd IS
            port(clk, D, P, C: IN BIT;
                              q: OUT BIT);
      END  component;

      signal mux2ffd, q_aux: bit;

begin

    mux: MUX8bits port map(in_vector(7) => q_aux,
                      in_vector(6) => q_aux,
                      in_vector(5) => q_aux,
                      in_vector(4) => i,
                      in_vector(3) => dd,
                      in_vector(2) => de,
                      in_vector(1) => '1',
                      in_vector(0) => '0',
                      s(2) => s(2),
```

```
                    s(1) => s(1),
                    s(0) => s(0),
                    out_bit => mux2ffd);
    ffd_lbl: ffd port map(clk => bloco_clk,
                    D => mux2ffd,
                    P => '1',
                    C => '1',
                    q => q_aux);

q_bloco <= q_aux;
end bloco_ckt;
```

```
vsim bloco

add wave *

force de 0 0
force dd 1 0
force i 0 0
force bloco_clk 0 0,1 5, 0 6 -repeat 10

force s(2) 0 0, 1 40
force s(1) 0 0, 1 20 -repeat 40
force s(0) 0 0, 1 10 -repeat 20

run 60
```

```
-- 6 function register
-- This register must has 6 functions, in priority order (most to least):
--    Keep the binary information (kp)
--    Load binary information (ld)
--    Shift rightward (shr)
--    Shift leftward (shl)
--    Synchronous setting (set)
--    Synchronous cleasing (clr)
-- In order to work properly, make sure the file are in folder: bloco.vhd
-- wanna know more? Go to github.com/mtxslv/DigitalCircuits

entity register_custom is
    port(I: in bit_vector(3 downto 0);
      kp, ld, shr, shl, set, clr, shr_in, shl_in, clk: in bit;
      saida: out bit_vector(3 downto 0));
end register_custom;

architecture register_custom_ckt of register_custom is

    signal saida_aux: bit_vector(3 downto 0);
    signal shift_aux: bit_vector(2 downto 0);

    component bloco is
       port(s: in bit_vector(2 downto 0);
            dd,de,i, bloco_clk: in bit;
            q_bloco: out bit);
    end component;

BEGIN

shift_aux(2) <=( ld AND NOT(kp) )OR kp;
shift_aux(1) <=( NOT(kp) AND NOT(ld) AND NOT(shr) AND shl )OR( NOT(kp) AND
NOT(ld) AND shr);
shift_aux(0) <= ( NOT(kp) AND NOT(ld) AND NOT(shr) AND NOT(shl) AND set )OR(
NOT(kp) AND NOT(ld) AND shr) OR kp;


bloco0: bloco port map (dd => shr_in,
             de => saida_aux(1),
             i => I(0),
             bloco_clk => clk,
             s(2) => shift_aux(2),
             s(1) => shift_aux(1),
             s(0) => shift_aux(0),
             q_bloco => saida_aux(0));

bloco1: bloco port map (dd => saida_aux(0),
             de => saida_aux(2),
```

```
            i => I(1),
            bloco_clk => clk,
            s(2) => shift_aux(2),
            s(1) => shift_aux(1),
            s(0) => shift_aux(0),
            q_bloco => saida_aux(1));

bloco2: bloco port map (dd => saida_aux(1),
            de => saida_aux(3),
            i => I(2),
            bloco_clk => clk,
            s(2) => shift_aux(2),
            s(1) => shift_aux(1),
            s(0) => shift_aux(0),
            q_bloco => saida_aux(2));

bloco3: bloco port map (dd => saida_aux(2),
            de => shl_in,
            i => I(3),
            bloco_clk => clk,
            s(2) => shift_aux(2),
            s(1) => shift_aux(1),
            s(0) => shift_aux(0),
            q_bloco => saida_aux(3));

saida(3) <= saida_aux(3);
saida(2) <= saida_aux(2);
saida(1) <= saida_aux(1);
saida(0) <= saida_aux(0);

end register_custom_ckt ;
```

```
vsim register_custom

add wave *

force shl_in 0 0
force shr_in 1 0
force I 2#0101 0, 2#0110  7

force clk 1 0, 0 1 -repeat 5

force kp  0 0, 1 7, 0 12
force ld  1 0, 1 7, 0 17
force shr 0 0, 1 7, 0 27
force shl 0 0, 1 7, 0 37
force set 0 0, 1 7, 0 47
force clr 0 0, 1 7


run 60
```