Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática – ICEI
Arquitetura de Computadores I

ARQ1 _ Aula_07
Tema: Introdução à linguagem Verilog e simulação em Logisim

Preparação

Como preparação para o início das atividades, recomendam-se

a.) leitura prévia do resumo teórico, do detalhamento na apostila e referências recomendadas
b.) estudo e testes dos exemplos
c.) assistir aos seguintes vídeos:

https://www.youtube.com/watch?v=_Wta-It79RU
https://www.youtube.com/watch?v=o8aHEaAsLw8
https://www.youtube.com/watch?v=bh1c5pv56IY

Orientação geral:

Apresentar todas as soluções em apenas um arquivo com formato texto (.txt).
Sugere-se usar como nome Guia_xx.txt, onde xx indicará o guia, exemplo Guia_01.txt.
Todos os arquivos deverão conter identificações iniciais com o nome e matrícula,
no caso de programas, usar comentários.
As implementações e testes dos exemplos em Verilog (**.v**) fornecidos como pontos de partida,
também fazem parte da atividade e deverão ter os códigos fontes entregues **separadamente**,
com o código fonte, a fim de que possam ser compilados e testados. Entregar os módulos de testes.
Sugere-se usar como nomes Guia_01yy.v, onde yy indicará a questão, exemplo Guia_0101.v
As saídas de resultados, opcionalmente, poderão ser copiadas ao final do código, em comentários.
Quaisquer outras anotações, observações ou comentários poderão ser colocadas em arquivo texto (README.txt) acompanhando a entrega.

Outras formas de solução serão **opcionais**; não servirão para substituir as atividades
a serem avaliadas. Caso entregues, poderão contar apenas como atividades extras.
Os programas com funções desenvolvidas em C, Java ou Python (c, .java, py),
como os modelos usados para verificação automática de testes das respostas;
caso entregues, também deverão estar em arquivos **separados**, com o código fonte e
módulos de testes, a fim de serem compilados e verificados.
As execuções deverão, preferencialmente, serão testadas mediante uso de redirecionamento de entradas e saídas padrões, cujos dados/resultados deverão ser armazenados em arquivos textos.
Os resultados poderão ser anexados ao código, ao final, como comentários.
Planilhas, caso venham a ser solicitadas, deverão ser **programadas** e/ou usar funções nativas.
Serão suplementares e opcionais, e deverão ser entregues em formato texto, preferencialmente,
com colunas separadas por tabulações ou no formato (**.csv**), acompanhando a solução em texto.
Arquivos em formato (**.pdf**), fotos, cópias de tela ou soluções manuscritas também poderão ser aceitos como recursos suplementares para visualização, mas não servirão como substitutos e **não** terão validade para fins de avaliação.
Os *layouts* de circuitos deverão ser entregues no formato (**.circ**), identificados internamente.
Figuras exportadas pela ferramenta serão aceitas apenas como arquivos para visualização,
mas **não** terão validade para fins de avaliação. Separar versões completas (a) e simplificadas (b).

Projeto de circuitos

Codificadores e decodificadores

Sequências de ***bits*** podem ser usadas para codificar valores numéricos. Dois códigos binários têm aplicações especiais: o BCD (Binary-Coded Decimal) e o de Gray.

Código BCD

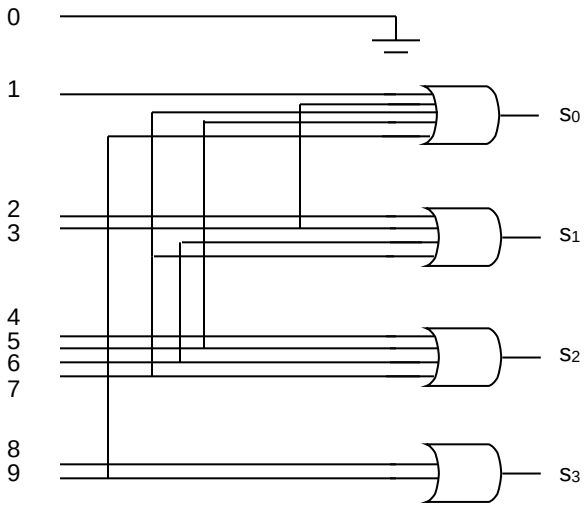
O código BCD é basicamente uma forma de codificar valores numéricos na base 10 em seus equivalentes binários.

| Decimal | BCD |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Exemplo:

Codificar o valor decimal 369 em código BCD: 369 = (3)(6)(9) = (0011) (0110) (1001)$_{(2)}$

Um circuito codificador "*decimal-para-BCD*" é aquele capaz de mapear um conjunto de entradas (0-9) em um outro conjunto de quatro valores binários ($s_3s_2s_1s_0$), se apenas uma das entradas for acionada.



Código de Gray

O código de Gray serve para expressar sequências binárias nas quais dois valores sucessivos tenham apenas um ***bit*** de diferença (distância de Hamming = 1).

| Decimal | Binário | Gray |
|---|---|---|
| 0 | 000 | 000 |
| 1 | 001 | 001 |
| 2 | 010 | 011 |
| 3 | 011 | 010 |
| 4 | 100 | 110 |
| 5 | 101 | 111 |
| 6 | 110 | 101 |
| 7 | 111 | 100 |

Esse código também é conhecido com "*código binário refletido*", por causa da característica abaixo:

eixo de reflexão

| | | |
|---|---|---|
| | | **0**00 |
| | | **0**01 |
| | **0**0 | **0**11 |
| 0 | **0**1 | **0**10 |
| 1 | **1**1 | **1**10 |
| | **1**0 | **1**11 |
| | | **1**01 |
| | | **1**00 |

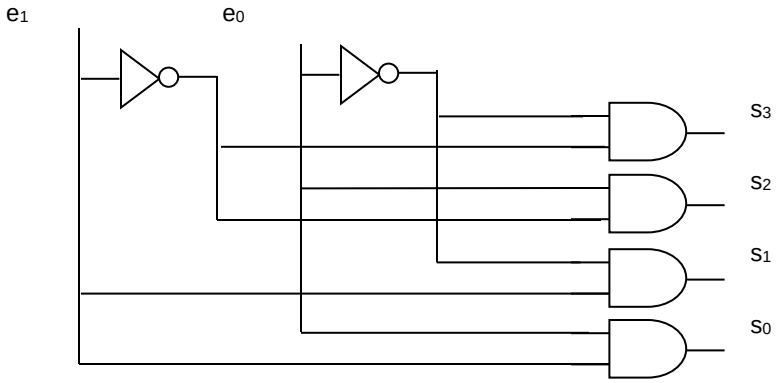
Decodificadores em geral são circuitos lógicos capazes de ativar uma saída de acordo com uma seleção de sinais de entrada.

Um decodificador de nível alto ativa uma saída quando uma das entradas estiver em nível 1 e as outras em nível 0.

Exemplo:

Montar um decodificador em nível alto para a tabela abaixo:

| $e_1$ | $e_0$ | $s_3\ s_2\ s_1\ s_0$ |
|---|---|---|
| 0 | 0 | 1 0 0 0 |
| 0 | 1 | 0 1 0 0 |
| 1 | 0 | 0 0 1 0 |
| 1 | 1 | 0 0 0 1 |

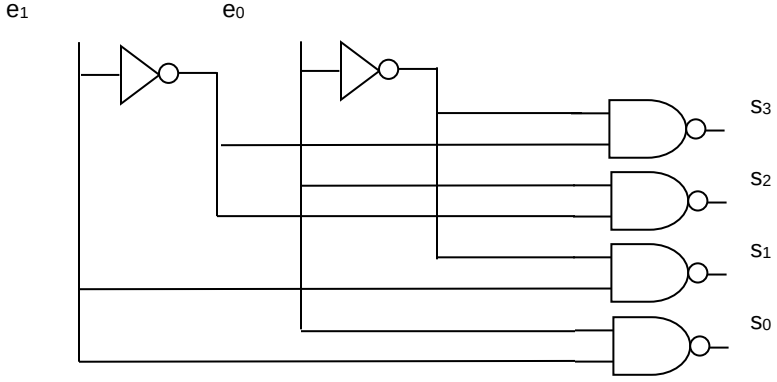



Um decodificador de nível baixo ativa uma saída quando uma das entradas estiver em nível 0 e as outras em nível 1.

Exemplo:

Montar um decodificador em nível baixo para a tabela abaixo:

| $e_1$ | $e_0$ | $s_3\ s_2\ s_1\ s_0$ |
|---|---|---|
| 0 | 0 | 0 1 1 1 |
| 0 | 1 | 1 0 1 1 |
| 1 | 0 | 1 1 0 1 |
| 1 | 1 | 1 1 1 0 |

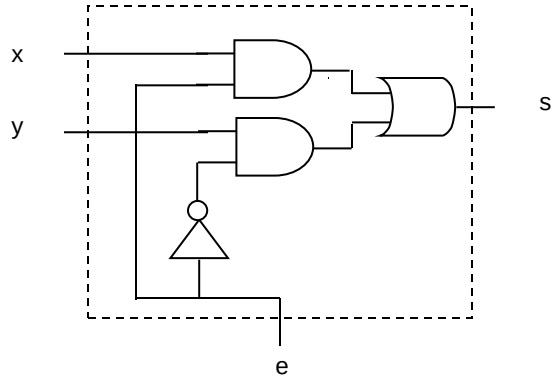


Multiplexadores (MUX) e demultiplexadores (DEMUX)

Multiplexadores (ou seletores de dados) são circuitos lógicos capazes de atuar como chaves digitais: recebem várias entradas e selecionam uma delas, em certo instante, e realizam sua transferência para a saída, mediante um código de seleção. Podem ser usados para rotear dados, sequenciar operações, realizar conversões do tipo paralelo-série e gerar tabelas ou formas de ondas.

Exemplo: MUX2x1

Dados dois sinais de entrada ( x ) e ( y ), escolher uma saída mediante sinal de seleção (e).

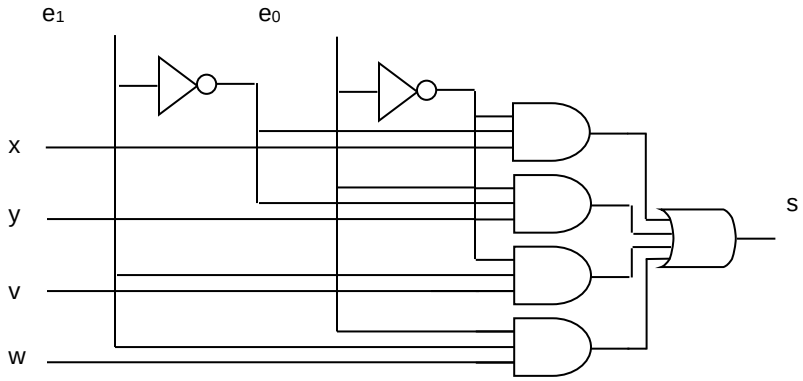



| e | s = x • e + y • e’ |
|---|---|
| 0 | y |
| 1 | x |

Multiplexadores podem selecionar mais sinais dependendo do número de bits (tamanho) da chave de seleção.

Exemplo: MUX4x1

Dados quatro sinais de entrada ( x,y,v,w ), escolher uma saída mediante seleção ($e_1$,$e_0$).

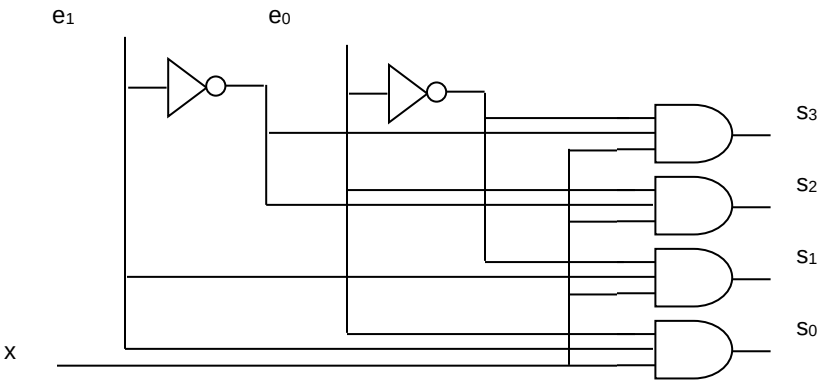



| $e_1$ | $e_0$ | s = x•$e_1$'•$e_0$' + y•$e_1$'•$e_0$ + v•$e_1$•$e_0$' + w•$e_1$•$e_0$ |
|---|---|---|
| 0 | 0 | x |
| 0 | 1 | y |
| 1 | 0 | v |
| 1 | 1 | w |

Demultiplexadores (ou distribuidores de dados) são circuitos capazes de receber um sinal de entrada e distribuí-lo em uma dentre várias saídas, segundo um código de seleção. Podem ser usados para distribuir um mesmo sinal de ativação ou sequenciamento (***clock***) para vários circuitos.

Exemplo: DEMUX1x4

Dados um sinal de entrada ( x ) e dois sinais de ativação $e_0$ e $e_1$, distribuí-lo à saída.




| $e_1$ | $e_0$ | $e_1$ $e_0$ x | $s_3$ $s_2$ $s_1$ $s_0$ |
|---|---|---|---|
| 0 | 0 | $s_3$ = 0 0 X | X 0 0 0 |
| 0 | 1 | $s_2$ = 0 1 X | 0 X 0 0 |
| 1 | 0 | $s_1$ = 1 0 X | 0 0 X 0 |
| 1 | 1 | $s_0$ = 1 1 X | 0 0 0 X |

Atividade: Projeto de unidade lógica

Para os exercícios a seguir, considerar o exemplo abaixo em Verilog.

```
// -------------------------
// Guia_0700 - GATES
// Nome: xxx yyy zzz
// Matricula: 999999
// -------------------------

// -------------------------
//  f7_gate
// -------------------------
module f7 ( output s,
            input  a,
            input  b );

// descrever por portas

endmodule // f7


// -------------------------
//  multiplexer
// -------------------------
module mux ( output s,
             input  a,
             input  b,
             input  select );

// definir dados locais
  wire not_select;
  wire sa;
  wire sb;

// descrever por portas
  not NOT1 ( not_select, select );

  and AND1 ( sa, a, not_select );
  and AND2 ( sb, b,    select     );

  or  OR1     ( s , sa, sb );
endmodule // mux


module test_f7;
// ------------------------- definir dados
    reg  x;
    reg  y;
    reg  s;
    wire w;
    wire z;

    f7 modulo   ( w, x, y );

    mux MUX1 ( z, x, y, s );
```

```
// ------------------------- parte principal

   initial
   begin : main
         $display("Guia_0700 - xxx yyy zzz - 999999");
         $display("Test LU's module");
         $display("   x     y    s    z");

          x = 1'b0;  y = 1'b1;  s = 1'b0;

       // projetar testes do modulo
   #1    $monitor("%4b %4b %4b %4b", x, y, s, z);
   #1     s = 1'b1;

   end

endmodule // test_f7
```

Exercícios

01.) Projetar e descrever em Verilog, usando portas **portas nativas**,
uma unidade lógica (LU) com operações AND e NAND, inicialmente
com duas saídas independentes (paralelas, 2 respostas),
para variáveis de entrada com 01 com bit cada.
Acrescentar escolha que permita apenas uma saída selecionável (1 resposta),
segundo escolha feita por uma chave de seleção.
O nome do arquivo deverá ser Guia_0701.v,
e poderá seguir o modelo descrito abaixo.
Incluir previsão de testes.
Simular o módulo no Logisim e apresentar *layout* do circuito e subcircuitos.
DICA: Usar para o sinal extra para a seleção (0-NAND; 1-AND).
NÃO usar switch-case ou operador ternário.

02.) Projetar e descrever em Verilog, usando **portas nativas**,
uma unidade lógica (LU) com operações OR e NOR,
com uma saída selecionável (1 resposta), para variáveis de entrada com 01 bit cada.
O nome do arquivo deverá ser Guia_0702.v.
Incluir previsão de testes.
Simular o módulo no Logisim e apresentar *layout* do circuito e subcircuitos.
DICA: Usar para o sinal extra para a seleção (0-NOR; 1-OR).
NÃO usar switch-case ou operador ternário.

03.) Projetar e descrever em Verilog, usando **portas nativas**,
uma unidade lógica (LU) com o acréscimo das operações
AND e NAND, com uma saída só para ambas, para variáveis de entrada com 01 bit cada,
além de OR e NOR na mesma situação. Os resultados de cada grupo serão
selecionados por uma primeira chave (2x1); para selecionar uma porta em cada grupo,
e outra chave (2x1) que selecionará entre o grupo (AND,NAND) ou o grupo (OR,NOR).
O nome do arquivo deverá ser Guia_0703.v.
Incluir previsão de testes.
Simular o módulo no Logisim e apresentar *layout* do circuito e subcircuitos.
DICA: Usar para um sinal para a seleção de portas (0-NAND/NOR ; 1-AND/OR).
Usar para outro sinal para a seleção de grupo (0-AND/NAND ; 1-OR/NOR;).
NÃO usar switch-case ou operador ternário.

04.) Projetar e descrever em Verilog, usando **portas nativas**,
uma unidade lógica (LU) com o acréscimo das operações
XOR e XNOR, , com uma saída só para ambas, para variáveis de entrada com 01 bit cada,
além de OR E NOR; na mesma situação. Os resultados de cada grupo serão selecionáveis
entre o grupo (OR, NOR) ou o grupo (XOR, XNOR), seleção (4x1).
O nome do arquivo deverá ser Guia_0704.v.
Incluir previsão de testes.
Simular o módulo no Logisim e apresentar *layout* do circuito e subcircuitos.
DICA: Usar para o sinal extra de 2 bits para a seleção (00-NOR; 01-OR; 10-XOR; 11-XNOR).
NÃO usar switch-case ou operador ternário.

05.) Projetar e descrever em Verilog, usando **portas nativas**,
uma unidade lógica (LU) com o acréscimo das operações
NOT, AND, NAND, OR, NOR, XOR, XNOR, simultâneas, com apenas 1 saída,
selecionável (7x1).
O nome do arquivo deverá ser Guia_0705.v.
Incluir previsão de testes.
Simular o módulo no Logisim e apresentar *layout* do circuito e subcircuitos.
DICA: Usar um sinal extra de 3 bits para a seleção.
Sobrará uma chave de seleção, e poderá ser empregada para a negação de outro operando.
NÃO usar switch-case ou operador ternário.

Extras

06.) Projetar e descrever em Verilog, usando portas nativas,
uma unidade lógica (LU) com um comparador de bits
usando portas XOR e XNOR para
calcular a igualdade ou desigualdade, respectivamente,
para duas variáveis com 4 bits cada,
selecionável (0-igual; 1-diferente).
O nome do arquivo deverá ser Guia_0706.v.
Incluir previsão de testes.
Simular o módulo no Logisim e
apresentar *layout* do circuito e subcircuitos.
DICA: Montar a tabela-verdade e identificar os mintermos.

07.) Projetar e descrever em Verilog, usando portas nativas,
uma unidade lógica (LU) com um comparador de bits
usando apenas portas básicas (NOT, AND, OR) para
calcular a magnitude (se maior ou menor),
para duas variáveis com 4 bits cada,
selecionável (1-menor; 0-maior).
O nome do arquivo deverá ser Guia_0707.v.
Incluir previsão de testes.
Simular o módulo no Logisim e
apresentar *layout* do circuito e subcircuitos.
DICA: Montar a tabela-verdade e identificar os mintermos.