Standard Componenets

Carmi Merimovich

The Academic College of Tel-Aviv

January 29, 2025

Basic Combinational Circuits

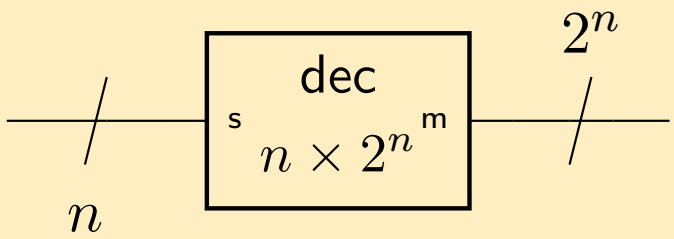
Detailed veriloging

1. Decoder
2. Encoder
3. Priority Encoder
4. Multiplexer
5. Demultiplexer

Realizing the Decoder

1. Block diagram
2. Logic circuit
3. Module
4. Testbench

Decoder



For each $i < 2^n$ the following holds:

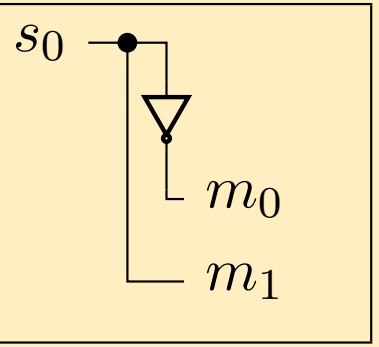
$$m[i] = \begin{cases} 0 & i \neq s, \\ 1 & i = s, \end{cases}$$

where $s = \sum_{j=0}^{n-1} s[j] \times 2^j$.

Decoder 1×2 (circuit)

Decoder 1×2

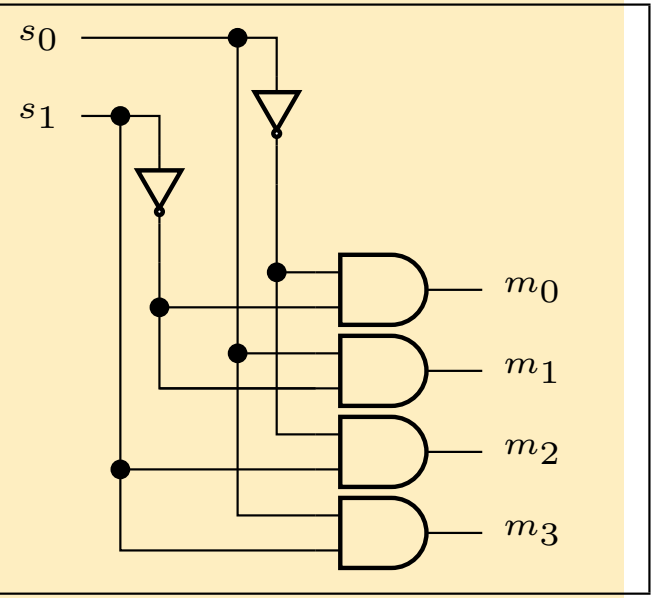
s_0	m_0	m_1
0	1	0
1	0	1



Decoder 2×4 (circuit)

Decoder 2×4

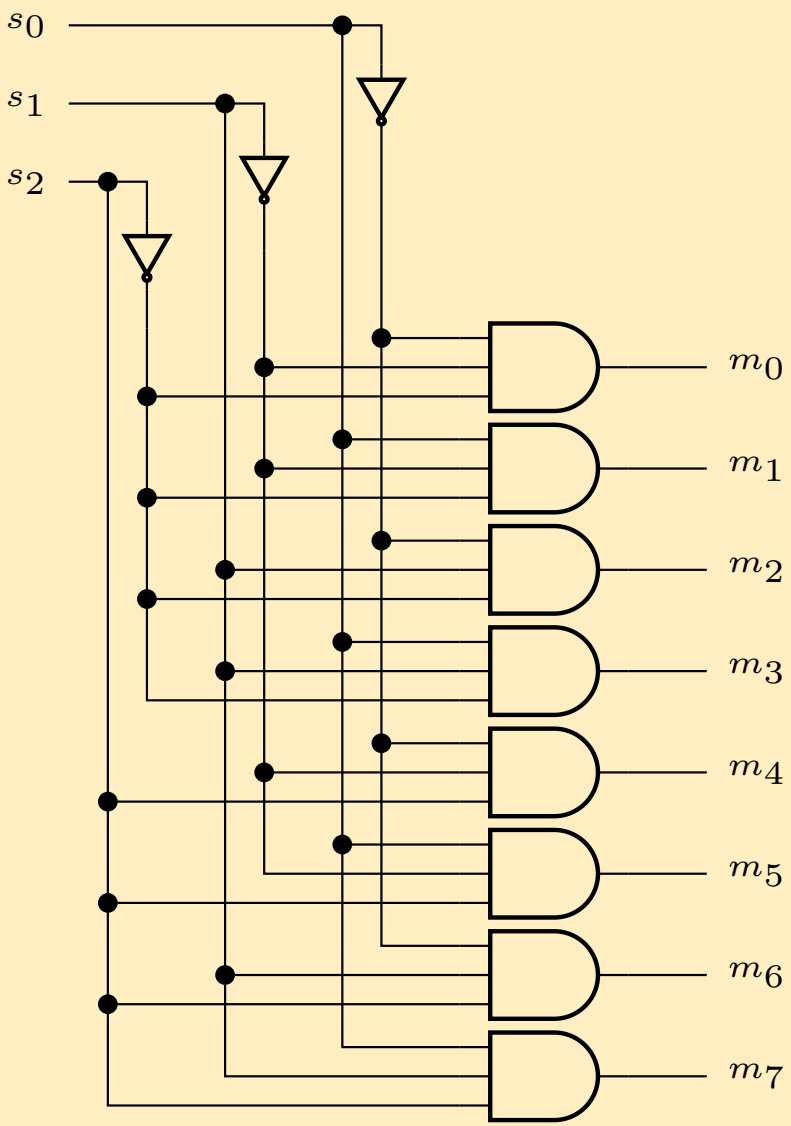
s_1	s_0	m_0	m_1	m_2	m_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decoder 3 $\times 8$ (Truth table)

s_2	s_1	s_0	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder 3×8 (circuit)



(Carmi) Lecture 10 reached here

Decoder $n \times 2^n$

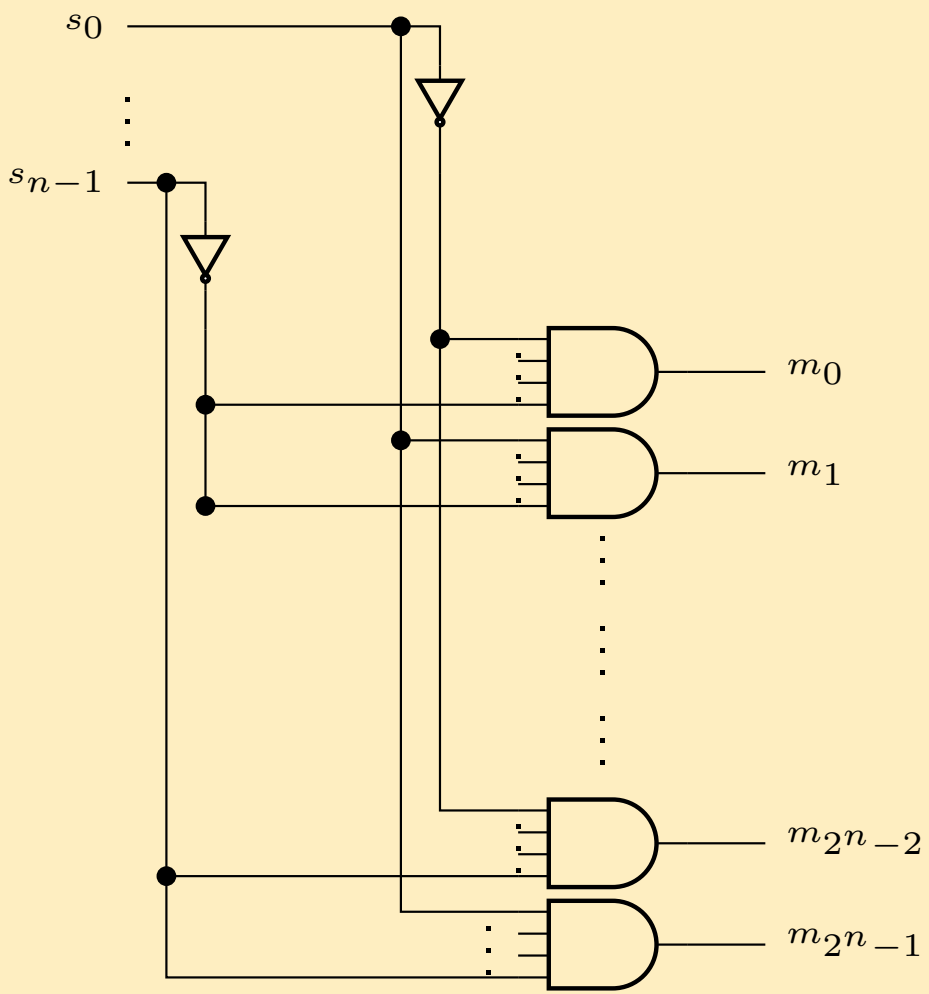
- The input variables are: $s_0, \ldots, s_{n-1}$.
- The output functions are $m_0, \ldots, m_{2^n-1}$.
- For each $i < 2^n$,

$$m_i = s^*_{n-1} \cdot \cdots \cdot s^*_0,$$

where $i = (b_{n-1} \cdot \cdots \cdot b_0)_2$ and for each $j < n$,

$$s^*_j = \begin{cases} \bar{s}_j & \text{if } b_j = 0, \\ s_j & \text{if } b_j = 1. \end{cases}$$

Decoder $n \times 2^n$ (circuit)



Decoder $n \times 2^n$ (verilog) I

Listing 1: decoder.v

```
‘timescale 1ns/1ns

module decoder #(parameter SEL=1) (
  output [2**SEL-1:0] m,
  input [SEL-1:0] sel
);

  wire [SEL-1:0] seln;

  for (genvar b = 0; b < SEL; b++)
    notGate n(seln[b], sel[b]);

  for (genvar i = 0; i < 2**SEL; i++) begin
```

Decoder $n \times 2^n$ (verilog) II

```
    wire [SEL-1:0] s;
    for (genvar b = 0; b < SEL; b++) begin
      if ((i & (2**b)) !== 0)
        assign s[b] = sel[b];
      else
        assign s[b] = seln[b];
    end
    andMulti #(SEL) a(m[i], s);
  end
endmodule
```

Decoder $n \times 2^n$ (testbench) I

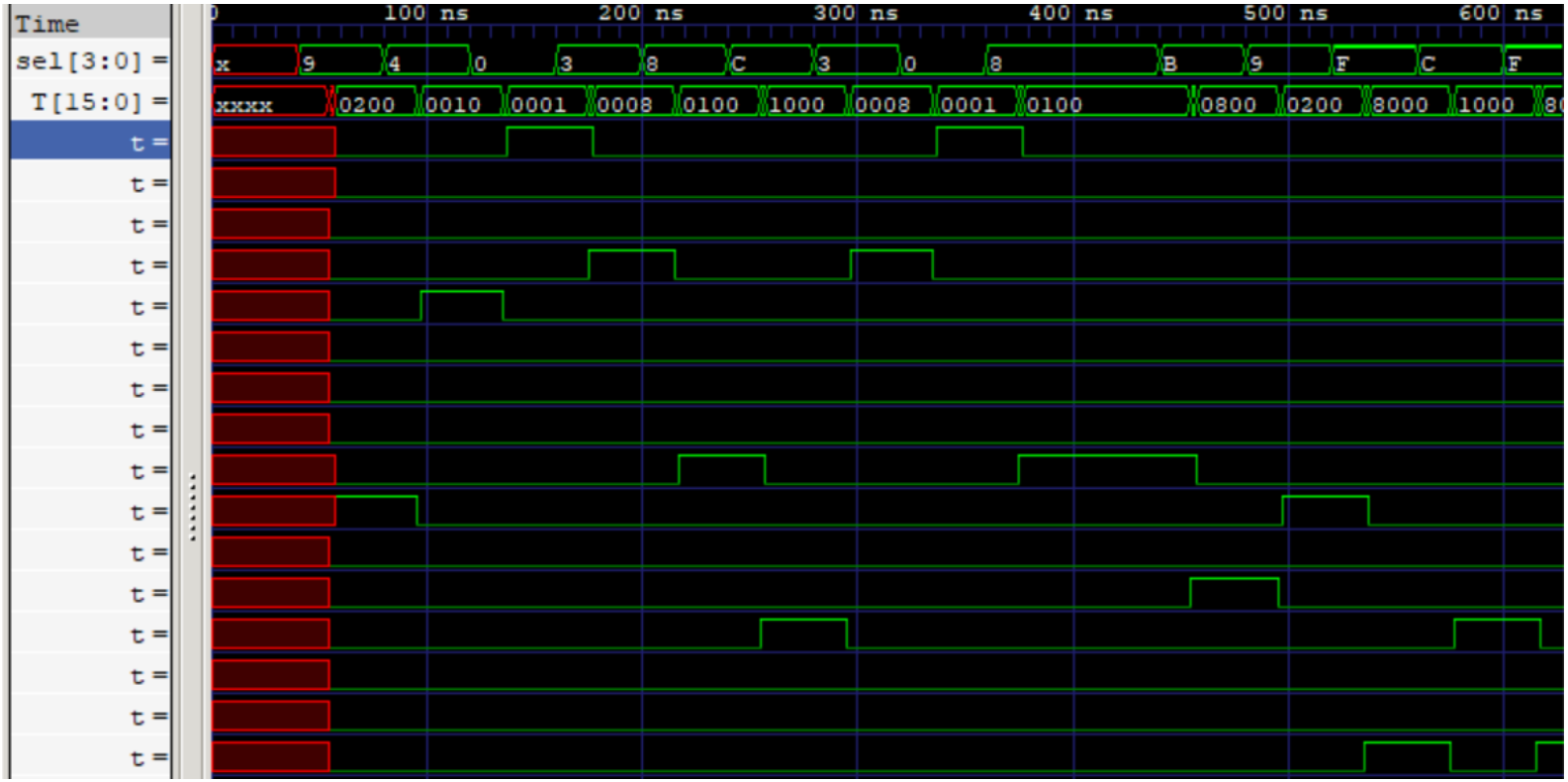
Listing 2: decoder_tb.v

```
‘timescale 1ns/1ns

module decoder_tb;
  parameter SEL = 5;

  logic [SEL-1:0] sel;
  wire [2**SEL-1:0] T;
  decoder  #(SEL) d(T, sel);

  initial begin
    $dumpfile("decoder.vcd");
    $dumpvars;
```

Decoder $n \times 2^n$ (testbench) II

```
    repeat (100) begin
      sel = $urandom_range(0,2**SEL-1);
      #40ns;
      $display("%d %x", sel, T);
    end
    $finish();
  end

endmodule
```

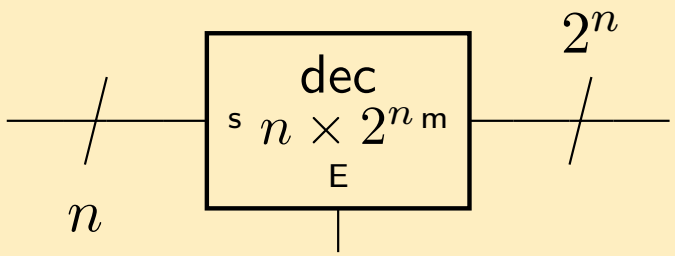
Decoder 4×16 waveform



Realizing the Decoder with Enable

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

Decoder with Enable

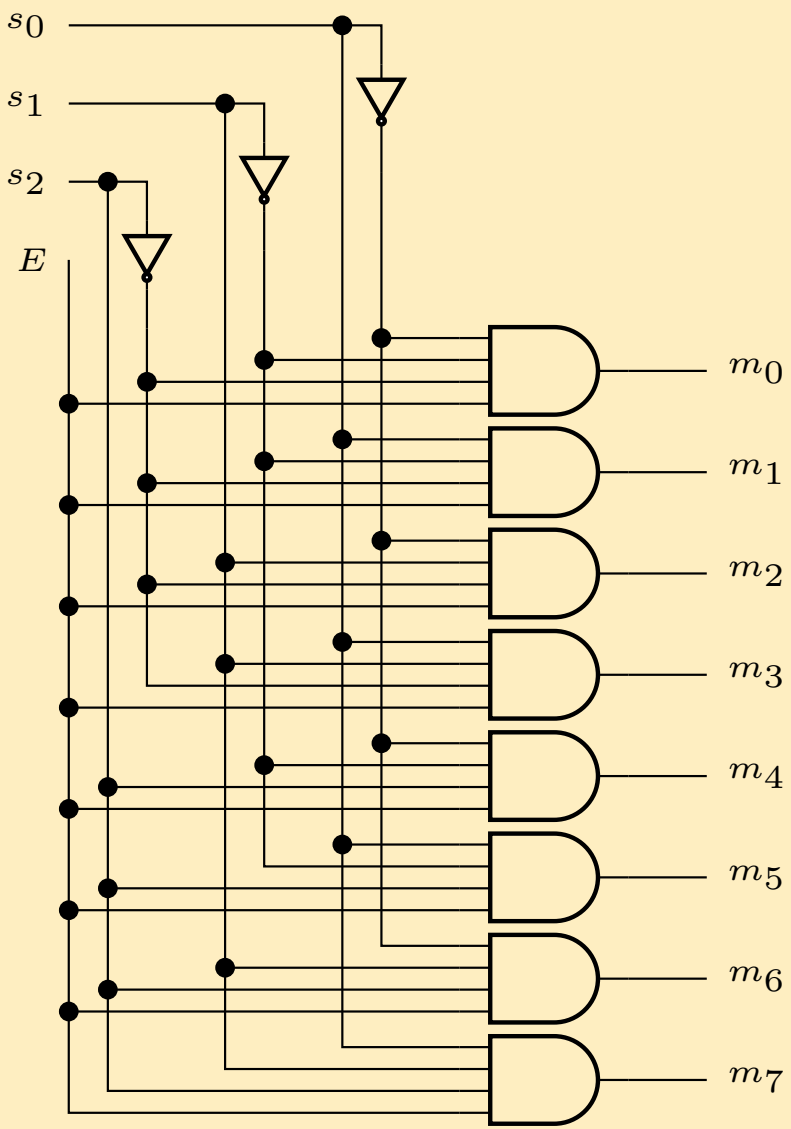


For each $i < 2^n$ the following holds:

$$m[i] = \begin{cases} 0 & i \neq s \text{ or } E = 0, \\ 1 & i = s \text{ and } E = 1, \end{cases}$$

where $s = \sum_{j=0}^{n-1} s[j] \times 2^j$.

Decoder 3×8 with Enable (circuit)



Decoder with Enable $n \times 2^n$ (module) I

Listing 3: decoderE.v

```
‘timescale 1ns/1ns

module decoderE #(parameter SEL=1) (
  output [2**SEL-1:0] m,
  input [SEL-1:0] sel,
  input E
);
  wire [2**SEL-1:0] t;

  decoder #(SEL) de(t, sel);

  for (genvar i = 0; i < 2**SEL; i++)
    andGate a(m[i], t[i], E);
```

Decoder with Enable $n \times 2^n$ (module) II

```
endmodule
```

Decoder with Enable $n \times 2^n$ (testbench) I

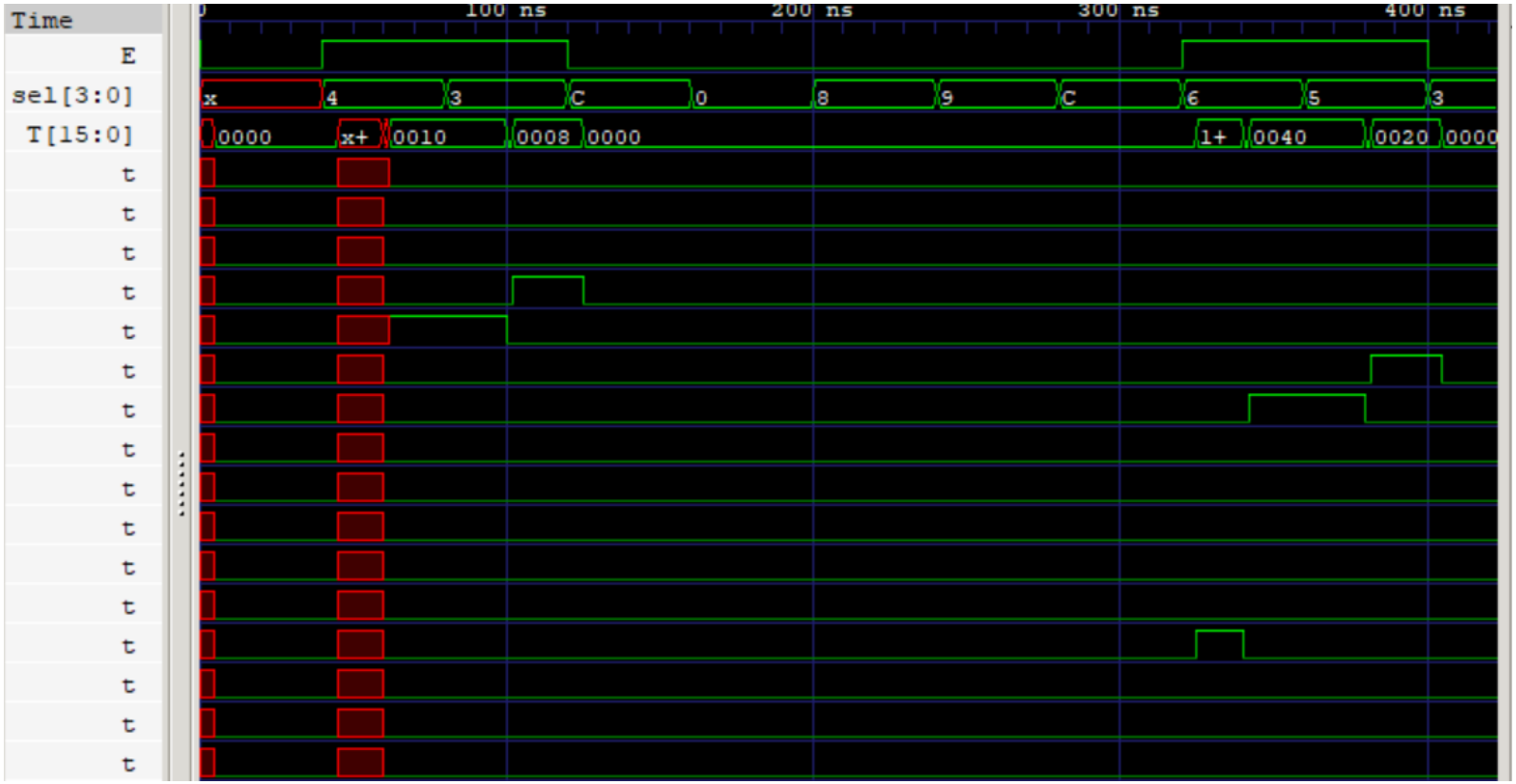
Listing 4: decoderE_tb.v

```
`timescale 1ns/1ns

module decoderE_tb;
  parameter SEL = 4;

  logic E;
  logic [SEL-1:0] sel;
  wire [2**SEL-1:0] T;
  decoderE  #(SEL) d(T, sel, E);

  initial begin
    $dumpfile("decoderE.vcd");
    $dumpvars;
```

Decoder with Enable $n \times 2^n$ (testbench) II

```
    E = 0;
    repeat (100) begin
      if ($urandom_range(0,1)==0)
        E = E + 1;
      sel = $urandom_range(0,2**SEL-1);
      #40ns;
      $display("%d %d %x", E, sel, T);
    end
    $finish();
  end
endmodule
```

Decoder with Enable 4×16 waveform



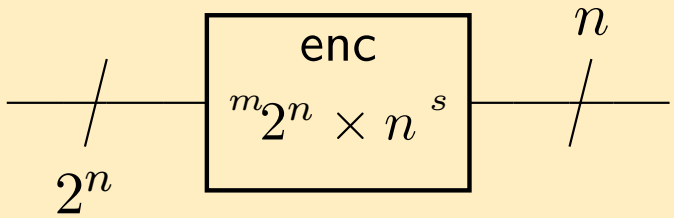
In class dicussion

1. Implement a boolean function with decoder help
2. Implement a decoder with smaller decoders

Realizing the Encoder

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

Encoder

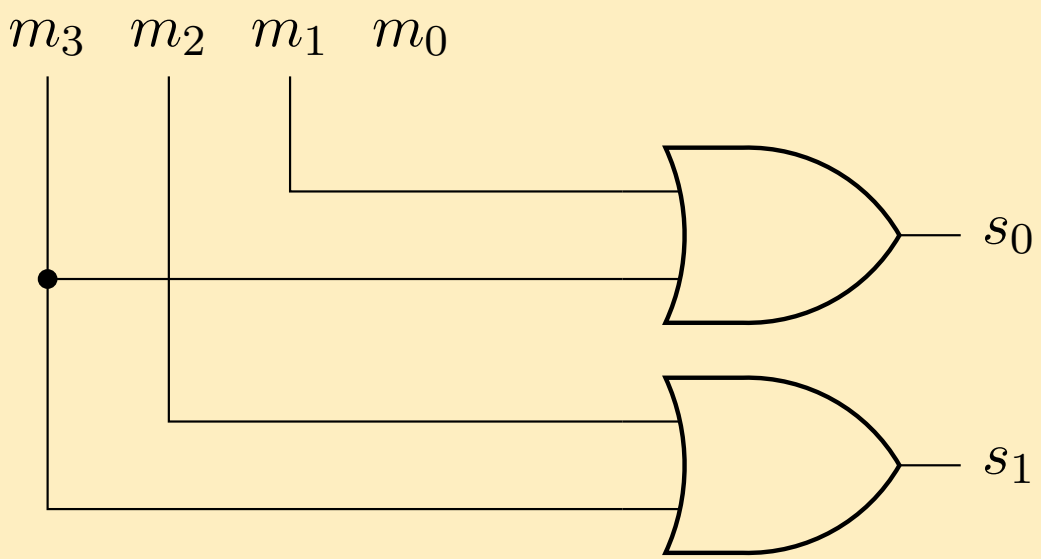


For $i < 2^n$,

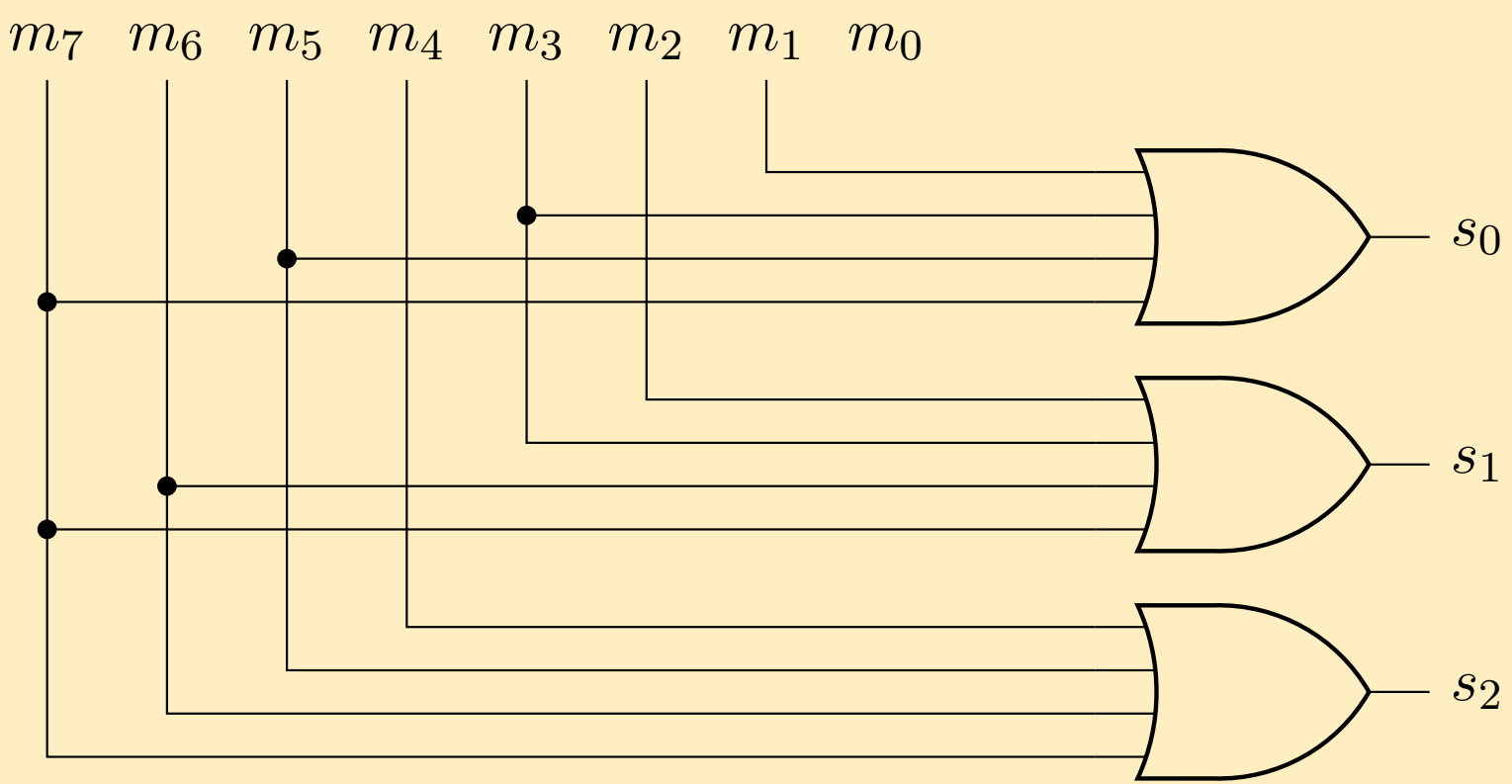
$$s = i \text{ if } \forall j < 2^n \; m[j] = \begin{cases} 1 & i = j, \\ 0 & i \neq j, \end{cases}$$

where $s = \sum_{j=0}^{n-1} s[j] \times 2^j$.
s is undefined if $m = 0$ or more than one bit of m is one

Encoder 4 $\times 2$ circuit



Encoder 8 $\times 3$ circuit



Encoder (verilog) I

Listing 5: encoder.v

```
‘timescale 1ns/1ns

module encoder #(parameter SEL=1) (
  output [SEL-1:0] sel,
  input [2**SEL-1:0] m
);
  for (genvar b = 0; b < SEL; b++) begin
    wire [2**SEL-1:0] consider;
    for (genvar i = 0; i < 2**SEL; i++) begin
      if ((i & (2**b)) !== 0)
        assign consider[i] = m[i];
      else
        assign consider[i] = 0;
```

Encoder (verilog) II

```
        end
        orMulti #(2**SEL) o(sel[b],
                                consider);
    end
endmodule
```

Encoder (testbench) I

Listing 6: encoder_tb.v

```
`timescale 1ns/1ns

module encoder_tb;
  parameter SEL = 4;

  logic [2**SEL-1:0] in;
  wire [SEL-1:0] sel;

  encoder #(SEL) e(sel, in);

  initial begin
    $dumpfile("encoder.vcd");
    $dumpvars;
```

Encoder (testbench) II

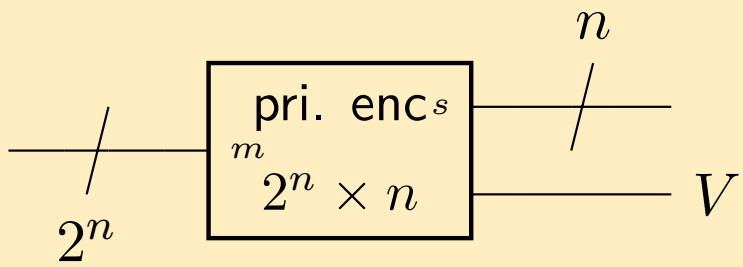
```
    in = 1;
    repeat (2**SEL) begin
      #50ns;
      $display("%x %d", in, sel);
      in = in * 2;
    end
    $finish();
end

endmodule
```

(Carmi) Lecture 11 reached here

Realizing the Priority Encoder

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

Priority Encoder



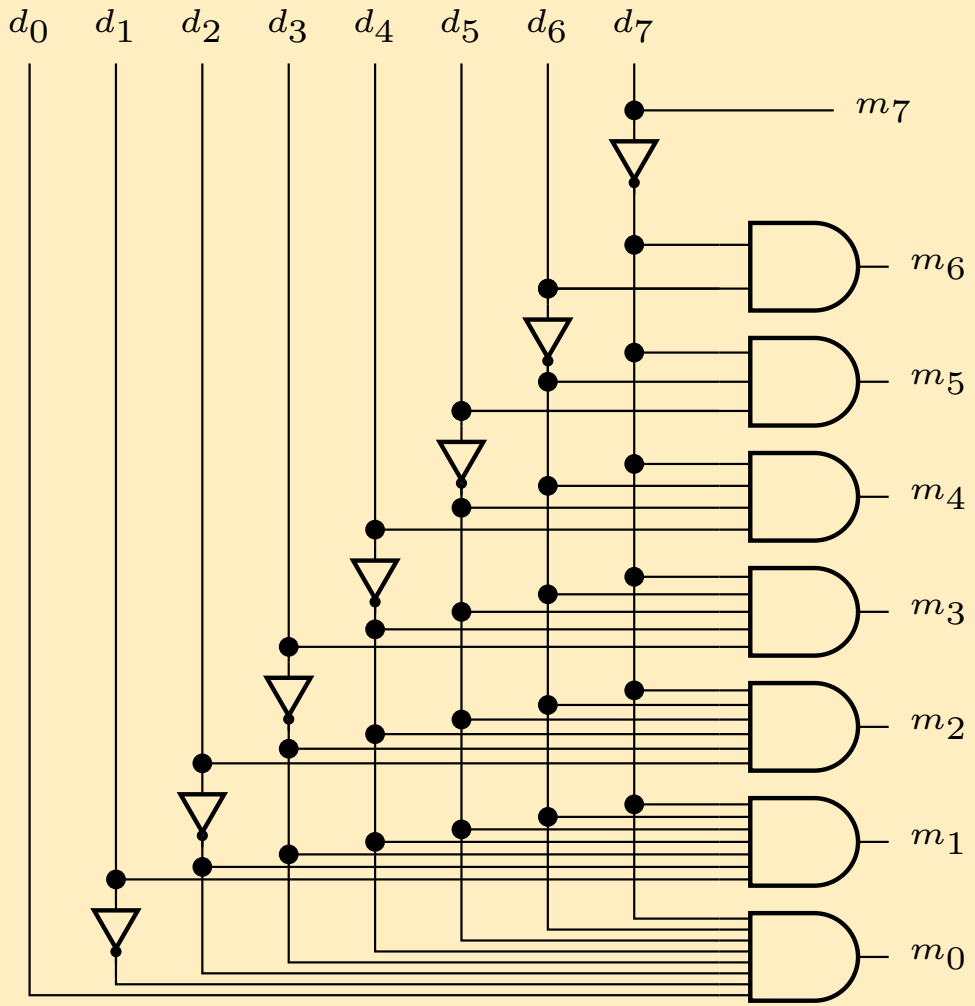
$$V = \begin{cases} 1 & \exists i < 2^n \ m[i] = 1, \\ 0 & \text{otherwise} \end{cases}$$

and

$$s = \begin{cases} \max\{i \mid m[i] = 1\} & \text{if } V = 1, \\ \text{undefined} & \text{if } V = 0. \end{cases}$$

Pri. encoder 8 × 3 (circuit)

The Fun Way (i.e., using an encoder)



Priority Encoder (module) I

Listing 7: encoderV.v

```
`timescale 1ns/1ns

module encoderV #(parameter SEL=1) (
  output [SEL-1:0] sel,
  output v,
  input [2**SEL-1:0] d
);
  orMulti #(2**SEL) om(v, d);

  wire [2**SEL-1:0] dn;
  for (genvar i = 0; i < 2**SEL; i++)
    notGate n(dn[i], d[i]);
```

Priority Encoder (module) II


```
// Leave only leftmost asserted bit
wire [2**SEL-1:0] m;
for (genvar i = 0; i < 2**SEL; i++) begin
  wire [2**SEL-1:0] e;

  for (genvar j = 0; j < i; j++)
    assign e[j] = 1;

  assign e[i] = d[i];
  for (genvar j = i + 1; j < 2**SEL; j++)
    assign e[j] = dn[j];

  andMulti #(2**SEL) a(m[i], e);
end
```

Priority Encoder (module) III

```
  encoder #(SEL) ee(sel, m);

endmodule
```

Priority Encoder (testbench) I

Listing 8: encoderV_tb.v

```
`timescale 1ns/1ns

module encoderV_tb;
  parameter SEL = 4;

  logic [2**SEL-1:0] in;
  wire [SEL-1:0] sel;
  wire v;

  encoderV  #(SEL) e(sel, v, in);

  initial begin
    $dumpfile("encoderV.vcd");
```

Priority Encoder (testbench) II

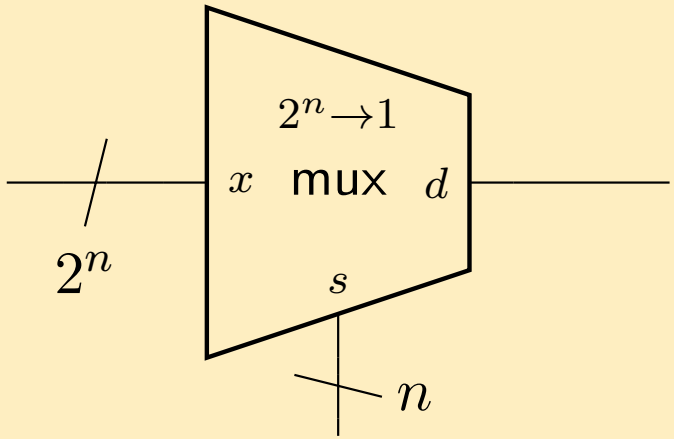
```
    $dumpvars;

    repeat (2**SEL) begin
      in = $urandom_range(0, 2**(2**SEL) - 1);
      #200ns;
      $display("%x %d %d", in, v, sel);
    end
    $finish();
end

endmodule
```

(Carmi) Lecture 12 reached here

Realizing the Multiplexer

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

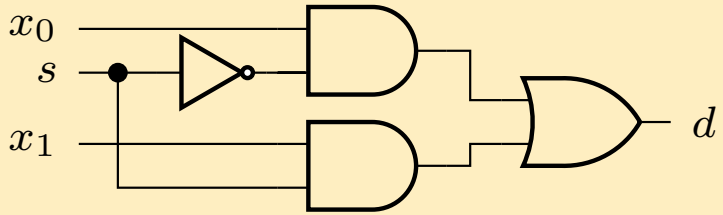
Multiplexer $2^n \to 1$



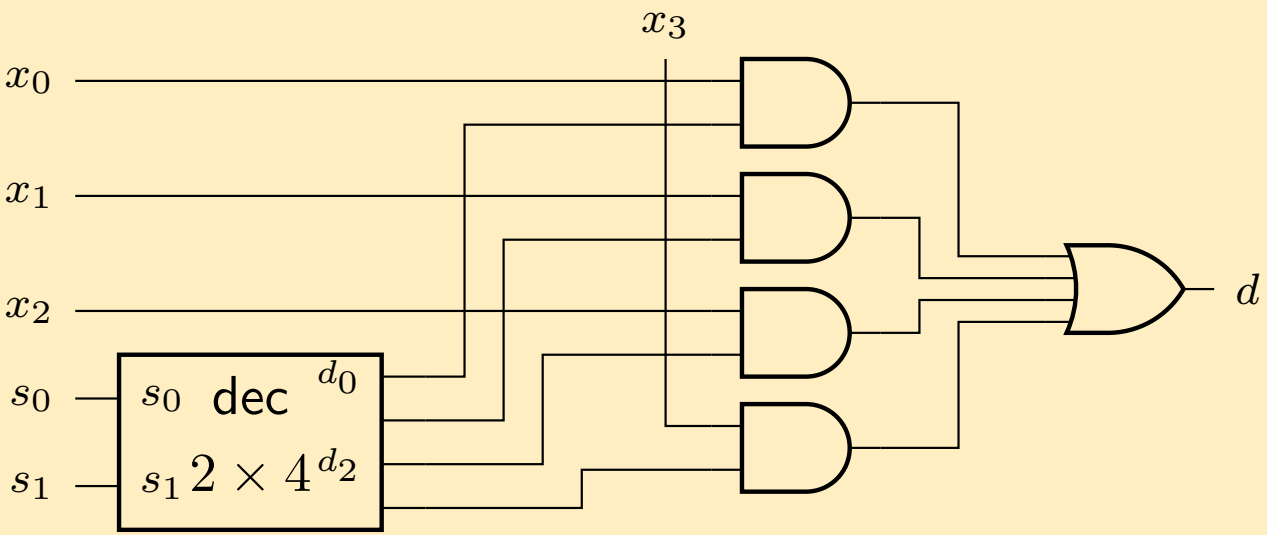
$d = x_s$, where $s = \sum_{i=0}^{n-1} s_i \times 2^i$.

Multiplexer $2 \to 1/4 \to 1$ (circuit)

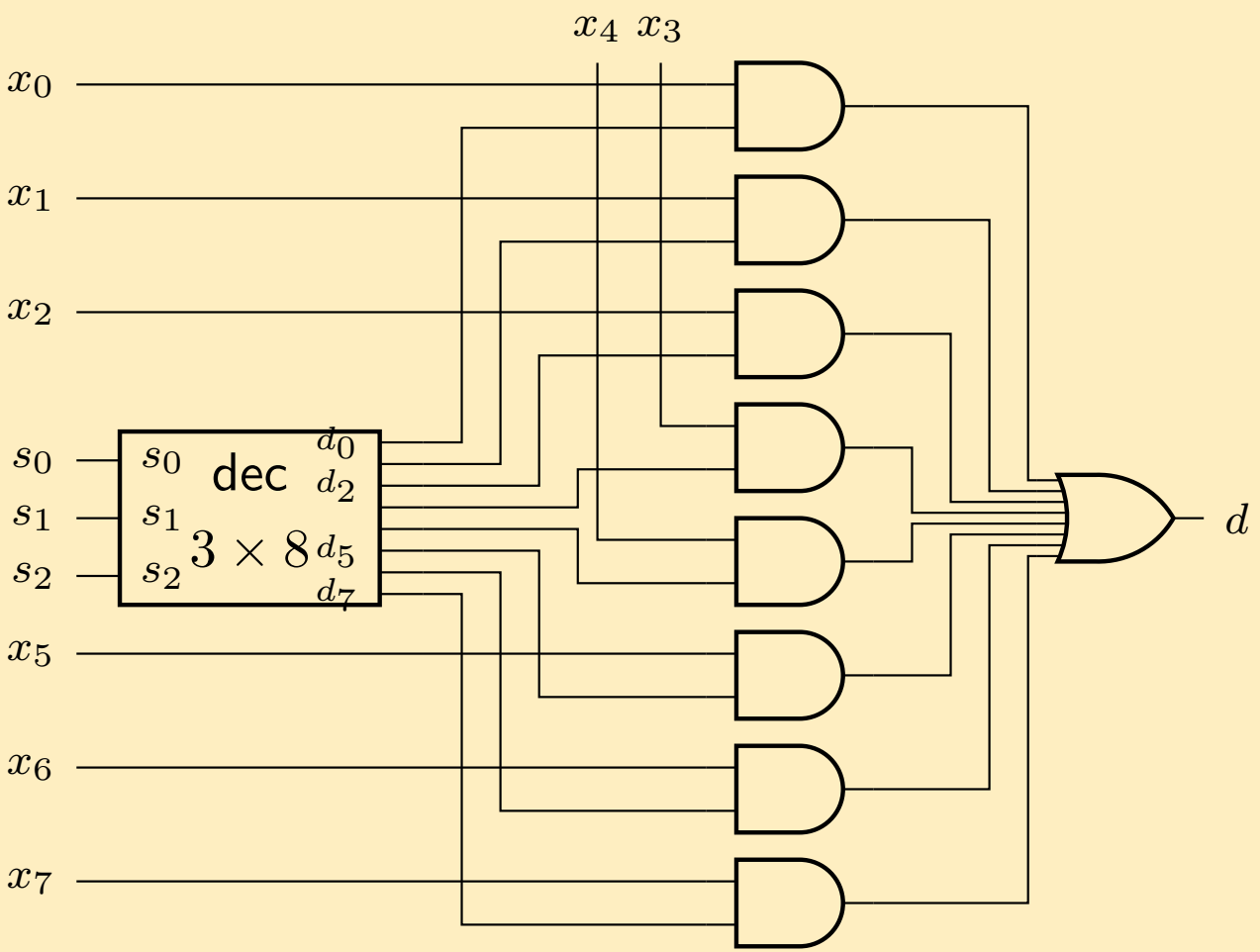
Mux $2 \to 1$



Mux $4 \to 1$



Multiplexer $8 \to 1$ (circuit)



Multiplexer $2^n \rightarrow 1$ (verilog) I

Listing 9: mux.v

```
`timescale 1ns/1ns

module mux #(parameter SEL=1) (
  output out,
  input [2**SEL-1:0] in,
  input [SEL-1:0] s
);
  wire [2**SEL-1:0] T;
  decoder #(SEL) dec(T, s);

  wire [2**SEL-1:0] inMasked;

  for (genvar i = 0; i<2**SEL; i++)
```

Multiplexer $2^n \rightarrow 1$ (verilog) II

```
    andGate a(inMasked[i], in[i], T[i]);

  orMulti #(2**SEL) o(out,inMasked);
endmodule
```

Mux (testbench) I

Listing 10: mux_tb.v

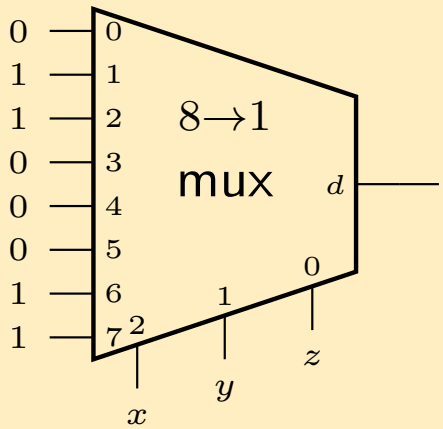
```
`timescale 1ns/1ns

module mux_tb;
  parameter SEL = 3;

  logic [SEL-1:0] sel;
  logic [2**SEL-1:0] x;

  wire T;

  mux #(SEL) m(T, x, sel);

  initial begin
```

Mux (testbench) II

```
    $dumpfile("mux.vcd");
    $dumpvars;

    sel = $urandom_range(0,2**SEL-1);

    repeat (100) begin
      x = $urandom_range(0,2**(2**SEL)-1);
      if ($urandom_range(0,3) == 0)
        sel = $urandom_range(0,2**SEL-1);
      #40ns;
      $display("%d %X %d", sel, x, T);
    end
    $finish();
  end
endmodule
```

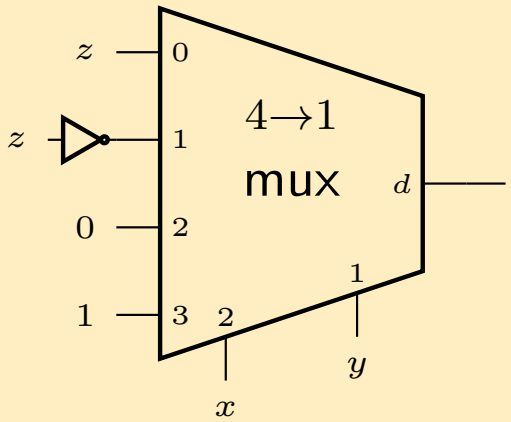
Mux (testbench) III

$f(x, y, z) = \Sigma(1, 2, 6, 7)$ with a $8 \rightarrow 1$ Mux



The connection order of x, y, z is **important**

$f(x, y, z) = \Sigma(1, 2, 6, 7)$ with a $4 \rightarrow 1$ Mux



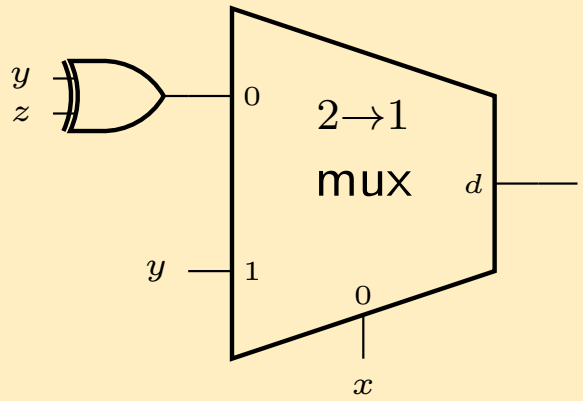
$x = 0, y = 0$	
z	f
0	0
1	1

$x = 1, y = 0$	
z	f
0	1
1	0

$x = 1, y = 0$	
z	f
0	0
1	0

$x = 1, y = 1$	
z	f
0	1
1	1

$f(x, y, z) = \Sigma(1, 2, 6, 7)$ with a $2 \to 1$ Mux



$x = 0$		
y	z	f
0	0	0
0	1	1
1	0	1
1	1	0

$x = 1$		
y	z	f
0	0	0
0	1	0
1	0	1
1	1	1

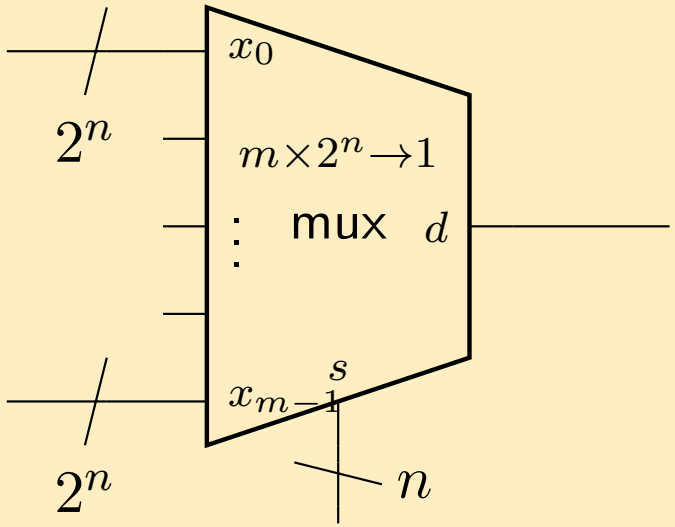
In Class Discussion

1. Introducing the three state buffer gate
2. Implementing mux with the three state buffer

Realizing the Multi-Multiplexer

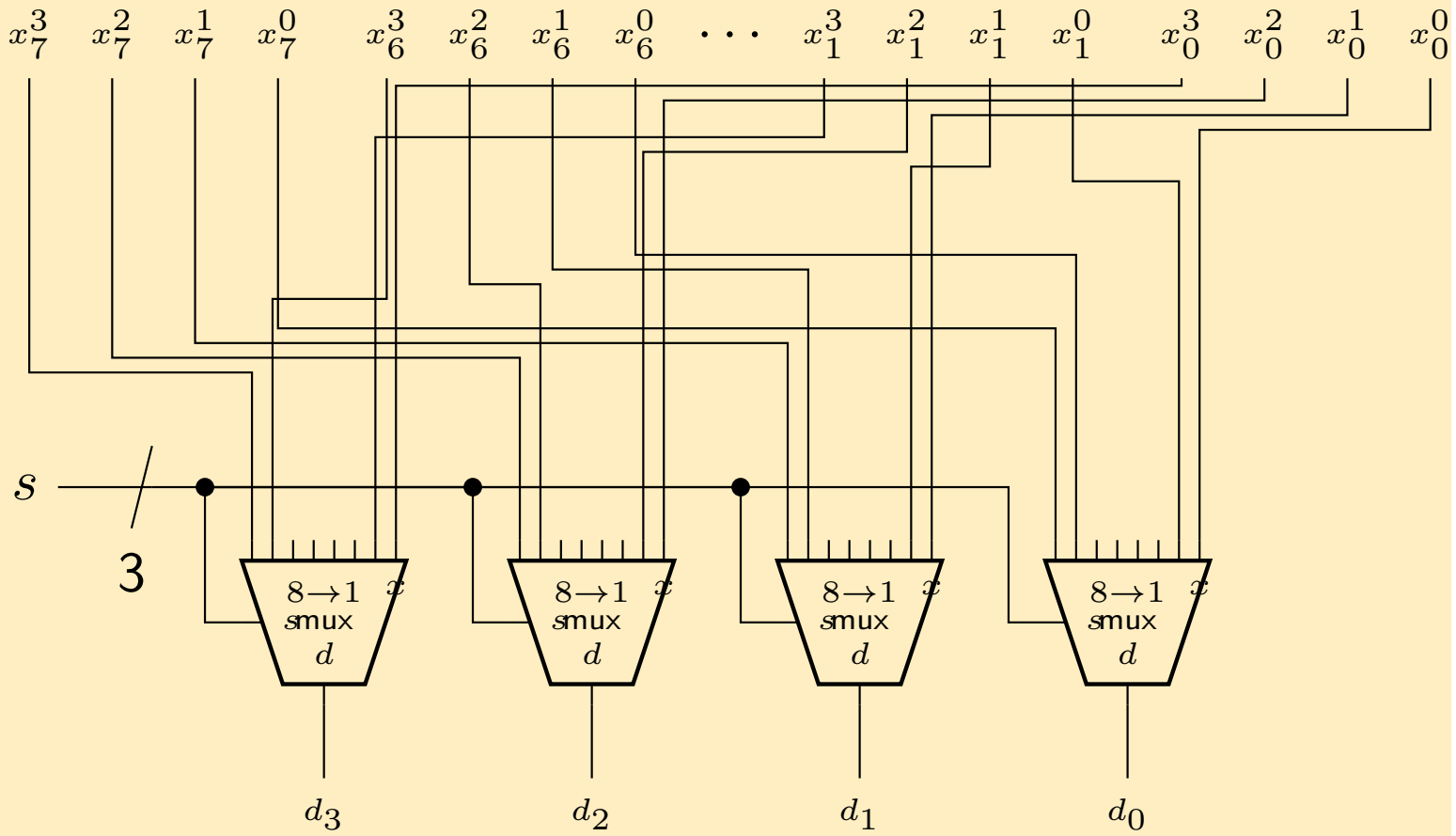
1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

Multi Multiplexer $m \times 2^n \to 1$



$d = x_s$, where $s = \sum_{j=0}^{n-1} s_j \times 2^j$.

Multi Multiplexer $4 \times 8 \to 1$



Multi Multiplexer $m \times 2^n \rightarrow 1$ (module) I

Listing 11: muxMulti.v

```
‘timescale 1ns/1ns

module muxMulti #(parameter SEL=1,
                            WIDTH=1) (
  output [WIDTH-1:0] out,
  input [WIDTH-1:0] in [2**SEL-1:0],
  input [SEL-1:0] sel
);
  for (genvar i = 0; i<WIDTH; i++) begin
    wire [2**SEL-1:0] t;

    for (genvar j = 0; j < 2**SEL; j++)
      assign t[j] = in[j][i:i];
```

Multi Multiplexer $m \times 2^n \rightarrow 1$ (module) II

```
    mux #(SEL) m(out[i], t, sel);
  end
endmodule
```

iverilog and/or gtkwave issue

- Two dimensions wires have view issues
- Hence the funny looking `$dumpvars` in the testbnech

Multi Multiplexer $m \times 2^n \rightarrow 1$ (testbench) I

Listing 12: muxMulti_tb.v

```
‘timescale 1ns/1ns

module muxMulti_tb;
  parameter SEL = 3;
  parameter WIDTH = 4;

  logic [SEL-1:0] sel;
  logic [WIDTH-1:0] t [2**SEL-1:0];
  wire [WIDTH-1:0] x;

  muxMulti #(SEL,WIDTH) m(x, t, sel);

  initial begin
```

Multi Multiplexer $m \times 2^n \rightarrow 1$ (testbench) II


```
    $dumpfile("muxMulti.vcd");
    $dumpvars;
    for (integer i = 0; i < 2**SEL; i++)
      $dumpvars(0, t[i]);

    sel = $urandom_range(0, 2**SEL-1);
    repeat (100) begin
      if ($urandom_range(0,3) == 0)
        sel = $urandom_range(0,2**SEL-1);
      for (integer i = 0; i < 2**SEL; i++)
        t[i] = $urandom_range(0, 2**WIDTH-1);
      #100ns;
    end
    $finish();
  end
```

Multi Multiplexer $m \times 2^n \rightarrow 1$ (testbench) III

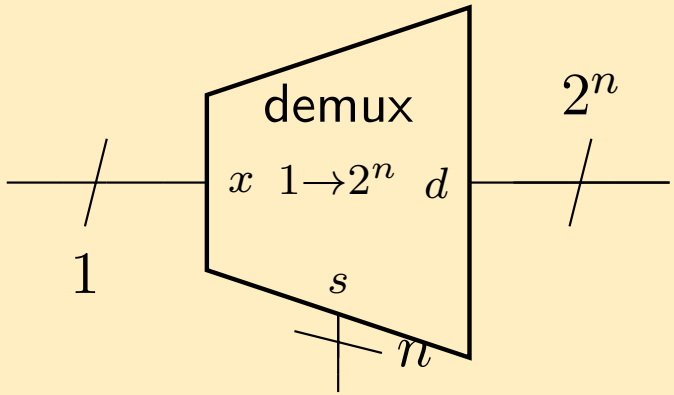
```
endmodule
```

(Carmi) Lecture 13 reached here

Realizing the De-multiplexer

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

Demultiplexer $1 \rightarrow 2^n$ (block diagram)



$$d_i = \begin{cases} x & i = s, \\ 0 & \text{otherwise}, \end{cases}$$

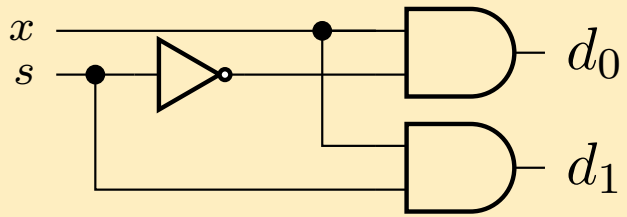
where $s = \sum_{j=0}^{n-1} s_j \times 2^j$.

Popular variation

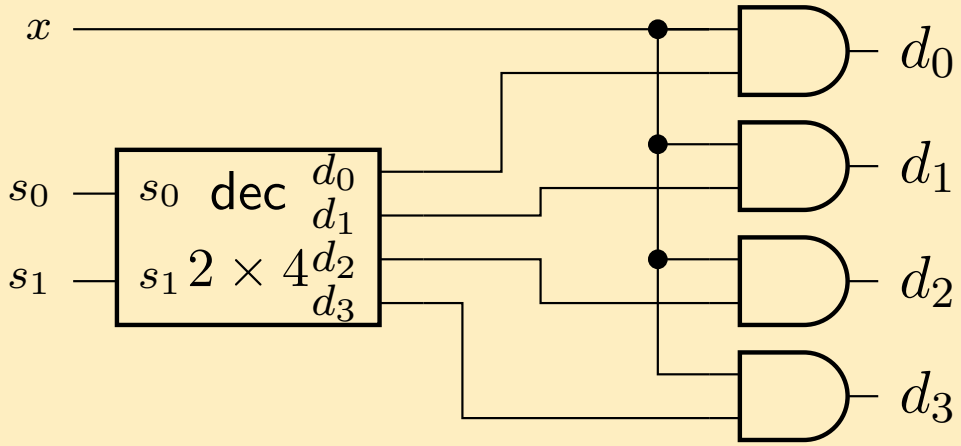
$$d_i = \begin{cases} x & i = s, \\ Z & \text{otherwise}. \end{cases}$$

Demultiplexer $1 \to 2/1 \to 4$ (circuit)

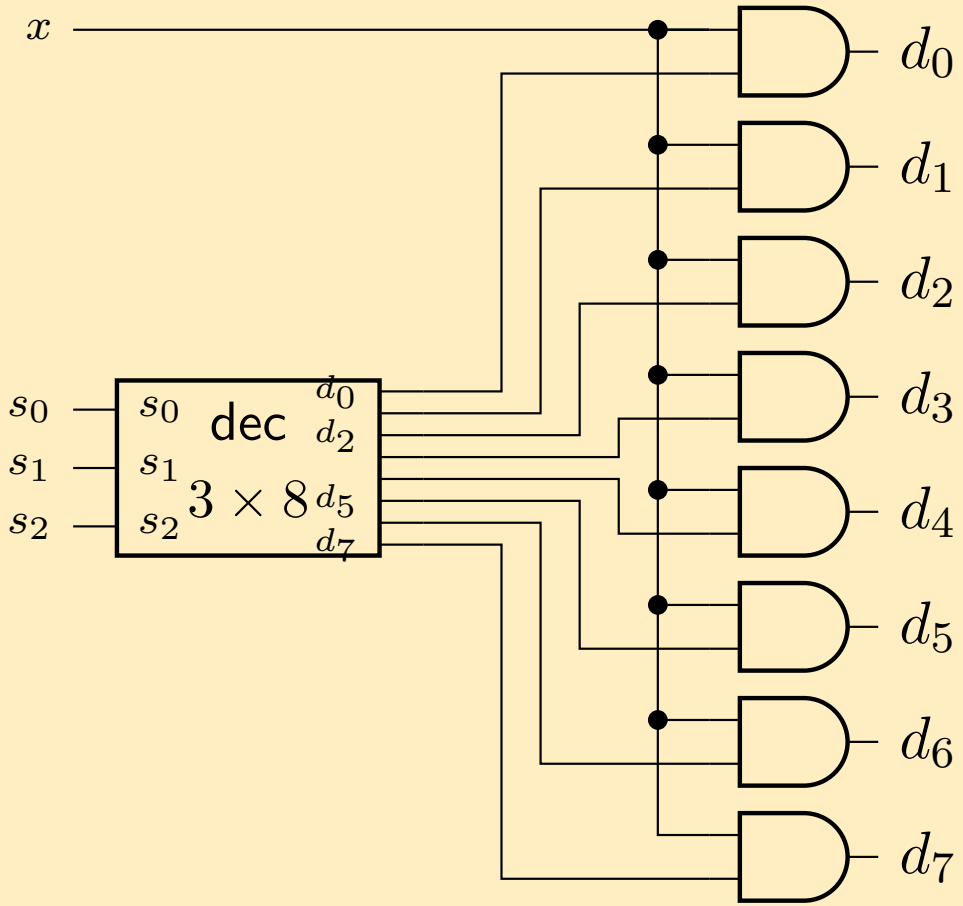
Demux $1 \to 2$



Demux $1 \to 4$



Demultiplexer $1 \rightarrow 8$ (circuit)



Demultiplexer $1 \rightarrow 2^n$ (module) I

Listing 13: demux.v

```
‘timescale 1ns/1ns

module demux #(parameter SEL=1) (
  output [2**SEL-1:0] out,
  input  in,
  input  [SEL-1:0] sel
);
  wire [2**SEL-1:0] T;
  decoder #(SEL) d(T,sel);

  for (genvar i = 0; i < 2**SEL; i++)
    andGate a(out[i:i], in, T[i]);
```

Demultiplexer $1 \rightarrow 2^n$ (module) II

```
endmodule
```

Demultiplexer $1 \rightarrow 2^n$ (testbench) I

Listing 14: demux_tb.v

```
`timescale 1ns/1ns

module demux_tb;
  parameter SEL = 3;

  logic [SEL-1:0] sel;
  logic in;
  wire [2**SEL-1:0] out;

  demux  #(SEL) m(out, in, sel);

  initial begin
    $dumpfile("demux.vcd");
```

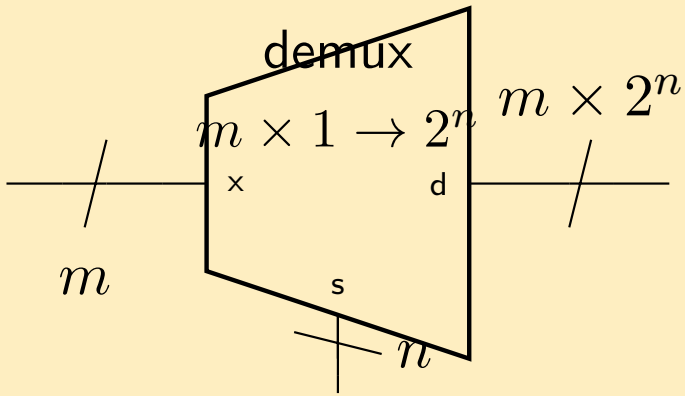
Demultiplexer $1 \rightarrow 2^n$ (testbench) II


```
    $dumpvars;

    sel = $urandom_range(0,2**SEL-1);

    in = 1;
    repeat (100) begin
      if ($urandom_range(0,3) == 0)
        sel = $urandom_range(0,2**SEL-1);
      #300ns;
    end
    $finish();
  end
endmodule
```

Realizing the De-multi-multiplexer

1. Block diagram
2. Logic circuit
3. Verilog module
4. Verilog testbench

De-multimultiplexer $m \times 1 \to 2^n$

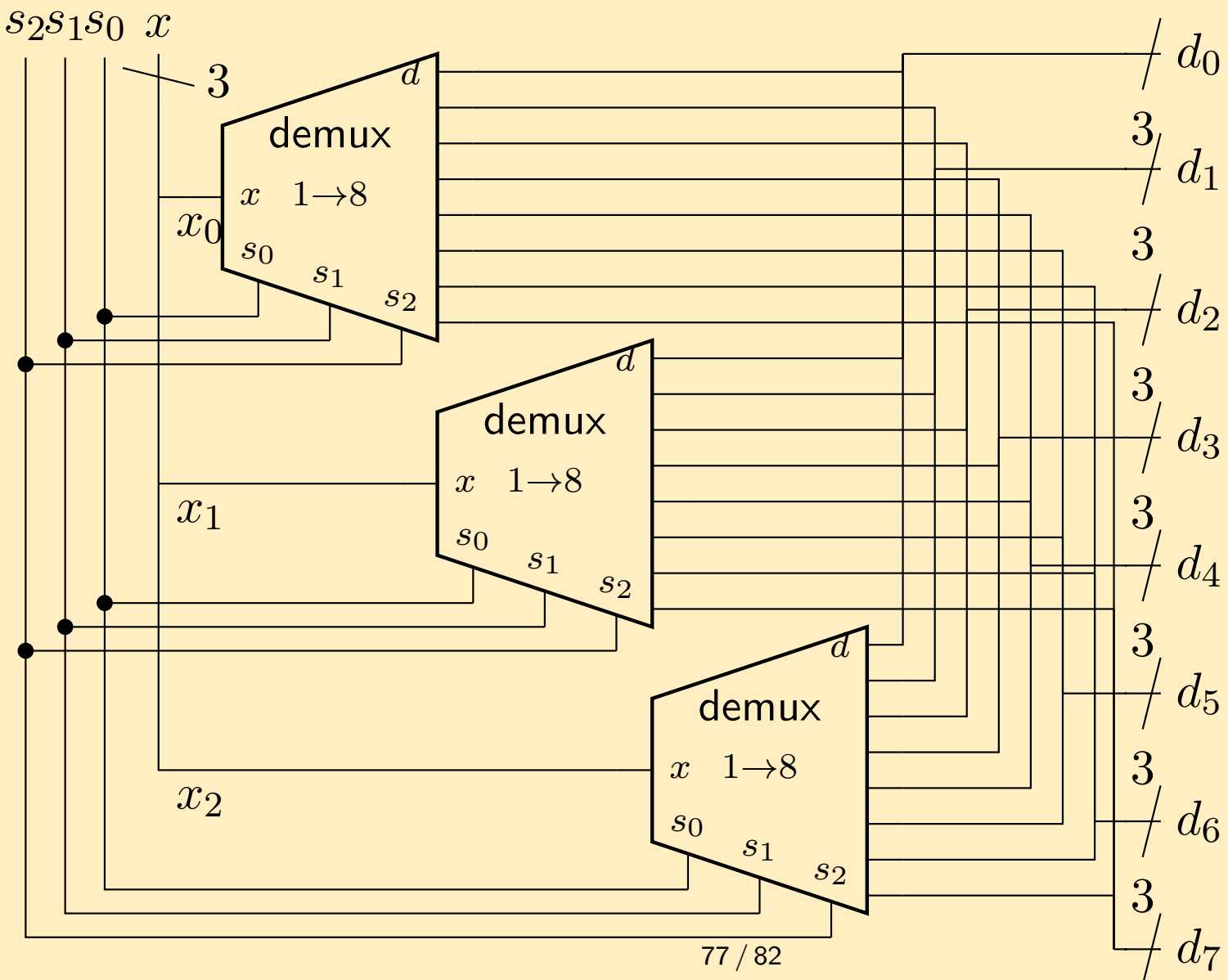


For each $i < 2^n$,

$$d_i = \begin{cases} x & i = s, \\ 0 & \text{otherwise}, \end{cases}$$

where $s = \sum_{j=0}^{n-1} s_j \times 2^j$.

Demux $3 \times 1 \to 8$



De-multimultiplexer $m \times 1 \rightarrow 2^n$ (verilog) I

Listing 15: demuxMulti.v

```
‘timescale 1ns/1ns

module demuxMulti #(parameter SEL=1,
                                WIDTH=1) (
  output [WIDTH-1:0] out [2**SEL-1:0],
  input  [WIDTH-1:0] in,
  input  [SEL-1:0] sel
);
  for (genvar i = 0; i<WIDTH; i++) begin
    wire [2**SEL-1 : 0] t;

    for (genvar j=0; j<2**SEL; j++) begin
      demux #(SEL) m(t, in[i], sel);
```

De-multimultiplexer $m \times 1 \rightarrow 2^n$ (verilog) II

```
      assign out[j][i:i] = t[j:j];
    end
  end
endmodule
```

De-multimultiplexer $m \times 1 \rightarrow 2^n$ (testbench) I

Listing 16: demuxMulti_tb.v

```
‘timescale 1ns/1ns

module demuxMulti_tb;
  parameter SEL = 3;
  parameter WIDTH = 4;

  reg [SEL-1:0] sel;
  reg [WIDTH-1:0] in;
  wire [WIDTH-1:0] out [2**SEL-1:0];

  demuxMulti #(SEL,WIDTH) m(out, in, sel);

  initial begin
```

De-multimultiplexer $m \times 1 \rightarrow 2^n$ (testbench) II

```
    $dumpfile("demuxMulti.vcd");
    $dumpvars;
    for (integer i=0; i < 2**SEL; i++)
      $dumpvars(0, out[i]);

    sel = $urandom_range(0,2**SEL-1);

    repeat (100) begin
      in = $urandom_range(0,2**WIDTH-1);
      if ($urandom_range(0,3) == 0)
        sel = $urandom_range(0,2**SEL-1);
      #100ns;
    end
    $finish();
  end
```

De-multimultiplexer $m \times 1 \rightarrow 2^n$ (testbench) III

```
endmodule
```