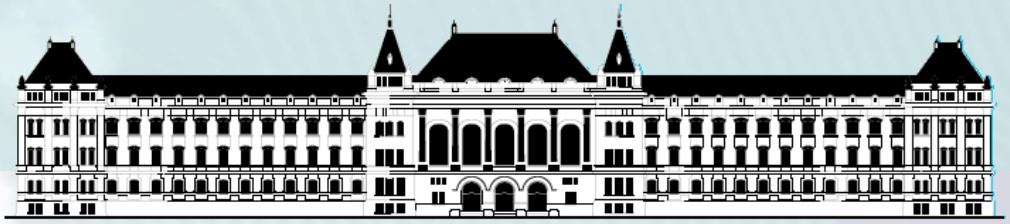


BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Digitaltechnik(VIMIAA01)
Labor 3

Renczes Balázs
Csuka Barna

BME MIT

Aufgabe Lab3_1: DEC/DMP, ENC, PRI, MPX

Während der Aufgabe geben wir einen Übersicht über die Realisationen der standardmäßigen kombinatorischen funtionalen Einheiten:

- ISE starten, Projekt erzeugen
- Quellenfile-Musterrahmen Lab3_1.v spezifizieren
  - ***sw[7:0]***, ***bt[3:0]***, ***ld[7:0]*** (***bt*** brauchen wir später…)
- File LOGSYS_SP3E.UCF zu den gewünschten Interfaces hinzugeben und adaptieren (Interfaces ***sw***, ***bt*** und ***ld***)
- Aufgaben Lab3_1x mit funktionalen Kodteilen spezifizieren
- Überprüfung des funktionalen Kodes mit Simulation
- Generieren des Konfigurationsfiles, Unterladen, Testen des Funktion auf der FPGA-Karte

Aufgabe Lab3_1a

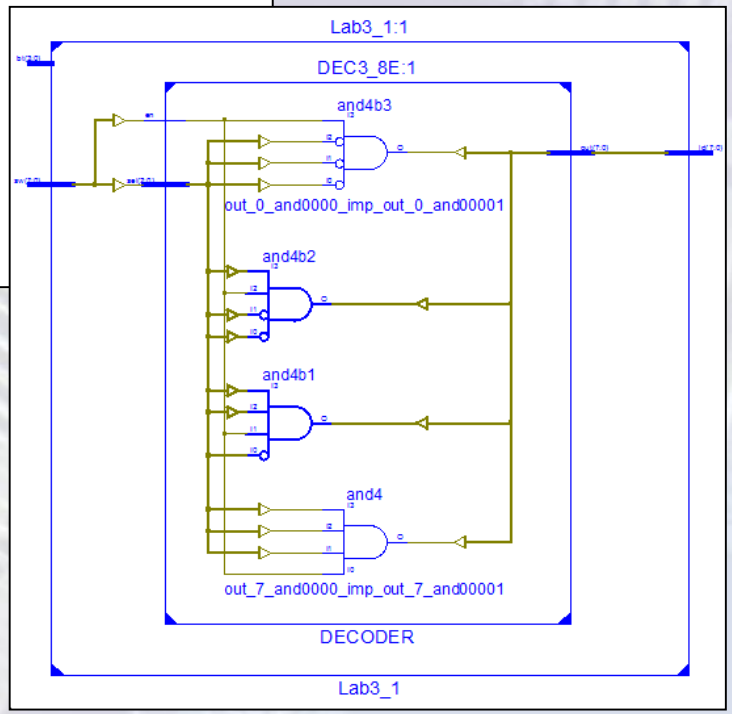
Entwurf vom Modul Dekoder/Demultiplexer

- Binärer Eingang→ 1-aus-N Ausgang (n → $N=2^n$)
- Ganzen Mintermvorrat mit Freigabensignal generieren
- Mögliche Spezifikationsstile
  - Einziges logisches Aufschreiben der Minterme, logische Gleichungen
  - Dasselbe mit arithmetischem Ausdruck, Operator ==
  - Spezifikation mit logisher Shiftoperation {} und Op. <<
  - Verhaltensbeschreibung: always @(*) Blockstruktur
  - Hierarchischer Strukturaufbau aus kleieneren Modulen
    - Bei HDL-Beschreibung hat es auf dieser Ebene keinen Vorteil

Aufgabe Lab3_1a: Kodierungsstile (1)

DEC3_8E Spezifikation: mit logischen Ausdrücken, 8 Minterme

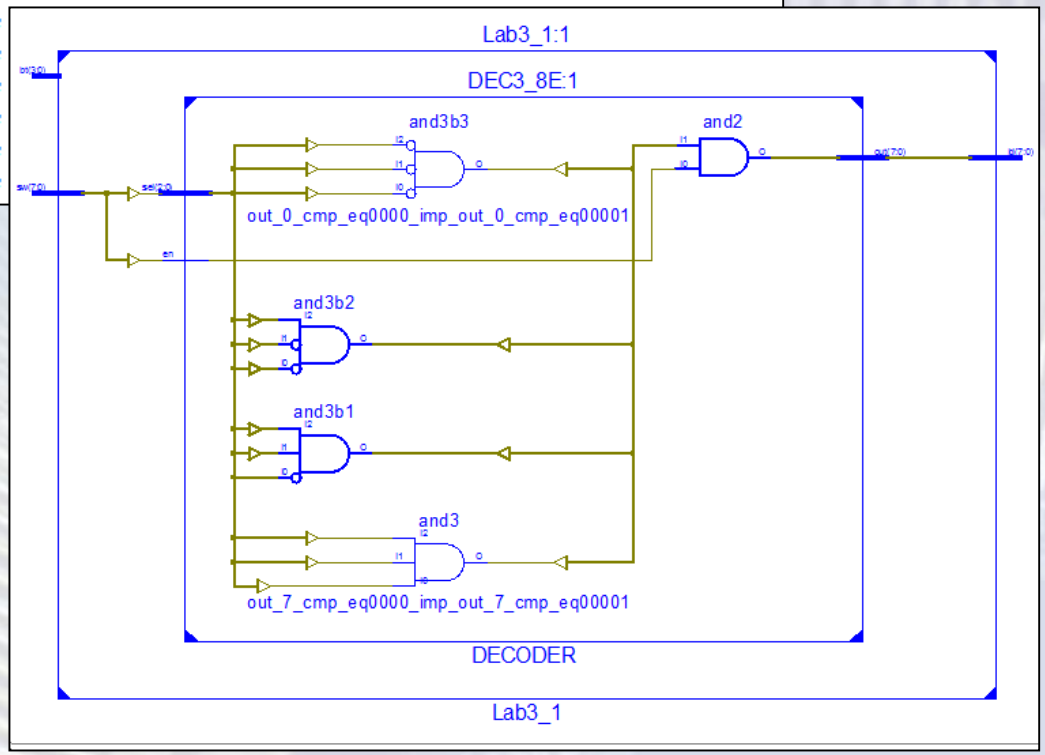
```
////////////////////////////////////////////////////////////////////////////////
// Specifikáció a dekódolás alapján, a kimeneteket egyedi jeleknek tekintve
// n kiválasztó jel, 2^n logikai kifejezés a kimenetekre
////////////////////////////////////////////////////////////////////////////////

assign out[0] = en & ~sel[2] & ~sel[1] & ~sel[0];    // sel = 0
assign out[1] = en & ~sel[2] & ~sel[1] &  sel[0];    // sel = 1
assign out[2] = en & ~sel[2] &  sel[1] & ~sel[0];    // sel = 2
assign out[3] = en & ~sel[2] &  sel[1] &  sel[0];    // sel = 3
assign out[4] = en &  sel[2] & ~sel[1] & ~sel[0];    // sel = 4
assign out[5] = en &  sel[2] & ~sel[1] &  sel[0];    // sel = 5
assign out[6] = en &  sel[2] &  sel[1] & ~sel[0];    // sel = 6
assign out[7] = en &  sel[2] &  sel[1] &  sel[0];    // sel = 7
```



Aufgabe Lab3_1a: Kodierungsstile (2)

DEC3_8E Spezifikation: mit arithmetischen Operationen (==)

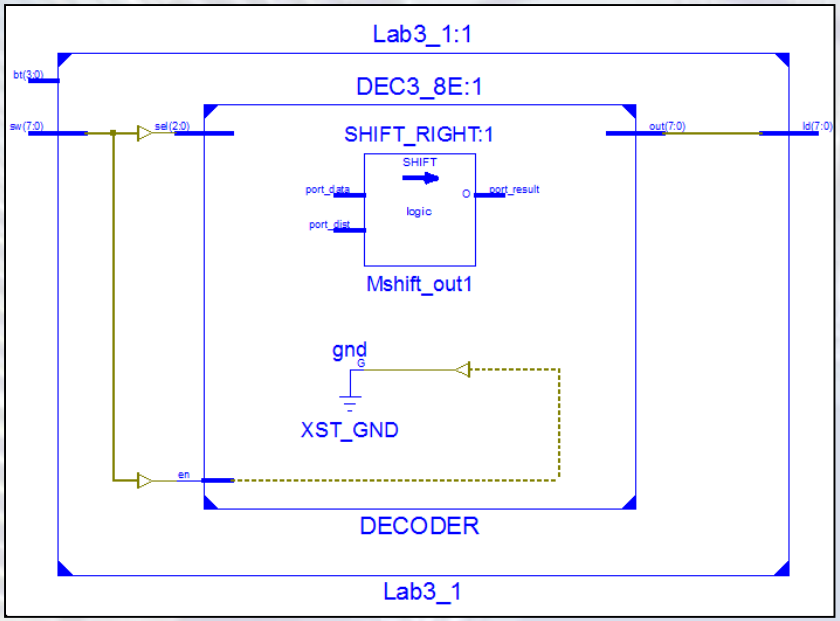
```
////////////////////////////////////////////////////////////////////////////////
// Olvashatóbb, öndokumentáló formában
////////////////////////////////////////////////////////////////////////////////

assign out[0] = en & (sel == 3'd0);   // Lehetne persze 3'b000 is
assign out[1] = en & (sel == 3'd1);
assign out[2] = en & (sel == 3'd2);
assign out[3] = en & (sel == 3'd3);
assign out[4] = en & (sel == 3'd4);
assign out[5] = en & (sel == 3'd5);
assign out[6] = en & (sel == 3'd6);
assign out[7] = en & (sel == 3'd7);
```



Aufgabe Lab3_1a: Kodierungsstile (3)

DEC3_8E Spezifikation: mit Shiftoperator (<<)

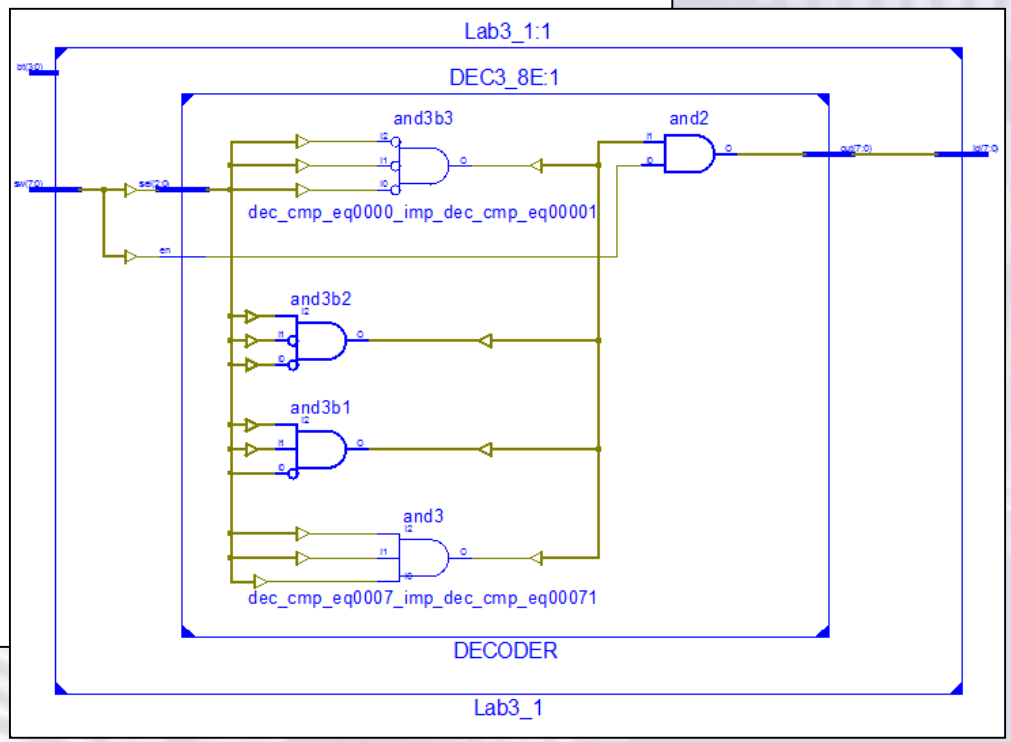
```
////////////////////////////////////////////////////////////////////////////////
// Specifikáció a logikai shiftelés művelet alapján
// A kimeneti 1-az-N-ből kódú N bites bitvektort 0...2^n lépéssel shifteljük.
// A 8 bites kimeneti bitvektort egy 0 értékű 7 bites vektorból és az EN 1 bites
// bemenetből képezzük. Ha EN 0, akkor nem lesz aktív kimenet sehol. Ha EN 1,
// akkor az aktív egyes abba apozícióba lép, amit a SEL kód meghatároz
////////////////////////////////////////////////////////////////////////////////

assign out = {7'b0000000,en} << sel;      // Ez egy dekóder funkció
```



Aufgabe Lab3_1a: Kodierungsstile (4)

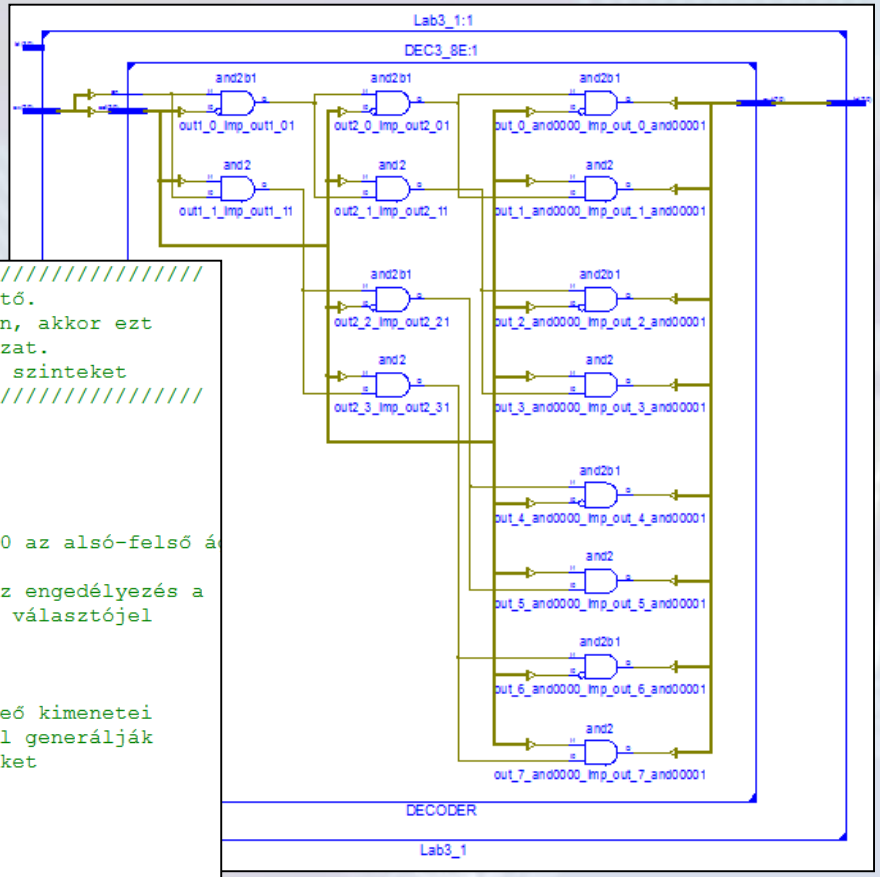
DEC3_8E Spezifikation mit always @(*)-Block

```
////////////////////////////////////////////////////////////////////////////////
// A viselkedési leírást biztosító alwasy @ (*) konkurens folyamattal
// Azt a működést fogalmazzuk meg, ami a funkcióhoz szeretnénk rendelni,
// azaz ha aktív az engedélyezés, akkor a sel értéke alapjéán a kimenet legyen...
// Ez a viselkedési leírás előnye, hogy jól olvasható, értelmezhető
////////////////////////////////////////////////////////////////////////////////

reg [7:0] dec;
always @ (*)
   begin
      if (en)
         case (sel)
         3'b000: dec = 8'b00000001;
         3'b001: dec = 8'b00000010;
         3'b010: dec = 8'b00000100;
         3'b011: dec = 8'b00001000;
         3'b100: dec = 8'b00010000;
         3'b101: dec = 8'b00100000;
         3'b110: dec = 8'b01000000;
         3'b111: dec = 8'b10000000;
         endcase
      else
                  dec = 8'b00000000;

   end

assign out = dec;
```



Aufgabe Lab3_1a: Kodierungsstile (5)

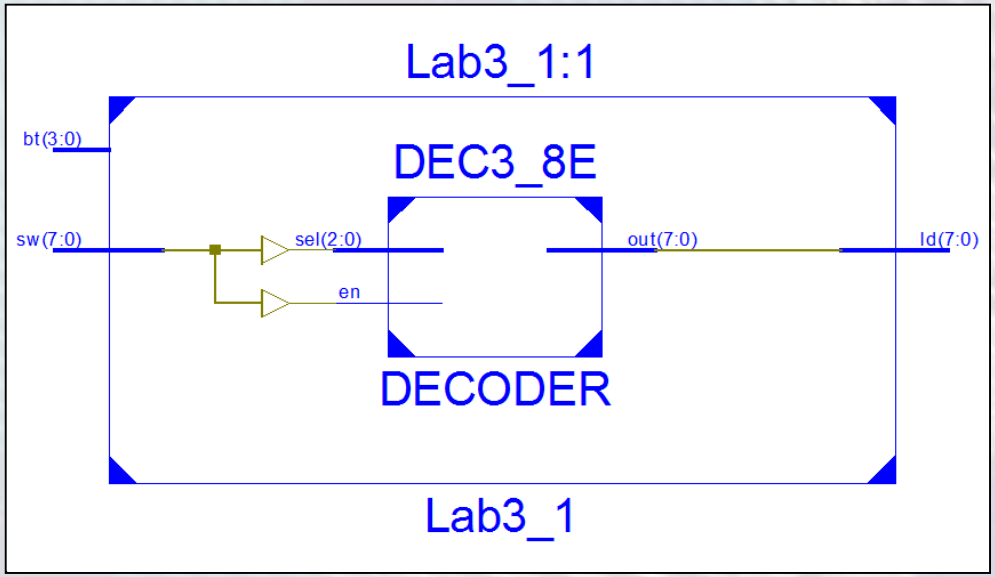
DEC3_8E Spezifikation: hierarchischer Aufbau aus 1:2 großen elementaren Einheiten

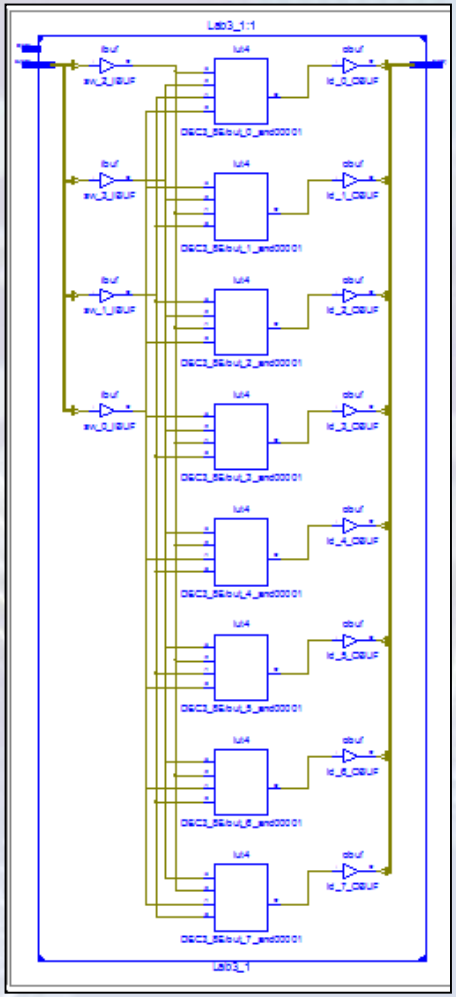



```
////////////////////////////////////////////////////////////////////////////////
// A dekóder modul, mint szabványos elem hierarchikusan is bővíthető.
// Ha rendelkezünk egyetlen alapmodullal, pl. 1:2 vagy 2:4 méretben, akkor ezt
// használva a bináris fa szerkezetű struktúrában bővíthető a hálózat.
// Nem készítettünk külön modult, csak az egyenletekkel jelezzük a szinteket
////////////////////////////////////////////////////////////////////////////////

wire out1_0, out1_1;                    // Első szint kimenetek
wire out2_0, out2_1, out2_2, out2_3;    // Második szint kimenetek

assign out1_0 = en & ~sel[2];           // A dekódoló fa gyökere
assign out1_1 = en &  sel[2];           // Az EN globális, a _1 és _0 az alsó-felső á

assign out2_0 = out1_0 & ~sel[1];       // Az out1_0 és az out1_1 az engedélyezés a
assign out2_1 = out1_0 &  sel[1];       // 2. szinten. Itt sel[1] a választójel
assign out2_2 = out1_1 & ~sel[1];
assign out2_3 = out1_1 &  sel[1];

assign out[0] = out2_0 & ~sel[0];       // A második szint megfeleleő kimenetei
assign out[1] = out2_0 &  sel[0];       // a sel[0] kiválsztó bittel generálják
assign out[2] = out2_1 & ~sel[0];       // a végleges kimeneti jeleket
assign out[3] = out2_1 &  sel[0];
assign out[4] = out2_2 & ~sel[0];
assign out[5] = out2_2 &  sel[0];
assign out[6] = out2_3 & ~sel[0];
assign out[7] = out2_3 &  sel[0];
```

Aufgabe Lab3_1a: Erfahrungen

DECODER Ergebnis: derselbe Aufbau in allen Fällen, unabhängig vom Spezifikationsstil: 8 Stücke





Aufgabe Lab3_1a

Wählen Sie einen möglichen Spezifikationsstil und anhand dessen schreiben Sie das Modul DEC3_8E.v!

- Das Modul dekodiert aus 3 auf 8 Bits.
- Eingänge: ***sel[2:0]***, ***en***; Ausgang: ***out[7:0]***

Aufgabe Lab3_1a

Einbau des Moduls DEC3_8E.v namens DECODER

```
////////////////////////////////////////////////////////////////////////////////
// Dekóder, demultiplexer: Kombinációs funkció
// Interfészek: Bemenetek: n cím/kiválasztás, +1..2 eng/adat
//              Kimenetek: N=2^n adatkimenet
// Feladata/funkciója kétféleképpen értelmezhető:
//    1. Dekóder/Kódkonverter: Bemeneti n bites bináris kódot 2^n bites 1-az-N-ből
//       kóddá konvertálja.
//    2. Demultiplexer: Az adatbemenetére adott adatbitet az n bites
//       bináris értékkel meghatározott N=2^n db kimenetei közül az egyikre
//       kapcsolja.
//    Az engedélyező bemenet és adatbemenet kölcsönösen mgfeleltethető egymásnak.
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Vezérlőjelek definiálása és a DECODER modul példányosítása
// A példa az n=3 esetet (3->8 dekóder) használja, az I/O arányok érzékeltetésére
////////////////////////////////////////////////////////////////////////////////

wire [2:0] sel;                 // Dekóder kiválasztó bemenet
wire en;                        // Dekóder engedélyező bemenet, pl. kaszkádosításra
wire [7:0] dout;                // Dekóder kimenet

assign sel = sw[2:0];           // A külső jelek használatának előírása
assign en  = sw[3];

DEC3_8E DECODER (.sel(sel), .en(en), .out(dout));  //Egy dekóder egység beépítése

assign ld[7:0] = dout;          // A kimenetek kijelzése a LED-eken
```

Aufgabe Lab3_1b

Entwurf eines Prioritätsenkoders

- Der normale Enkoder ist nicht genügend allgemein
- Beim Prioritätsenkoder ist der Eingang beliebig. Es komprimiert auch aus N Bits auf $\log_2(N)$ Bits.
- Lautere 0 Kode ist möglich am Eingang
  - Signal INVAL ist notwendig

Aufgabe Lab3_1b

Entwurf von Funktion PRIE8_3E: bitweise Spezifikation des binären Ausgangs pout[2:0], die Maskierungswirkung der oberen Bits ist zu besorgen!

```
////////////////////////////////////////////////////////////////////////////////
// Specifikáció a funkció alapján
// A kimeneten a legngyaobb helyiértékű aktív bit sorszáma jelenik meg, binárisan,
// ha engedélyezett
// A 0. sorszámú bemeneti bit érdekes. Ezt a valid kimenet érvényesíti!
////////////////////////////////////////////////////////////////////////////////


assign pout[2] = en & ( dinp[7] | dinp[6] | dinp[5] | dinp[4]);

assign pout[1] = en & ( dinp[7]
                      | dinp[6]
                      | ~(dinp[5] | dinp[4]) & (dinp[3] | dinp[2]));

assign pout[0] = en & ( dinp[7]
                      | ~dinp[6] & dinp[5]
                      | ~(dinp[6] | dinp[4]) & dinp[3]
                      | ~(dinp[6] | dinp[4] | dinp[2]) & dinp[1]);

assign inval = ~en | ~|dinp;                // A kimenet nem érvényes, ha EN = 0,
                                            //  vagy EN = 1, de a bemenet csupa 0
```

Aufgabe Lab3_1b, Kodierungsstile(2)

Entwurf von Funktion PRIE8_3E: Spezifikation des binären Ausgangs pout[2:0] im Block always @ (*), mit Verhaltensbeschreibung

- x in Block casex ist don't care

```
////////////////////////////////////////////////////////////////////////////////
// Specifikáció az igazságtáblázat alapján, kihasználva a don't care jelölést
// Ebben az esetben a casex kulcsszó használatával tömör, kifejező formában
// írhatjuk fel a bemenet - kimenet közötti kapcsolatot
////////////////////////////////////////////////////////////////////////////////

reg [2:0] rout;

always @ (*)
   begin
      if (en)
         casex (dinp)
            8'b00000001 : rout = 3'b000;
            8'b0000001x : rout = 3'b001;
            8'b000001xx : rout = 3'b010;
            8'b00001xxx : rout = 3'b011;
            8'b0001xxxx : rout = 3'b100;
            8'b001xxxxx : rout = 3'b101;
            8'b01xxxxxx : rout = 3'b110;
            8'b1xxxxxxx : rout = 3'b111;
            default:      rout = 3'b000;
            endcase
      else
            rout = 3'b000;

   end

assign pout = rout;

assign inval = ~en | ~|dinp;
```

Aufgabe Lab3_1b

Einbau des Moduls PRIE8_3E.v namens PRIENC

- Freigabe durch BT[0]

```
////////////////////////////////////////////////////////////////////////////////
// Prioritás enkóder: A normál enkóder funkció jobban használható verziója
// Kiküszöböli a nem értelmezhető bemeneti kódók nehéz kezelését, egyúttal
// nevének megfeleleően sorrendiséget, fontosságot is rendel a bemeneti bitekhez.
// Az N=2**n bemeneti vonalai közül tetszőleges bitkombináció esetén a kimeneten
// azt a log2(N) bináris kódot adja, amelyik a legnagyobb aktív helyiértékű bit.
// Lényegében ez is egy adatkompressziót valósít meg, és egyúttal prioritást is ad.
// Az I/O arányok azonosak a normál enkóderrel.
// Kisebb egyértelműségi probléma itt is felléphet a 000 kimenet esetén!
////////////////////////////////////////////////////////////////////////////////

wire [7:0] dinp;
wire en;
wire [2:0] pout;                    // Prioritás enkóder bináris kimenet
wire inval;                         // Érvénytelen kimenet

assign dinp = sw[7:0];
assign en   = bt[0];

PRIE8_3E PRIENC (.dinp(dinp), .en(en), .pout(pout), .inval(inval));

assign ld[2:0] = pout;              // Binárisan kódolt kimenet
assign ld[6:3] = 4'b0;              // Nem használt kimeneteket 0-ra kötjük
assign ld[7]   = inval;             // Jelzi ha a kimenet nem érvényes
```

Aufgabe Lab1_c

Entwurf eines Multiplexers

- **Mit logischen Ausdrücken**

```
//////////////////////////////////////////////////////////////////////////////////
// A klasszikus, kapuszintű leírás
//////////////////////////////////////////////////////////////////////////////////

assign out = en & ((dinp[0] & ~sel[2] & ~sel[1] & ~sel[0])
                  |(dinp[1] & ~sel[2] & ~sel[1] &  sel[0])
                  |(dinp[2] & ~sel[2] &  sel[1] & ~sel[0])
                  |(dinp[3] & ~sel[2] &  sel[1] &  sel[0])
                  |(dinp[4] &  sel[2] & ~sel[1] & ~sel[0])
                  |(dinp[5] &  sel[2] & ~sel[1] &  sel[0])
                  |(dinp[6] &  sel[2] &  sel[1] & ~sel[0])
                  |(dinp[7] &  sel[2] &  sel[1] &  sel[0]));
```

- **Arithmetische Beschreibung**

```
//////////////////////////////////////////////////////////////////////////////////
// Ugyanez olvashatóbb formában
//////////////////////////////////////////////////////////////////////////////////

assign out = en & ( (dinp[0] & (sel == 3'd0))
                  | (dinp[1] & (sel == 3'd1))
                  | (dinp[2] & (sel == 3'd2))
                  | (dinp[3] & (sel == 3'd3))
                  | (dinp[4] & (sel == 3'd4))
                  | (dinp[5] & (sel == 3'd5))
                  | (dinp[6] & (sel == 3'd6))
                  | (dinp[7] & (sel == 3'd7)));
```

Aufgabe Lab1_c, Kodierungsstile(2)

MPX8_1E: Spezifikation im Block always @(*) blokkban, mit Verhaltensbeschreibung

```
////////////////////////////////////////////////////////////////////////////////
// A viselkedési leírást biztosító alwasy @ (*) konkurrens folyamattal
// Azt a működést fogalmazzuk meg, amit a funkcióhoz szeretnénk rendelni
// Ez a forma jól olvasható
////////////////////////////////////////////////////////////////////////////////

reg mpx;

always @ (*)
   begin
      if (en)
         case (sel)
         3'd0:  mpx = dinp[0];
         3'd1:  mpx = dinp[1];
         3'd2:  mpx = dinp[2];
         3'd3:  mpx = dinp[3];
         3'd4:  mpx = dinp[4];
         3'd5:  mpx = dinp[5];
         3'd6:  mpx = dinp[6];
         3'd7:  mpx = dinp[7];
         endcase
      else
                mpx = 1'b0;       // Nics engedélyezve
   end

assign out = mpx;
```

Aufgabe Lab1_c, Kodierungsstile(3)

MPX8_1E: Kaskadierung

Aufbau nach oben: sel[0] → sel[1] → sel[2]

```
////////////////////////////////////////////////////////////////////////////////
// Kaszkád modell, 2:1 elemi multiplexer segítségével
// Specifikáció a feltételes operátorral, 3 szinten, LSB-vel kezdve,
// páros-párstlan választással
// A kiválasztó bitek sorrendje sel[0] - sel[1] - sel[2]
////////////////////////////////////////////////////////////////////////////////

wire out76, out54, out32, out10;

assign out76 = sel[0] ? dinp[7] : dinp[6];   // 2 db 2:1 MUX a kisebb helyiértékű
assign out54 = sel[0] ? dinp[5] : dinp[4];   // sel bittel vezérelve
assign out32 = sel[0] ? dinp[3] : dinp[2];   //(páros/páratlan) választások
assign out10 = sel[0] ? dinp[1] : dinp[0];

wire out7654, out3210;

assign out7654  = sel[1] ? out76 : out54;    // A középső szint
assign out3210  = sel[1] ? out32 : out10;    // első szintű kimenetekből választ

assign out = en & (sel[2] ? out7654 : out3210);
                                             // A teljes kimenet a két félből,
                                             // engedélyezéssel
```

Aufgabe Lab1_c, Kodierungsstile(4)

MPX8_1E

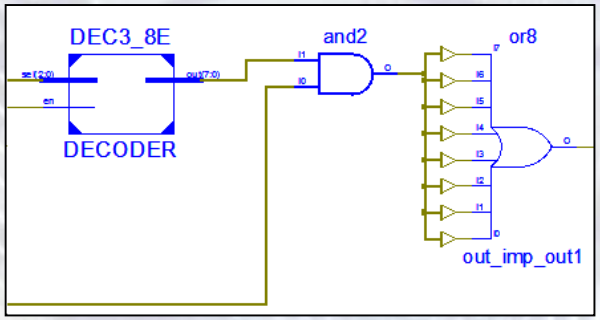
- Aufbau nach unten: sel[2] → sel[1] → sel[0]
- Einfacher, Bitvektoren können benutzt werden

```
////////////////////////////////////////////////////////////////////////////////
// Lehetne felső-alsó 4-4 választással is indulni, a legnagyobb helyiértékű
// választó bit alapján, aztán a maradék 4-ből 2-2 és végül 1 kimenet + EN
// Könnyebben olvasható, de az eredmény ugyanaz: 3 rétegű hierarchikus MPX
// A kiválasztó bitek sorrendje sel[2] - sel[1] - sel[0]
////////////////////////////////////////////////////////////////////////////////

wire [3:0] outx;
wire [1:0] outy;

assign outx = sel[2] ? (dinp[7:4]) : (dinp[3:0]);
assign outy = sel[1] ? (outx[3:2]) : (outx[1:0]);

assign out  = en & (sel[0] ? (outy[1]) : (outy[0]));
```

Aufgabe Lab1_c, Kodierungsstile(5)

MPX8_1E mithilfe von einem Dekoder

```
////////////////////////////////////////////////////////////////////////////////
// Realizációja a DEKÓDER alapján is lehetséges.
// A DEKÓDER aktiválja a sel által kiválasztott kimenetét, ha az engedélyezett, és
// ezt az 1-az-N-ből kódú bitvektort bitenkénti ÉS kapcsolatba hozzuk a bemeneti
// adatvektorral és a kimenetet egy redukciós VAGY függvénnyel generáljuk
////////////////////////////////////////////////////////////////////////////////

wire [7:0] decout;

DEC3_8E DECODER (.sel(sel), .en(en), .out(decout));

assign out = |(decout & dinp);          // Bitvektorok ÉS kapcsolata,
                                        // redukciós VAGY művelettel az eredményen
```



Aufgabe Lab1_c

Einbau des Moduls MPX8_1E.v namens MULTIPX

```
////////////////////////////////////////////////////////////////////////////////
// Multiplexer: Kombinációs funkció
// Interfészek: Bemenetek: n cím/kiválasztás, N=2^n adat, engedélyezés
//              Kimenetek: 1 adatkimenet
// Feladata/funkciója:
//    N db bemeneti adat küzül 1 kiválasztása és kimenetre kapcsolása
//    Ha nincs engedélyezve, a kimenet 0.
////////////////////////////////////////////////////////////////////////////////

wire [2:0] sel;
wire en;
wire [7:0] dinp;
wire out;

assign sel = sw[2:0];
assign en  = sw[3];

assign dinp[7:4] = sw[7:4];
assign dinp[3:0] = bt[3:0];


MPX8_1E MULTIPX(.sel(sel), .en(en), .dinp(dinp), .out(out));

assign ld[0] = out;
assign ld[7:1] = 7'b0;              // Nem használt kimenetek 0-ra kötjük
```

Aufgabe Lab3_2

7-Segment-Anzeige

- Zeitmultiplexe Anzeige auf 2 Digits
- Die Zeitmultiplexierung wird mit dem Signal CLK des LOGSYS Unterladenkabels verwirklicht
- Anzeigebild (2-Digit-Hexawert) wird mit dem Schaltern eingestellt
- Wir entwerfen Umsetzer HEXA→7SEGMENT
- Wir benutzen die Multiplexerfunktion

Lab3_2 feladat: HEX→7SEG

Verilog-Modul:

- Eingang: value[3:0]
- Ausgang: code[7:0]

```
// Die 7-Segment-Segmentbilder lassen sich im
// Kodenvektor  in positiver Logik nach
// Bitreihenfolge 76543210 abgelesen
//        0
//       ---
//   5 |   | 1
//       --- <--6
//   4 |   | 2
//       ---       . 7
//        3
```

```
module HEX7SEG(
    input       [3:0] value,
    output reg [7:0] code
    );

always @ (*)
   case (value)
      4'h0: code = 8'b00111111;
      4'h1: code = 8'b00000110;
      4'h2: code = 8'b01011011;
      4'h3: code = 8'b01001111;
      4'h4: code = 8'b01100110;
      4'h5: code = 8'b01101101;
      4'h6: code = 8'b01111101;
      4'h7: code = 8'b00000111;
      4'h8: code = 8'b01111111;
      4'h9: code = 8'b01101111;
      4'hA: code = 8'b01110111;
      4'hb: code = 8'b01111100;
      4'hC: code = 8'b00111001;
      4'hd: code = 8'b01011110;
      4'hE: code = 8'b01111001;
      4'hF: code = 8'b01110001;
   endcase

endmodule
```

Aufgabe Lab3_2: das ganze Projekt

Im Topmodul:

- **Ziffern definieren:**
  - TENS[3:0]→SW[7:4], ONES[3:0] → SW[3:0]
- **Punktmatrix sperren!**
- **Zeitmultiplexierung einstellen!**

```
//////////////////////////////////////////////////////////////////////////////////
// A megejelenítés vezérlőjelei
// A szegmens jeleket egy 7 szegmenses hexadecimális dekóder szolgáltatja
// A modulnak 4 bites bemenete és (7+1) bites kimenete van, pozitív logikával
// A digitjelek közül a felső kéttő inaktív, az alsó kettőt (tízesek-egyesek) az
// órajel vezérli. Ennek következtében felváltva villannak fel
// Az 5x7-es pontmátrix kijelző oszlopjelei inaktívak
//////////////////////////////////////////////////////////////////////////////////

wire [3:0] tens, ones;
assign tens = sw[7:4];
assign ones = sw[3:0];

assign col_n = 5'b11111;                // Pontmátrix kikapcsolva

assign dig_n = {2'b11, clk, ~clk};      // 4 bites vezérlés, felő kettő kikapcsolva
                                        // Ha clk = 1 akkor az egyes helyiérték,
                                        // ha 0, akkor a tízes helyiérték kapcsol be
```