Chapter 5: Selectors, Shifters, Priority Encoders & Half-Decoders

Computer Structure
&
Intro. to Digital Computers

Dr. Guy Even

Tel-Aviv Univ.

Goals

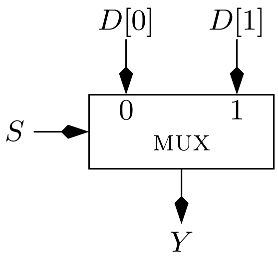
- Selectors:
  - review definition of multiplexer.
  - build $(n:1)$-multiplexers.
- Shifters:
  - Cyclic shifting
  - Barrel shifting
- Priority encoders
  - Unary priority encoder
  - Binary priority encoder

Multiplexer

DEF: A MUX-gate (also known as a $(2:1)$-multiplexer) is a combinational gate that has three inputs $D[0], D[1], S$ and one output Y. The functionality is defined by

$$Y = \begin{cases} D[0] & \text{if } S = 0 \\ D[1] & \text{if } S = 1. \end{cases}$$

Equivalently: $Y = D[S]$



Selectors

DEF: An (n:1)-MUX is a combinational circuit defined as follows:

Input: $D[n-1:0]$ and $S[k-1:0]$ where $k = \lceil \log_2 n \rceil$.

Output: $Y \in \{0, 1\}$.

Functionality:

$$Y = D[\langle \vec{S} \rangle].$$

Example: Let $n = 4$, $D[3:0] = 0101$, and $S[1:0] = 11$. The output Y should be 1.

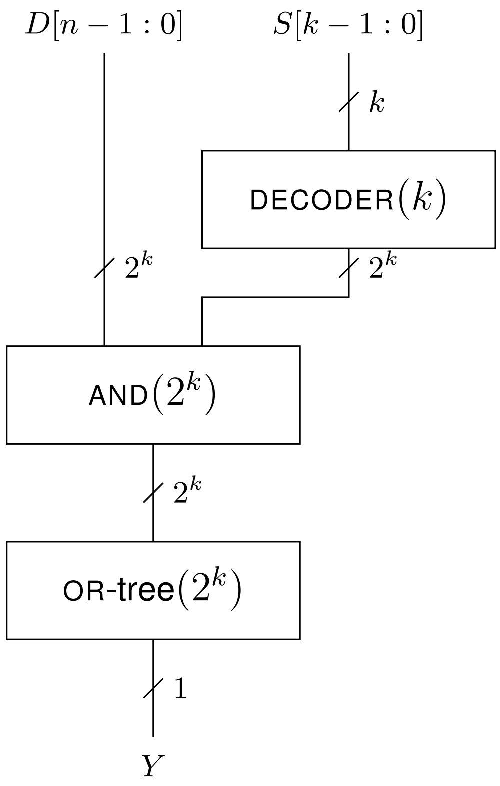
- $\vec{D}$ - data input
- $\vec{S}$ - select input
- simplify: assume that n is a power of 2, namely, $n = 2^k$.

Implementation of (n:1)-MUX

We will present two implementations:

- a decoder based implementation
- a tree-like implementation

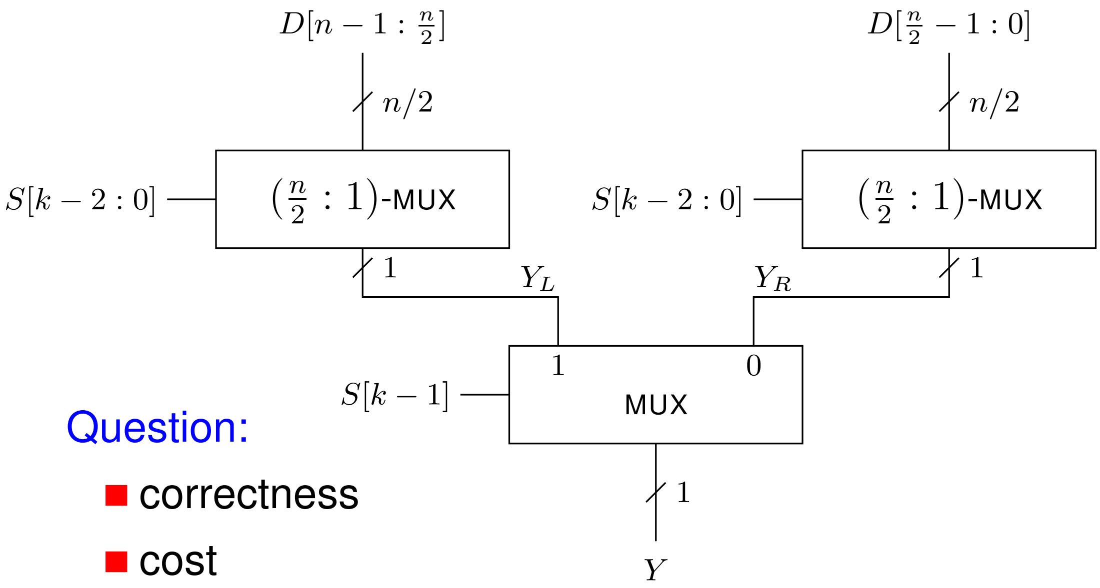
(n:1)-MUX : a decoder based implementation



Question:

- correctness
- cost
- delay
- asymptotic optimality

(n:1)-MUX : a tree-like implementation



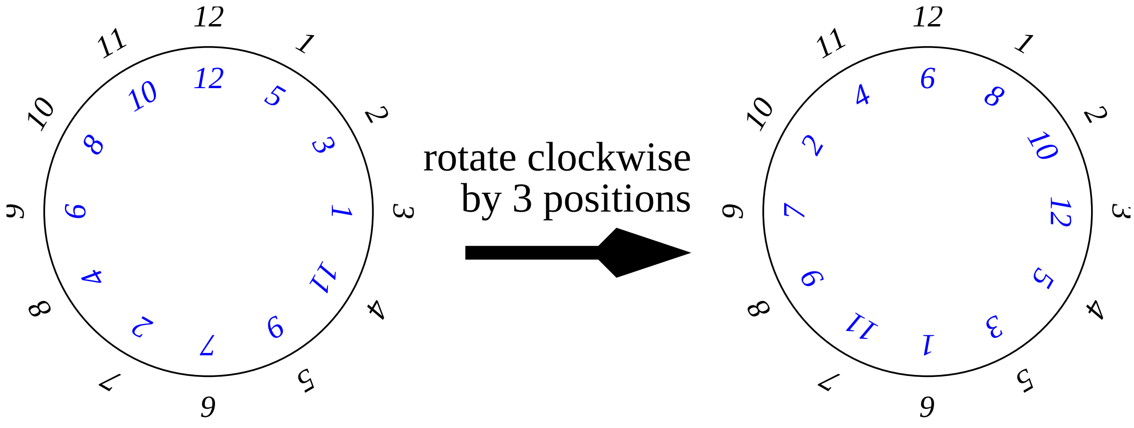
Question:

- correctness
- cost
- delay
- asymptotic optimality

Which design is better?

- both designs are asymptotically optimal.
- based on the tables of Müller & Paul, the tree-like design is better.
- decision is based on specific gate costs in the technology one uses.
- fast MUX-gates in CMOS (transmission gates) do not restore the signals well.
  $\Rightarrow$ long paths consisting only of MUX-gates are not allowed.

Cyclic shift - example



"clock"reads:
5,3,1,11,...,8,10,12

"clock"reads:
8,10,12,...,2,4,6

Cyclic shift - definition

The string $b[n-1:0]$ is a cyclic left shift by i positions of the string $a[n-1:0]$ if

$$\forall j : \quad b[j] = a[\text{mod}(j+i, n)].$$

Example: Let $a[3:0] = 0010$. A cyclic left shift by one position of $\vec{a}$ is the string 0100. A cyclic left shift by 3 positions of $\vec{a}$ is the string 0001.

Barrel Shifter

DEF: A BARREL-SHIFTER(n) is a combinational circuit defined as follows:

Input: $x[n-1:0]$ and $sa[k-1:0]$ where $k = \lceil \log_2 n \rceil$.

Output: $y[n-1:0]$.

Functionality: $\vec{y}$ is a cyclic left shift of $\vec{x}$ by $\langle \vec{sa} \rangle$ positions. Formally,

$$\forall j \in [n-1:0] : \quad y[j] = x[\text{mod}(j + \langle \vec{sa} \rangle, n)].$$

- $\vec{x}$ - data input
- $\vec{sa}$ - shift amount input
- simplify - assume that n is a power of 2, namely, $n = 2^k$.

CLS(n, i) - Cyclic Left Shift by 2^i positions

DEF: A CLS(n, i) is a combinational circuit defined as follows:

Input: $x[n-1:0]$ and $s \in \{0, 1\}$.

Output: $y[n-1:0]$.

Functionality:

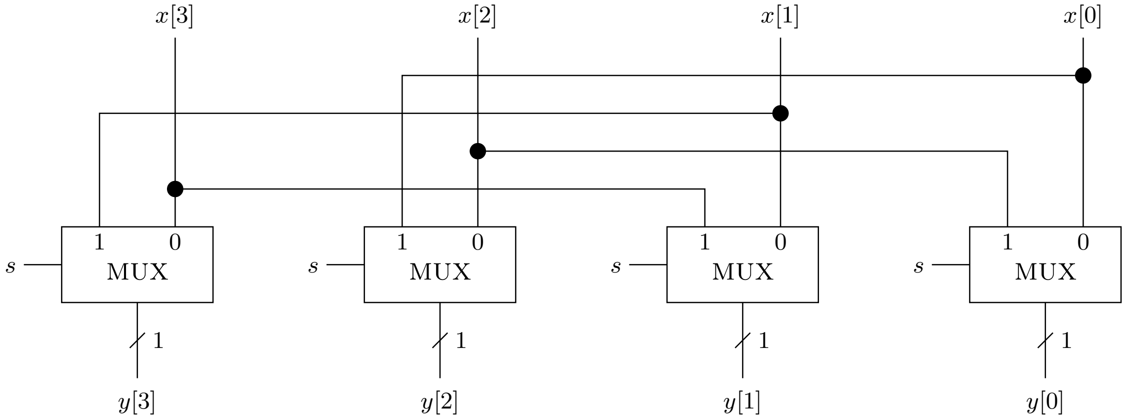
$$\forall j \in [n-1:0] : \quad y[j] = x[\text{mod}(j + s \cdot 2^i, n)].$$

Equivalently,

$$y[j] = \begin{cases} x[j] & \text{if } s = 0 \\ x[\text{mod}(j + 2^i, n)] & \text{if } s = 1. \end{cases}$$

$\Rightarrow$ can implement CLS(n, i) with a row of n MUX-gates.

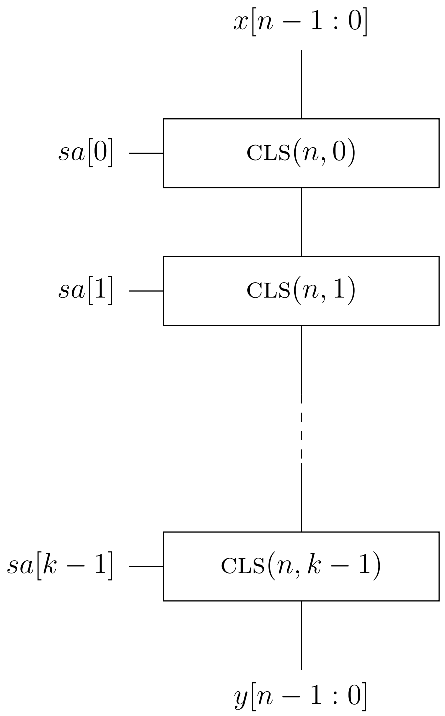
CLS$(4, 1)$



Evident that a CLS(n, i) requires a lot of area for the wires.

Our model does not capture routing cost.

BARREL-SHIFTER(n) - a chain of CLS(n, i)



BARREL-SHIFTER(n) - correctness

Define the strings $y_i[n-1:0]$, for $0 \leq i \leq k-1$, recursively as follows:

$$y_0[n-1:0] \leftarrow \text{CLS}_{n,0}(x[n-1,0], sa[0])$$
$$y_{i+1}[n-1:0] \leftarrow \text{CLS}_{n,i+1}(y_i[n-1,0], sa[i+1])$$

Claim: $y_{k-1}[n-1:0]$ is a cyclic left shift of $x[n-1:0]$ by $\langle sa[k-1:0]\rangle$ positions.

Proof: Induction. $k=0$ - trivial because $\text{CLS}(n,0)$ shifts by zero/one position.

induction step

- $$y_i[j] = \text{CLS}_{n,i}(y_{i-1}[n-1,0], sa[i])[j] \quad \text{(by definition of } y_i)$$
  $$= y_{i-1}[\text{mod}(j + 2^i \cdot sa[i], n)] \quad \text{(by definition of } \text{CLS}_{n,i}).$$
- Let $\ell = \text{mod}(j + 2^i \cdot sa[i], n)$.
- Ind. Hyp. $\Rightarrow y_{i-1}[\ell] = x[\text{mod}(\ell + \langle sa[i-1:0]\rangle, n)$.
- Note that

$$\text{mod}(\ell + \langle sa[i-1:0]\rangle, n) = \text{mod}(j + 2^i \cdot sa[i] + \langle sa[i-1:0]\rangle, n)$$
$$= \text{mod}(j + \langle sa[i:0]\rangle, n).$$

- Therefore $y_i[j] = x[\text{mod}(j + \langle sa[i:0]\rangle, n)]$, and the claim follows.

Priority Encoders

- Priority encoders detect the position of the leading one.
- Example: $x[0:6] = 0110100$. The leading one is $x[1]$.
- Position of leading one can be represented in binary or unary representation.
- DEF: A binary string $x[0:n-1]$ represents a number in unary representation if $x[0:n-1] \in 1^* \cdot 0^*$. The value represented in unary representation by the binary string $1^i \cdot 0^j$ is i.
- Example: 01001011 does not represent a number in unary representation. 111100 represents 4 in unary representation.

Unary priority encoder

U-PENC(n) - a combinational circuit specified as follows.

Input: $x[0:n-1]$.

Output: $y[0:n-1]$.

Functionality:

$$y[i] = \text{OR}(x[0:i]).$$

- Example: $x[0:6] = 0010101 \Longrightarrow y[0:6] = 0011111$.
- $\vec{x} \neq 0^n \quad \Longrightarrow y[0:n-1] = 0^j \cdot 1^{n-j}$, where $j = \min\{i \mid x[i] = 1\}$.
- Hence $\text{INV}(\vec{y})$ is a unary representation of the position of the leading one of the string $\vec{x} \cdot 1$.

Binary priority encoder

B-PENC(n) - a combinational circuit specified as follows.

Input: $x[0 : n-1]$.

Output: $y[k : 0]$, where $k = \lfloor \log_2 n \rfloor$. (Note that if $n = 2^\ell$, then $k = \ell$.)

Functionality:

$$\langle \vec{y} \rangle = \begin{cases} \min\{i \mid x[i] = 1\} & \text{if } \vec{x} \neq 0^n \\ n & \text{if } \vec{x} = 0^n. \end{cases}$$

Example: Given input $x[0 : 5] = 000101$, a U-PENC(6) outputs $y[0 : 5] = 000111$, and B-PENC(6) outputs $y[2 : 0] = 011$.

U-PENC(n) - Implementation

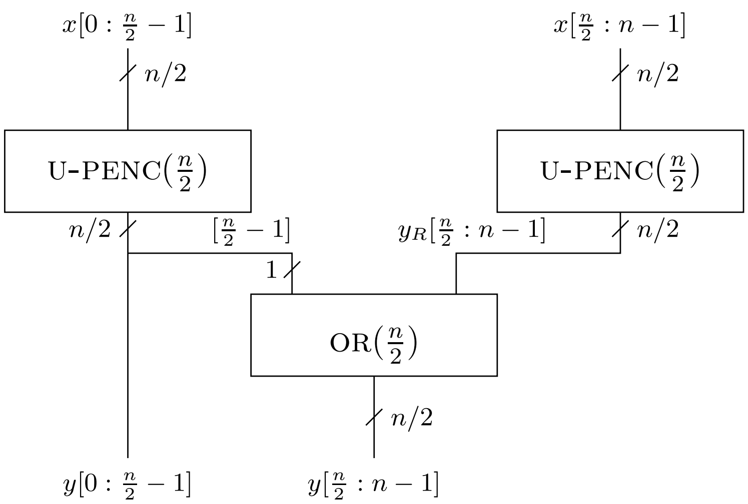
a brute force unary priority encoder:

- separate OR-tree for each output bit.
- delay $O(\log n)$ and cost $O(n^2)$.

- How can we efficiently combine these trees? To be discussed in detail when we discuss parallel prefix computation.
- Now we present a design based on divide-and-conquer.

divide & conquer U-PENC(n)

If $n = 1$, then $y[0] \leftarrow x[0]$.



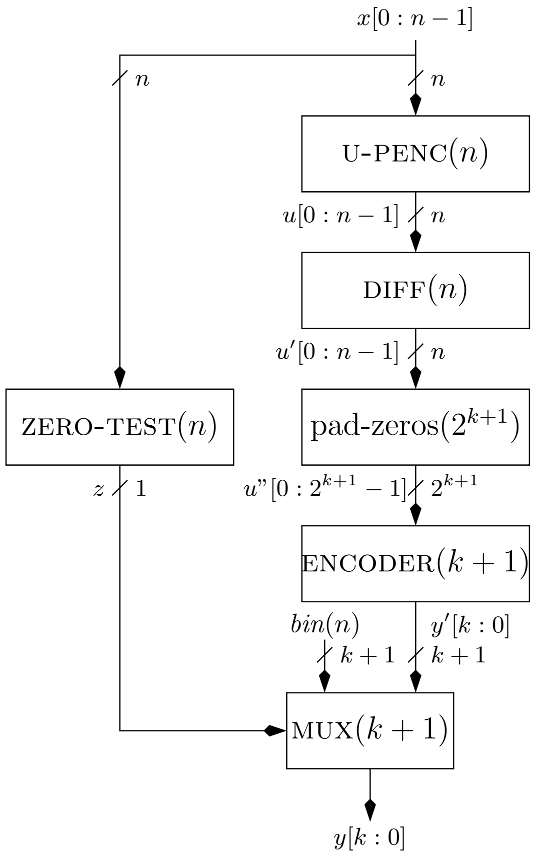
- Question: Prove correctness.
- cost = $O(n \log n)$ (not optimal!)
- delay = $O(\log n)$ (optimal)

Implementation of a binary priority encoder

We present two designs for a binary priority encoder.

- design based on a reduction to a unary priority encoder.
- design based on divide-and-conquer

reduction of B-PENC(n) to U-PENC(n)

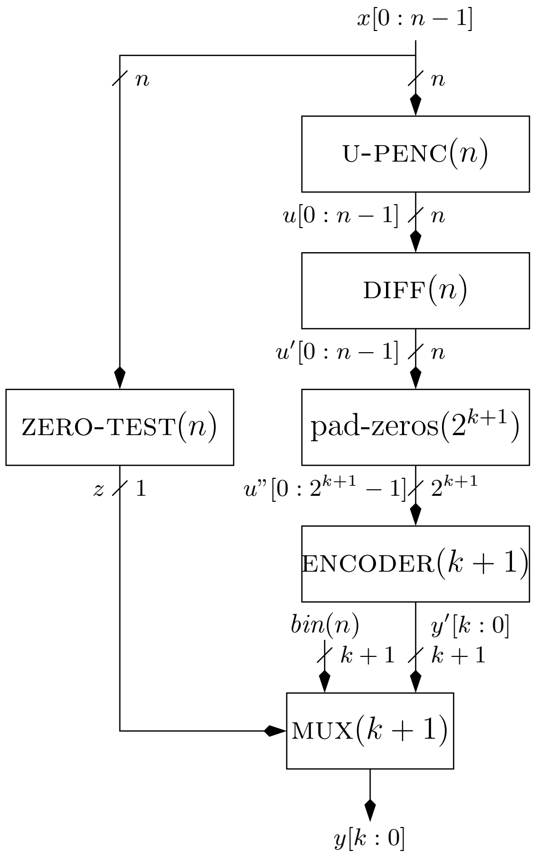


- apply U-PENC(n)
- difference:

$$u'[i] = \begin{cases} u[0] & \text{if } i = 0 \\ u[i] - u[i-1] & \text{otherwise.} \end{cases}$$

  if $\vec{x} \neq 0^n$, then $\vec{u'}$ is a 1-out-of-n representation of leading one's position.
- Pad & Encode.
- Select *bin*(n) if $\vec{x} = 0^n$.

cost analysis

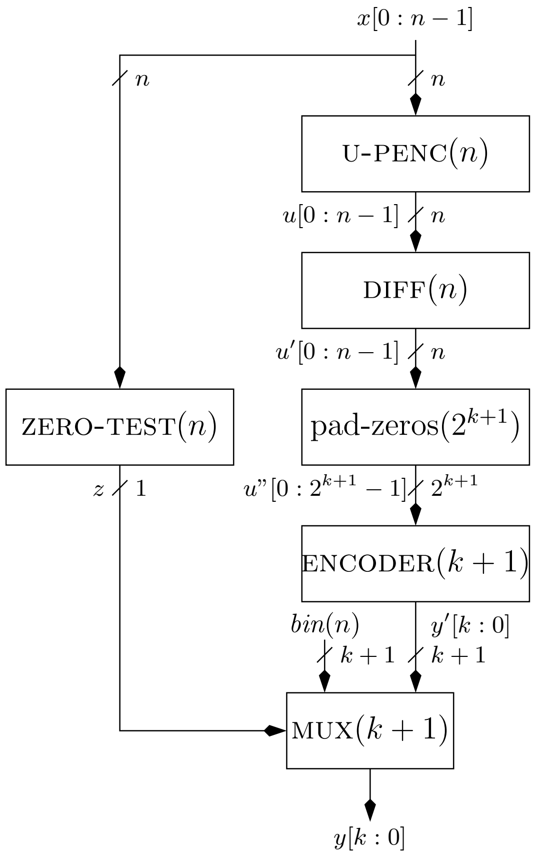


- zero cost: padding with zeros.
- logarithmic cost: multiplexer
- linear cost:
  - difference
  - encoder
  - zero tester
- $\Longrightarrow$

$$c(\text{B-PENC}(n)) = c(\text{U-PENC}(n)) + O(n).$$

- If $c(\text{U-PENC}(n)) = O(n)$, then also $c(\text{B-PENC}(n)) = O(n)$.

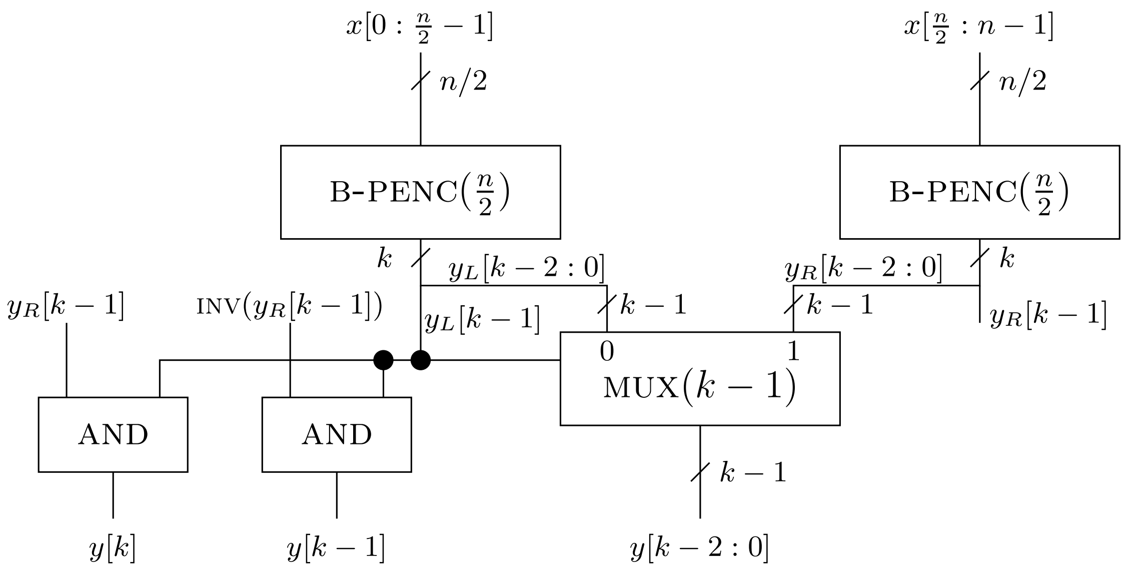
delay analysis



- zero delay: padding with zeros.
- constant delay:
  - difference
  - multiplexer
- logarithmic delay:
  - U-PENC(n)
  - encoder
  - zero tester
- $\Longrightarrow$

$$d(\text{B-PENC}(n)) = O(\log n).$$

a divide-and-conquer design for $n = 2^k$

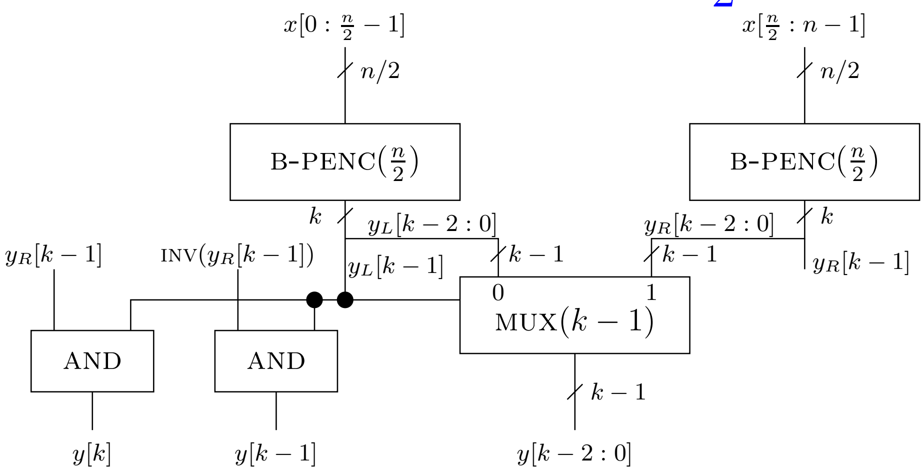


correctness

We prove correctness by induction.

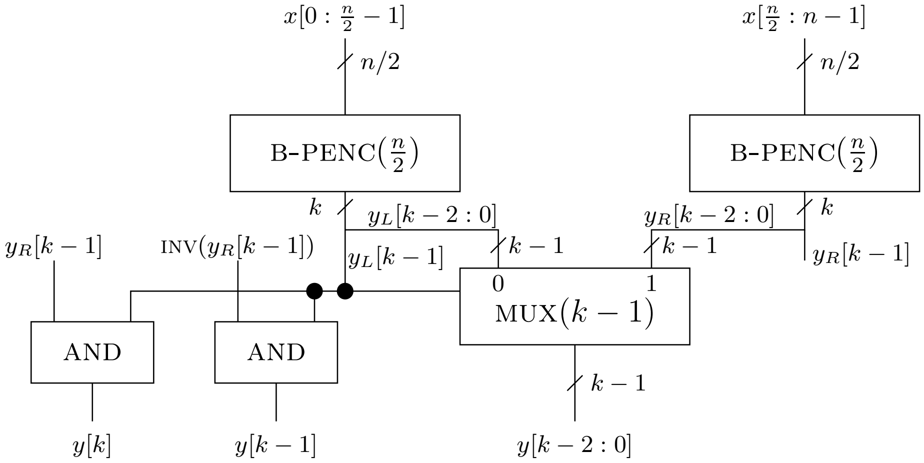
- Induction basis: $n = 1$ is trivial.
- Induction step deals with 3 cases:
  - leading one is in left half $x[0 : \frac{n}{2} - 1]$.
  - leading one is in right half $x[\frac{n}{2} : n - 1]$.
  - $\vec{x} = 0^n$.

correctness: case $x[0 : \frac{n}{2} - 1] \neq 0^{n/2}$



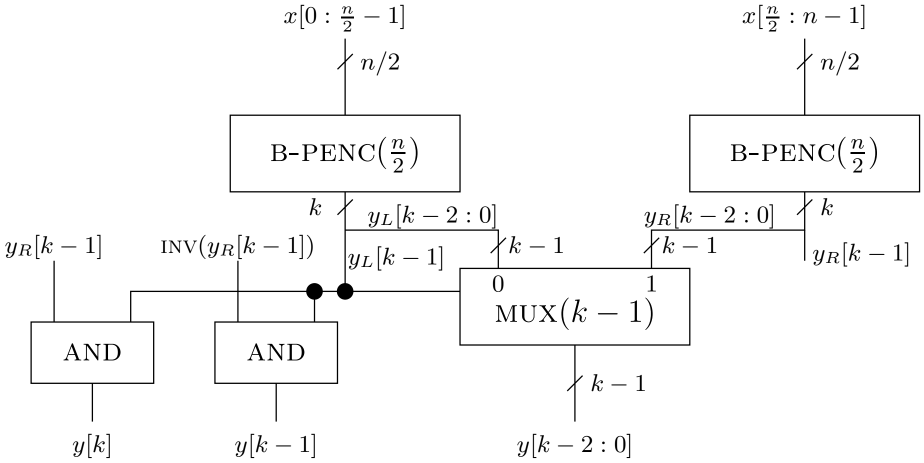
- Ind. Hyp. $\Rightarrow$ required output is $0 \cdot y_L[k-1:0]$.
- index of the leading one $< n/2 \Rightarrow \quad y_L[k-1] = 0$.
- $\Rightarrow \quad y[k] = y[k-1] = 0$ and $y[k-2:0] = y_L[k-2:0]$.
- $\Rightarrow$ output $\vec{y} = 0 \cdot y_L[k-1:0]$.

correctness: case $x[0 : \frac{n}{2} - 1] = 0^{n/2}$ & $x[\frac{n}{2} : n - 1] \neq 0^{n/2}$



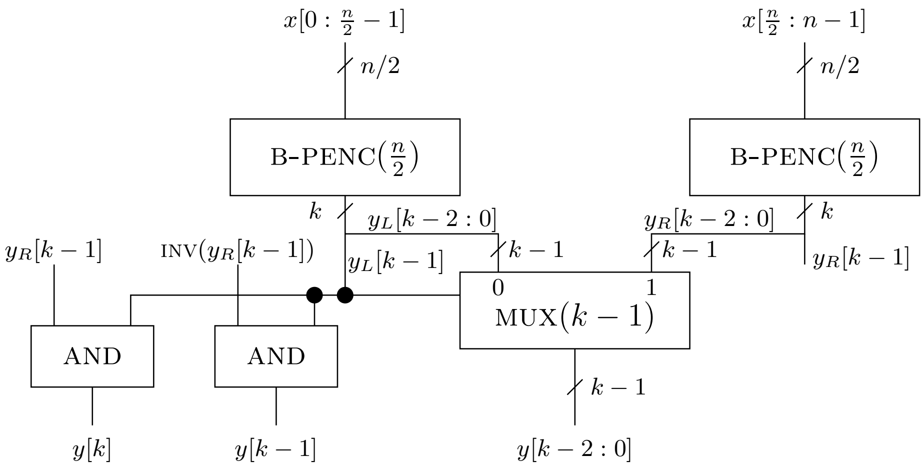
- Ind. Hyp. $\Rightarrow$ index of the leading one is $n/2 + \langle \vec{y_R} \rangle$.
- $\Rightarrow$ required output is $0 \cdot 1 \cdot y_R[k-2:0]$.
- Ind. Hyp. $\Rightarrow$ $y_L[k-1] = 1$ and $y_R[k-1] = 0$.
- $\Rightarrow$ $y[k] = 0$ & $y[k-1] = 1$.
- $y_L[k-1] = 1 \quad \Rightarrow \quad y[k-2:0] = y_R[k-2:0]$.

correctness: case $x[0 : \frac{n}{2} - 1] = 0^{n/2}$ & $x[\frac{n}{2} : n - 1] = 0^{n/2}$



- required output is $1 \cdot 0^k$.
- Ind. Hyp. $\Rightarrow$ $y_L[k-1:0] = y_R[k-1:0] = 1 \cdot 0^{k-1}$.
- $y_L[k-1] = y_R[k-1] = 1 \quad \Rightarrow \quad y[k] = 1$.
- $y_L[k-1] = y_R[k-1] = 1 \quad \Rightarrow \quad y[k-1] = 0$.
- $y_L[k-1] = 1 \quad \Rightarrow \quad y[k-2:0] = y_L[k-2:0] = 0^{k-1}$.

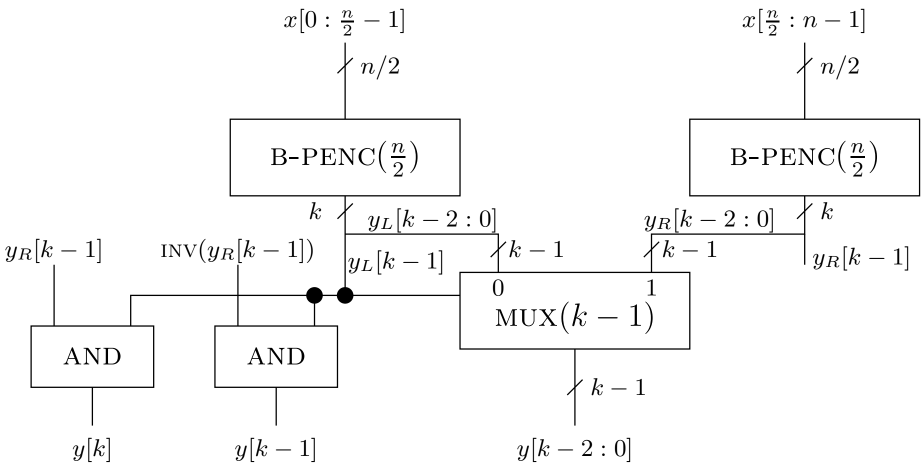
cost analysis



$$c(\text{B-PENC}(n)) = \begin{cases} c(\text{NOR}) & \text{if n=2} \\ 2 \cdot c(\text{B-PENC}(n/2)) + 2 \cdot c(\text{AND}) & \\ +(k-1) \cdot c(\text{MUX}) & \text{otherwise.} \end{cases}$$

Solution is $c(n) = O(n)$, same recurrence as for ENCODER$''(k)$ with substitution $k = \log n$.

delay analysis



$$d(\text{B-PENC}(n)) = \begin{cases} t_{pd}(\text{NOR}) & \text{if n=2} \\ d(\text{B-PENC}(n/2)) + \max\{d(\text{MUX}) + d(\text{AND})\} & \text{otherwise.} \end{cases}$$

$$\Longrightarrow \quad d(\text{B-PENC}(n)) = O(\log n)$$

Summary - priority encoders

- Two types of priority encoders: U-PENC(n) & B-PENC(n).
- Implementation of U-PENC(n):
  - brute force (separate OR-trees): cost $O(n^2)$, delay $O(\log n)$.
  - divide-&-conquer: cost $O(n \log n)$, delay $O(\log n)$.
  - to be shown: cost $O(n)$ and delay $O(\log n)$.
- Implementation of B-PENC(n):
  - reduction to U-PENC: overhead is cost - $O(n)$ & delay - $O(\log n)$.
  - divide-&-conquer: cost $O(n)$, delay $O(\log n)$.
  - will later show techniques based on adder trees.

Half-decoder

DEF: H-DEC(n) is a combinational circuit defined as follows:

Input: $x[n-1:0]$.

Output: $y[0:2^n-1]$

Functionality:

$$y[0:2^n-1] = 1^{\langle x[n-1:0]\rangle} \cdot 0^{2^n - \langle x[n-1:0]\rangle}.$$

- Decoder: binary $\to$ 1-out-of-2^n.
- Half-decoder: binary $\to$ unary.
- Example: $x[2:0] = 101 \quad \Longrightarrow \quad y[0:7] = 11111000.$
- Example: $\vec{x} = 0^n \quad \Longrightarrow \quad \vec{y} = 0^{2^n}$.
- Example: $\vec{x} = 1^n \quad \Longrightarrow \quad \vec{y} = 1^{2^n-1} \cdot 0.$
- Remark: always $y[2^n-1] = 0$.

Try to design H-DEC(n) using known modules

Question: Suggest an implementation of a half-decoder based on a decoder and a unary priority encoder. Analyze the cost and delay of the design. Is it optimal with respect to cost or delay?

Look for a better design...

Claim 1

$$y[i] = 1 \iff i < \langle \vec{x} \rangle.$$

Follows trivially from definition of H-DEC(n):

$$y[0 : 2^n - 1] = 1^{\langle x[n-1:0] \rangle} \cdot 0^{2^n - \langle x[n-1:0] \rangle}.$$

Claim 2

- $z[0:n-1]$ - represents the number *wt*$(\vec{z})$ in unary representation.
- $i \in [0, n-1]$ - a fixed constant

Easy to compare *wt*$(\vec{z})$ and i:

$$wt(\vec{z}) < i \iff z[i-1] = 0$$
$$wt(\vec{z}) > i \iff z[i] = 1$$
$$wt(\vec{z}) = i \iff z[i] = 0 \text{ and } z[i-1] = 1.$$

Example:

$$z[0:5] = 111100 \Rightarrow wt(\vec{z}) = 4$$
$$wt(\vec{z}) < 4 \iff z[3] = 0$$
$$wt(\vec{z}) > 4 \iff z[4] = 1$$
$$wt(\vec{z}) = 4 \iff z[4] = 0 \text{ and } z[3] = 1.$$

Comparison box COMP$(\vec{z}, i)$

Input: $z[0:n-1]$

Output: $GT, EQ, LT \in \{0, 1\}$.

Functionality:

$$GT = 1 \text{ if } wt(\vec{z}) > i$$
$$EQ = 1 \text{ if } wt(\vec{z}) = i$$
$$LT = 1 \text{ if } wt(\vec{z}) < i$$

Implementation:

$GT = z[i]$

$LT = \text{INV}(z[i-1])$

$EQ = \text{AND}(z[i-1], \text{INV}(z[i]))$ $\quad$ ($wt(\vec{z}) < i+1$ and $wt(\vec{z}) > i-1$)

Claim 3: comparison based on quotient & remainder

- partition $\vec{x}$ into $\vec{x}_L$ and $\vec{x}_R$

  $x_L[n-k-1:0] = x[n-1:k]$ and $x_R[k-1:0] = x[k-1:0]$.

- Binary representation implies that

$$\langle \vec{x} \rangle = 2^k \cdot \langle \vec{x}_L \rangle + \langle \vec{x}_R \rangle.$$

- Divide $i \in [0, 2^n - 1]$ by 2^k:

$$i = 2^k \cdot q + r, \quad \text{where } r \in [0, 2^k - 1].$$

- claim:

$$i < \langle \vec{x} \rangle \iff q < \langle \vec{x}_L \rangle \text{ or } (q = \langle \vec{x}_L \rangle \text{ and } r < \langle \vec{x}_R \rangle)$$

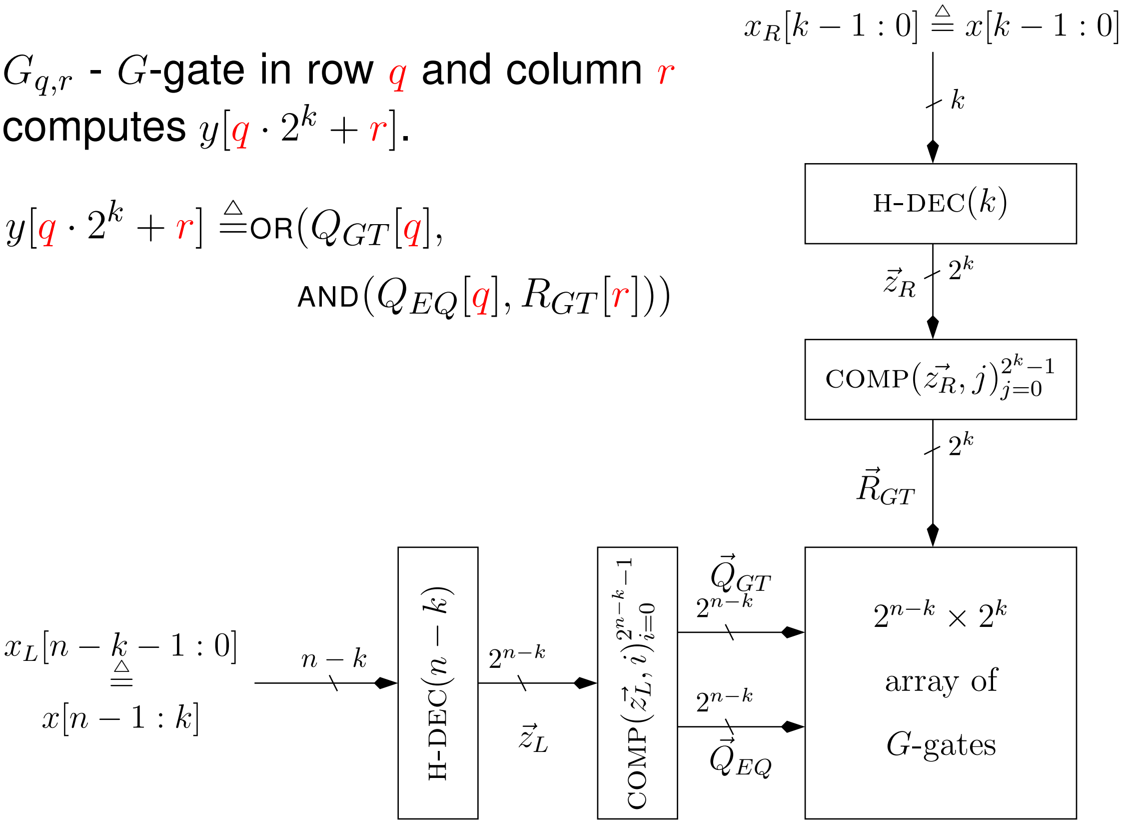
Implementation: H-DEC(n)

- Recursive divide-and-conquer design.
- Recursion basis: H-DEC(1) - $y[0] \leftarrow x[0]$. (unary representation $\equiv$ binary representation for a single bit).
- Recursion step...

Recursion step: H-DEC(n)

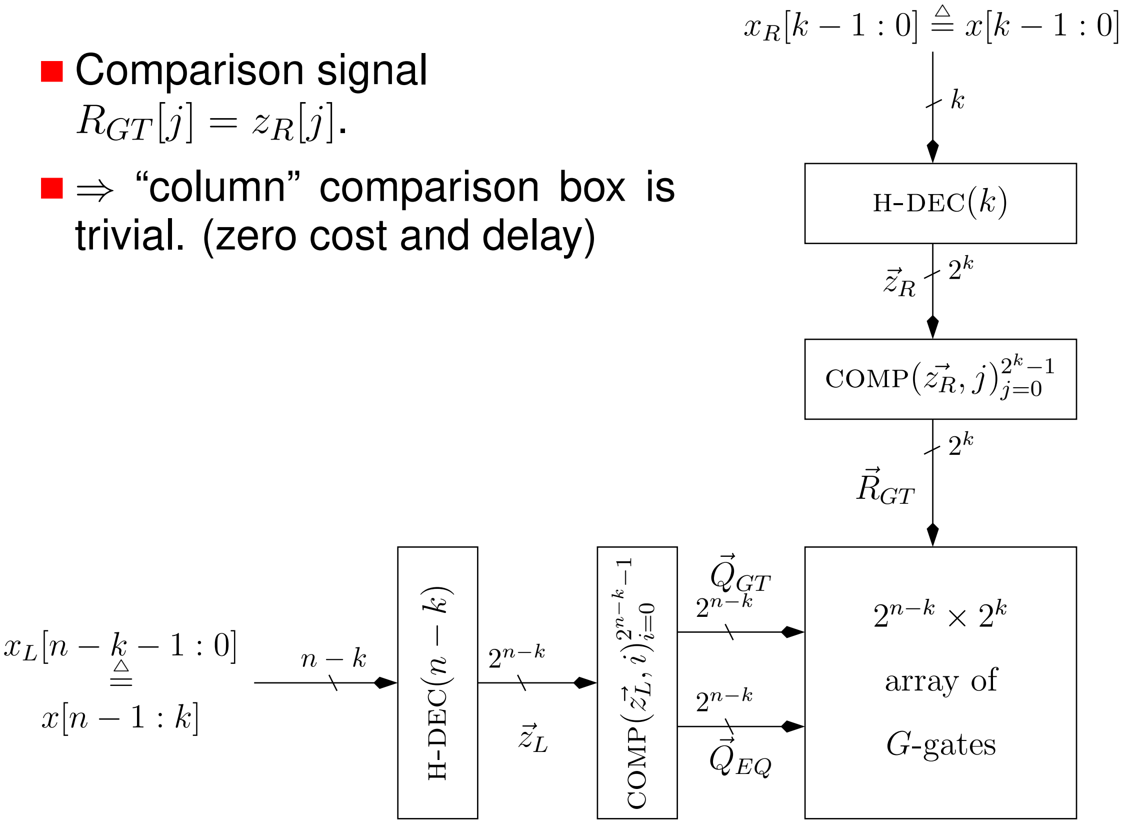
$G_{q,r}$ - G-gate in row q and column r computes $y[q \cdot 2^k + r]$.

$$y[q \cdot 2^k + r] \triangleq \text{OR}(Q_{GT}[q], \text{AND}(Q_{EQ}[q], R_{GT}[r]))$$



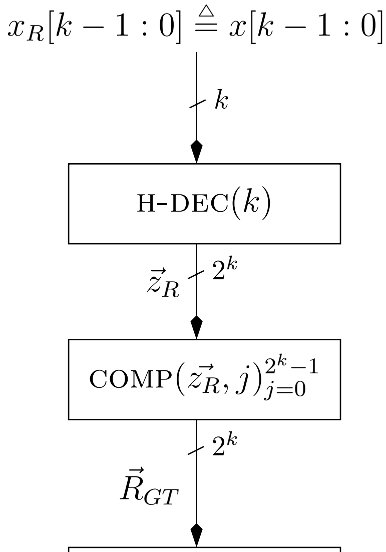
remarks

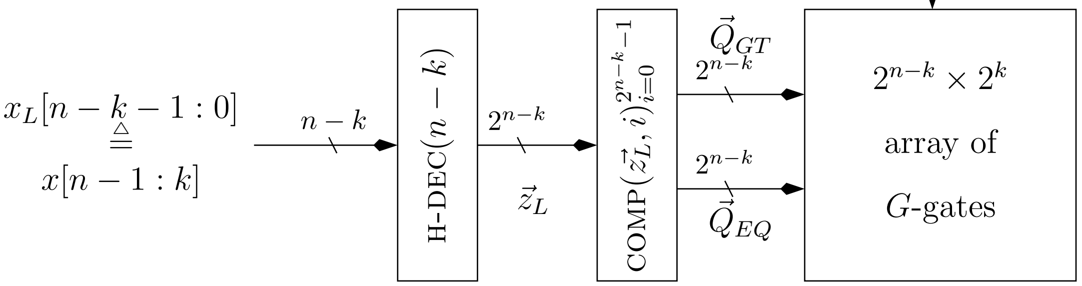
- Comparison signal $R_{GT}[j] = z_R[j]$.
- $\Rightarrow$ "column" comparison box is trivial. (zero cost and delay)



example

- $n = 4, k = 2$.
- $i = 6 \Longrightarrow q = 1, r = 2$.
- $y[6] = 1 \Longleftrightarrow \langle x[3:0] \rangle > 6$
- $\Longleftrightarrow (\langle x[3:2] \rangle > 1)$
  or $(\langle x[3:2] \rangle = 1$ and $\langle x[1:0] \rangle > 2)$
- $Q_{GT}[q] = 1 \Longrightarrow y[6] = 1$.
- $Q_{EQ}[q] = 1 \Longrightarrow y[6] = R_{GT}[r]$.
- otherwise, $y[6] = 0$





Correctness: H-DEC(n)

- Proof by induction on n.
- Induction basis: H-DEC(1) - trivial.
- Induction step: by Claim 1, suffices to show that

$$y[i] = 1 \Longleftrightarrow i < \langle \vec{x} \rangle.$$

- Claim 3:

$$i < \langle \vec{x} \rangle \Longleftrightarrow (q < \langle \vec{x}_L \rangle) \text{ or } ((q = \langle \vec{x}_L \rangle) \text{ and } (r < \langle \vec{x}_R \rangle)).$$

- The induction hypothesis implies that:

$$\begin{aligned} q < \langle \vec{x}_L \rangle &\Longleftrightarrow z_L[q] = 1 \\ q = \langle \vec{x}_L \rangle &\Longleftrightarrow z_L[q] = 0 \text{ and } z_L[q-1] = 1 \\ r < \langle \vec{x}_R \rangle &\Longleftrightarrow z_R[r] = 1. \end{aligned}$$

Correctness - cont.

- Definition of comparison boxes implies that:

$$q < \langle \vec{x}_L \rangle \iff z_L[q] = 1 \iff Q_{GT}[q] = 1$$
$$q = \langle \vec{x}_L \rangle \iff \text{INV}(z_L[q]) \cdot z_L[q-1] \iff Q_{EQ}[q] = 1$$
$$r < \langle \vec{x}_R \rangle \iff z_R[r] = 1 \iff R_{GT}[r] = 1.$$

- $G_{q,r}$ outputs:

$$y[i] = \text{OR}(Q_{GT}[q], \text{AND}(Q_{EQ}[q], R_{GT}[r])).$$

- $\Longrightarrow y[i]$ is correct. QED

Cost analysis: H-DEC(n)

$$c(\text{H-DEC}(n)) = \begin{cases} 0 & \text{if n=1} \\ c(\text{H-DEC}(k)) + c(\text{H-DEC}(n-k)) & \\ +2^{n-k} \cdot c(EQ) + 2^n \cdot c(G) & \text{otherwise.} \end{cases}$$

- cost of computing the EQ is $c(\text{INV}) + c(\text{AND})$.
- $c(G) = c(\text{AND}) + c(\text{OR})$.
- It follows that

$$c(\text{H-DEC}(n)) = c(\text{H-DEC}(k)) + c(\text{H-DEC}(n-k)) + \Theta(2^n)$$

- same recurrence in the case of decoders.
$\Rightarrow c(\text{H-DEC}(n)) = \Theta(2^n)$.

H-DEC(n) - lower bound on cost

Question*: Prove that every implementation of a half-decoder design must contain at least $2^n - 2$ non-trivial gates. (Here we assume that every non-trivial gate has a single output, and we do not have any fan-in or fan-out restrictions).

Delay analysis: H-DEC(n)

$$d(\text{H-DEC}(n)) = \begin{cases} 0 & \text{if n=1} \\ \max\{d(\text{H-DEC}(k)), \\ \qquad d(\text{H-DEC}(n-k)) + d(EQ)\} & \\ +d(G) & \text{otherwise.} \end{cases}$$

- $d(EQ), d(G)$ are constant.
- Set $k = \lceil \frac{n}{2} \rceil$, then the recurrence degenerates to

$$\begin{aligned} d(\text{H-DEC}(n)) &= d(\text{H-DEC}(n/2)) + \Theta(1) \\ &= \Theta(\log n). \end{aligned}$$

- delay of H-DEC(n) is asymptotically optimal since all the inputs belong to the cone of a half-decoder.

Summary - Half decoders

- half decoder translates binary $\longrightarrow$ unary.
- implementation:
  - variation of decoder design (divide-and-conquer with array of simple gates).
  - based on comparison of a unary number with a constant.
  - asymptotically optimal cost and delay.