Chapter 4: Decoders & Encoders

Computer Structure & Intro. to Digital Computers

Dr. Guy Even

Tel-Aviv Univ.

Goals

- vector notation: buses, indexed signals, multiple copies of gates
- representation: binary representation
- Decoders:
  - definition (specification)
  - implementation
  - correctness proof
  - analyze delay & cost
  - optimality
- Encoders: definition (specification), correctness proof, analyze delay & cost, optimality

Bus related definitions

- bus - an indexed set of buses (e.g. $a[1:4]$).
- reversing - reversing the indexes of buses: e.g. if reversing is used, then $b[4:1] \leftarrow a[1:4]$ means $b[1] \leftarrow a[4], \ldots, b[4] \leftarrow a[1]$.
- hardwired shifting - shifting of indexes: e.g. hardwired shifting implies that $b[5:8] \leftarrow a[1:4]$ means $b[5] \leftarrow a[1], \ldots, b[8] \leftarrow a[4]$.

Bus assignment conventions

- Unless stated explicitly, reversing is not used. The assignments:

  $$b[i:j] \leftarrow a[i:j] \quad \& \quad b[j:i] \leftarrow a[i:j]$$

  have the same meaning (so we don't have to worry about ascending/descending indexes).
- Hardwired-shifting is used for shifted index ranges. For example,

  $$b[i+5:j+5] \leftarrow a[i:j]$$

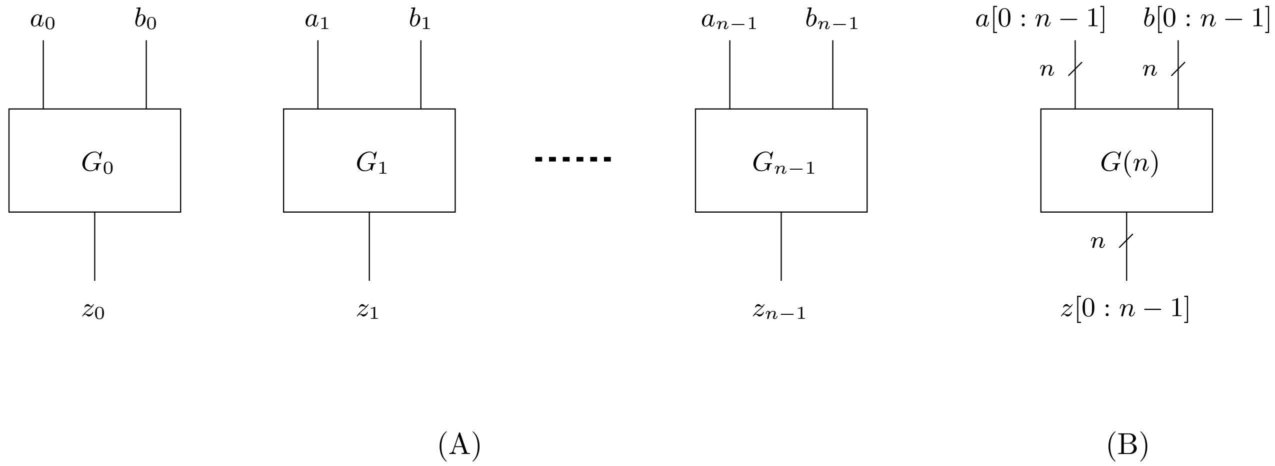
  means

  $$b[i+5] \leftarrow a[i] \quad b[i+6] \leftarrow a[i+1] \quad \cdots \quad b[j+5] \leftarrow a[j]$$

Signals on buses

- Consider a bus $a[5:1]$.
- The signal on $a[3]$ at time t is $a[3](t)$.
- Abbreviate and simply write $a[3]$ as the signal on $a[3]$ after it stabilizes.
- Similarly: $a[5:1]$ refers both to the bus and to the stable signal on the bus.
- $\Longrightarrow a[5:1]$ is both a bus and a binary string.
- Abbreviate $a[5:1]$ and write $\vec{a}$ if index range is clear from context.

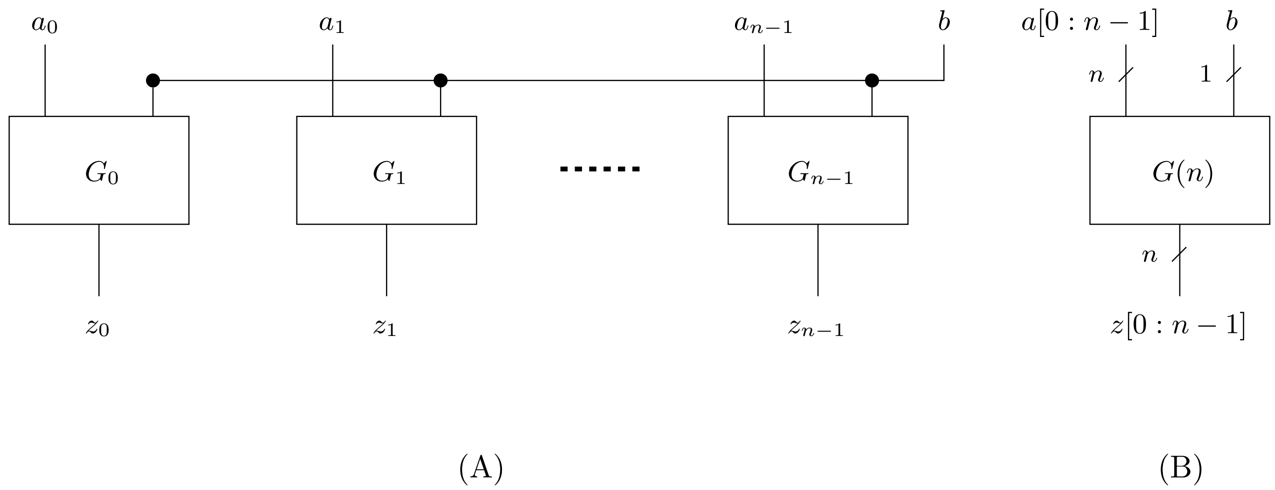
multiple instances of the same gate



- G_i - the ith instance of gate G in $G(n)$.
- a_i, b_i - the two input terminals of G_i.
- z_i - the output terminal of G_i.

Common input in $G(n)$

b is fed to the second input terminal of all the gates.



Fanout of the net b is n. In practice, a large fanout increases the capacity of a net and causes an increase in the delay of the circuit. We usually ignore this phenomena in this course.

Concatenation of binary strings

- assume that a and b are binary strings.
- $a \cdot b$ - the string obtained by concatenating a and b.
- example: if $a = 01$ and $b = 10$, then $a \cdot b = 0110$.
- a^i - the string $\underbrace{a \cdot a \cdots a}_{i \text{ times}}$.
- example: if $a = 01$ and $i = 3$, then $a^i = 010101$.
- If A is a set of strings, then A^i is the set of strings

$$A^i = \{a_1 \cdots a_i : \forall j \leq i : a_j \in A\}.$$

- example: $\{0,1\}^n$ - set of binary strings of length n.
- $A^* \triangleq \cup_{i=0}^{\infty} A^i$ ($A^0 = \{\Lambda\}$, not the empty set).
- $A^+ \triangleq \cup_{i=1}^{\infty} A^i$.

Values represented by binary strings

Binary strings can be used to represent numbers. Among the many methods for representing natural numbers:

- Binary representation
- Unary representation
- 1-out-of-n (one-hot).

Binary representation

The value represented in binary representation by a binary string $a[n-1:0]$ is denoted by $\langle a[n-1:0]\rangle$. It is defined as follows

$$\langle a[n-1:0]\rangle \triangleq \sum_{i=0}^{n-1} a_i \cdot 2^i.$$

- Could define a function $\langle\rangle_n : \{0,1\}^n \to [0, 2^n - 1]$.
- Usually omit the parameter n - clear from context.

Inverse function *bin* $: [0, 2^n - 1] \to \{0,1\}^n$.

$$\forall a[n-1:0] \in \{0,1\}^{:} \quad \textit{bin}_n(\langle a[n-1:0]\rangle) = a[n-1:0].$$

Division by 2^k in binary representation

- reminder: division by b with remainder:

$$x = q \cdot b + r, \quad \text{where } r \in [0, b-1].$$

  - q - quotient equals $\lfloor x/b \rfloor$
  - r - remainder equals $\text{mod}(x, b)$.
- Consider a binary string $a[n-1:0]$ and $x = \langle a[n-1:0] \rangle$.
- How do we compute (the binary representation of) $\lfloor x/2^k \rfloor$ & $\text{mod}(x, 2^k)$?
  - $r = \langle a[k-1:0] \rangle$
  - $q = \langle a'[n-k-1:0] \rangle$, where $a'[n-k-1:0] \leftarrow a[n-1:k]$ (with shifting).

Decoders

A decoder with input length n is a combinational circuit specified as follows:

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

Output: $y[2^n-1:0] \in \{0,1\}^{2^n}$

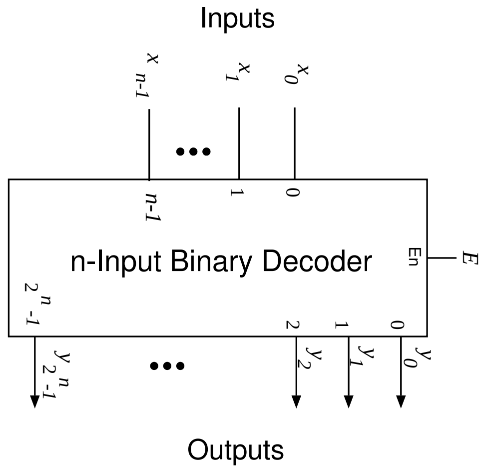
Functionality:

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

- $|$outputs$|$ of a decoder is exponential in $|$inputs$|$.
- exactly one bit of the output $\vec{y}$ is set to one. (called also one-hot encoding or 1-out-of-k encoding.)
- DECODER(n) - a decoder with input length n

example: DECODER(3) - on input $x = 101$, the output y equals 00100000.

Schematic symbol



Example: DECODER(3) - a truth table...

EXAMPLE 9.1: 3-INPUT BINARY DECODER

E	x_2	x_1	x_0	x	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	0	2	0	0	0	0	0	1	0	0
1	0	1	1	3	0	0	0	0	1	0	0	0
1	1	0	0	4	0	0	0	1	0	0	0	0
1	1	0	1	5	0	0	1	0	0	0	0	0
1	1	1	0	6	0	1	0	0	0	0	0	0
1	1	1	1	7	1	0	0	0	0	0	0	0
0	-	-	-	-	0	0	0	0	0	0	0	0

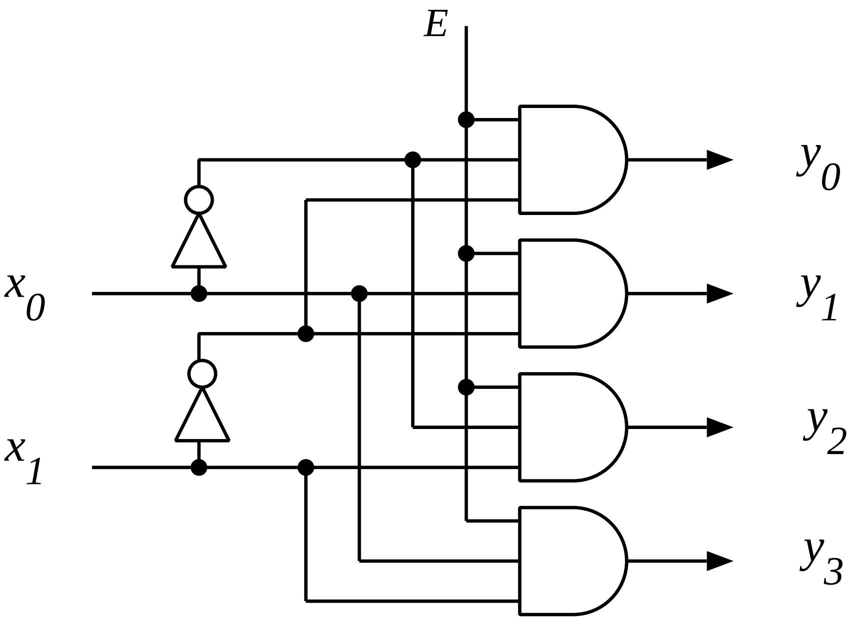
BINARY SPECIFICATION:

Inputs: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0,1\}$
$E \in \{0,1\}$

Outputs: $\underline{y} = (y_{2^n-1}, \ldots, y_0), \quad y_i \in \{0,1\}$

Function: $y_i = E \cdot m_i(\underline{x}) \quad , \quad i = 0, \ldots, 2^n - 1$

Example: DECODER(2) - schematic



An application

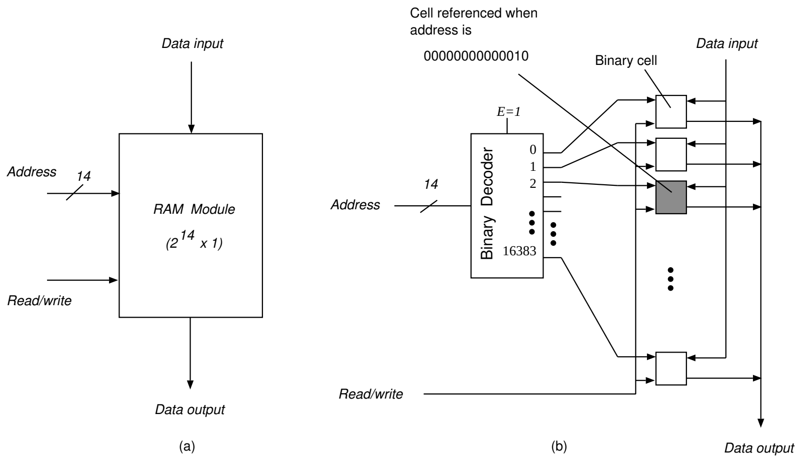
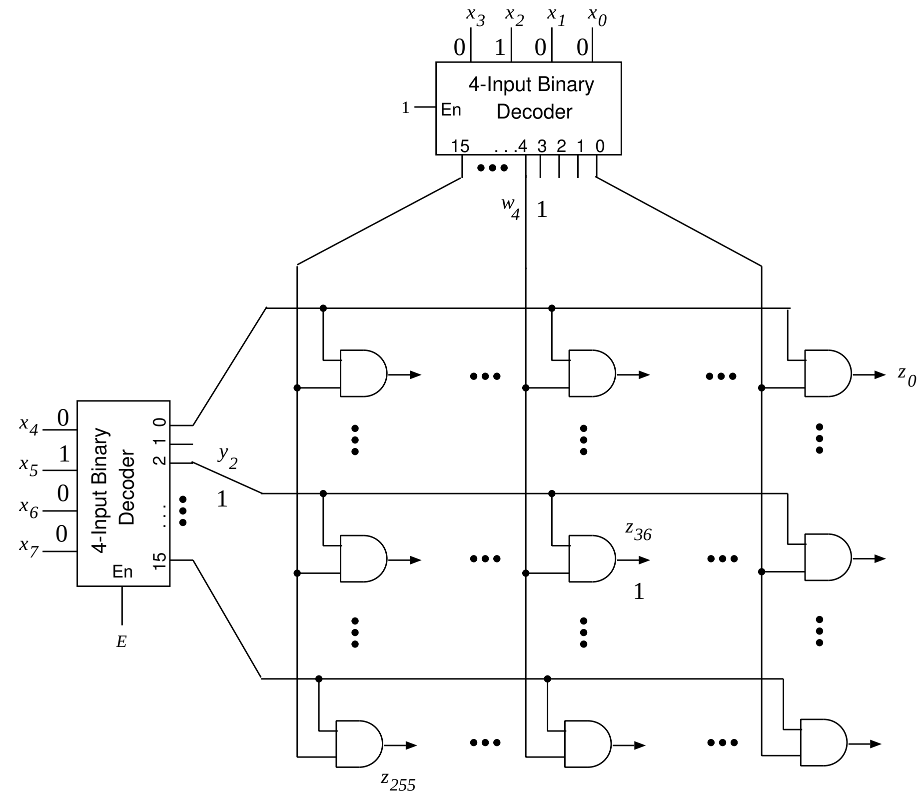
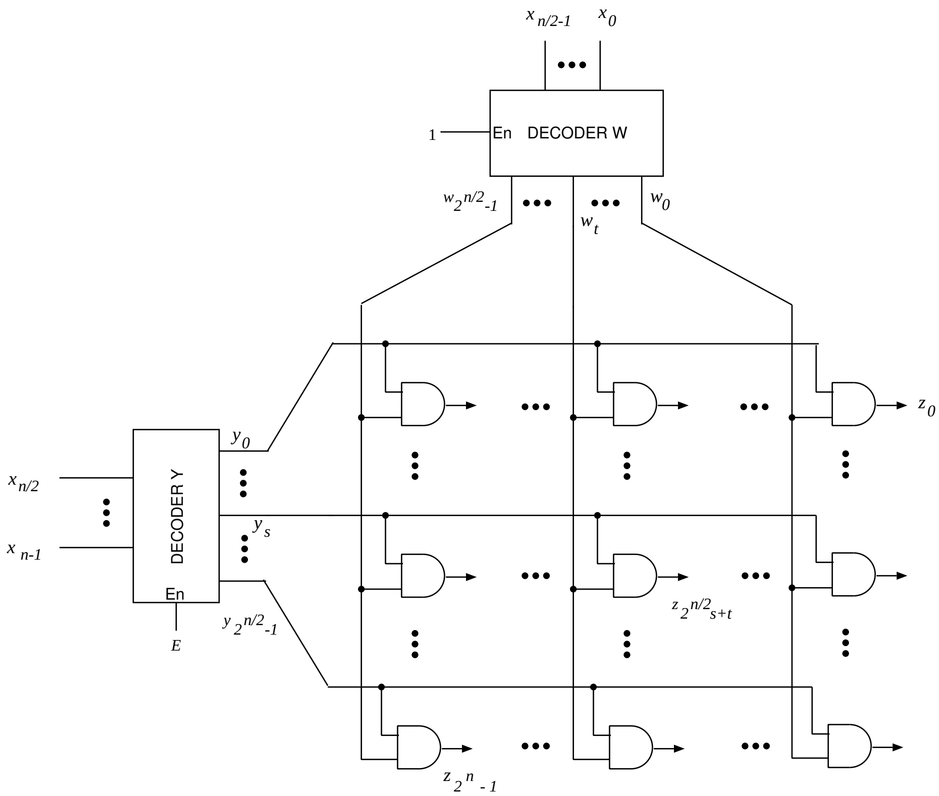


Figure 9.4: RANDOM ACCESS MEMORY (RAM): a) MODULE; b) ADDRESSING OF BINARY CELLS.

Example: DECODER(8) - schematic



DECODER(n) - schematic



Any questions?

- Why is the design correct?
- Can you prove it?
- Can't prove it unless the design is formally defined...
- Is the design (asymptotically) optimal?
- Why partition into $n/2$ & $n/2$? Is that the best partition?

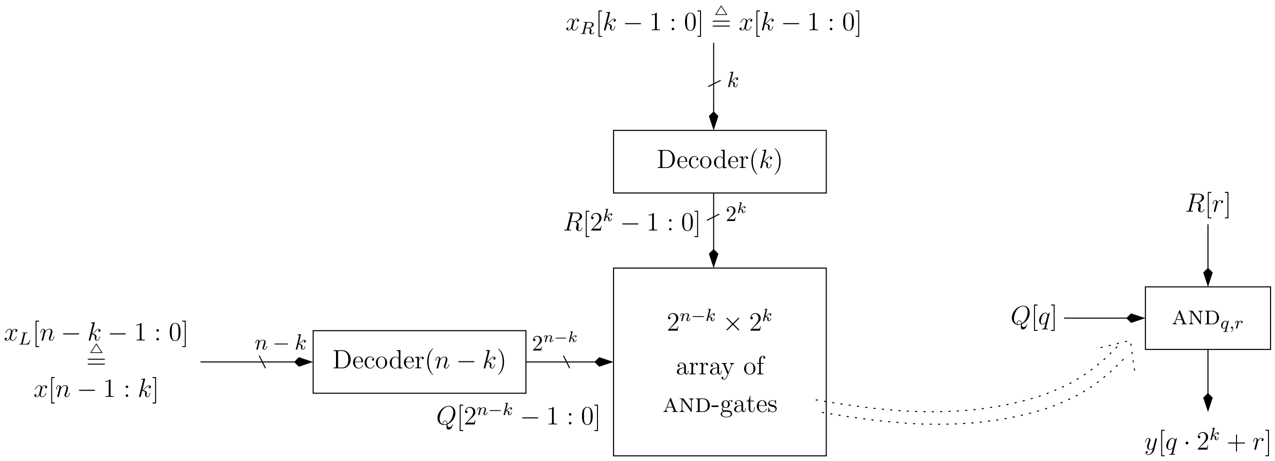
Formal description of $\text{DECODER}(n)$

Use recursion.

Basis: $\text{DECODER}(1)$: is simply one inverter where:

$$y[0] \leftarrow \text{INV}(x[0])$$
$$y[1] \leftarrow x[0].$$

Formal description of DECODER(n) - recursive step



Correctness proof

The proof is by induction.

Induction hypothesis for n: if DECODER(n) is input $x[n-1:0]$, then the output $y[2^n-1:0]$ satisfies:

$$\forall 0 \leq i < 2^n : \quad y[i] = 1 \quad \Longleftrightarrow \quad \langle x[n-1:0] \rangle = i.$$

Induction basis: $n = 1$: trivial.

Induction step: show that:

$$[\forall n' < n : \quad \text{Ind. Hyp. } (n')] \quad \Longrightarrow \quad \text{Ind. Hyp. } (n).$$

induction step - cont.

- Fix $i \in [2^n - 1 : 0]$.
- Divide by 2^k: $\quad i = q \cdot 2^k + r$.
- $q = \langle x_L[n-k-1:0] \rangle$ & $r = \langle x_R[k-1:0] \rangle$.
- Ind. Hyp. $(k) \Rightarrow$

$$R[r] = 1 \iff \langle x_R[k-1:0] \rangle = r.$$

- Ind. Hyp. $(n-k) \Rightarrow$

$$Q[q] = 1 \iff \langle x_L[n-k-1:0] \rangle = q$$

- $\Longrightarrow$

$$\begin{aligned} y[i] = 1 &\iff R[r] = 1 \text{ and } Q[q] = 1 \\ &\iff \langle x_R[k-1:0] \rangle = r \text{ and } \langle x_L[n-k-1:0] \rangle = q. \\ &\iff \langle x[n-1:0] \rangle = i \quad \text{QED} \end{aligned}$$

Cost analysis

The cost $c(n)$ satisfies the following recurrence equation:

$$c(n) = \begin{cases} c(\text{INV}) & \text{if n=1} \\ c(k) + c(n-k) + 2^n \cdot c(\text{AND}) & \text{otherwise.} \end{cases}$$

It follows that

$$c(n) = c(k) + c(n-k) + \Theta(2^n)$$

- Obviously $c(n) = \Omega(2^n)$.
- Next slide: for every choice of k, $c(n) = O(2^n)$.
- See lecture notes for tight analysis with $k = n/2$.

Cost analysis - cont.

Claim: $c(n) = O(2^n)$.

Proof: Set $\beta = c(\text{AND})$, and
$\alpha = \max\{3\beta, c(1)/2, c(2)/4, c(3)/8\}$.
We prove by induction that $c(n) \leq \alpha \cdot 2^n$. Induction basis follows for $n \leq 3$ by definition of α.

Induction step for $n \geq 4$:

$$\begin{aligned} c(n) &= c(k) + c(n-k) + \beta \cdot 2^n \\ &\leq \alpha \cdot 2^k + \alpha \cdot 2^{n-k} + \beta \cdot 2^n \\ &\leq \alpha \cdot 2^{n-1} + \alpha \cdot 2 + \beta \cdot 2^n \\ &= 2^n \left(\frac{\alpha}{2} + \beta + \frac{2\alpha}{2^n} \right). \end{aligned}$$

But,
$\beta \leq \alpha/3$, and
$n \geq 4 \Rightarrow \frac{2}{2^n} \leq \frac{1}{6}$.
Hence $c(n) \leq \alpha \cdot 2^n$.

Delay analysis

The delay of $\text{DECODER}(n)$ satisfies the following recurrence equation:

$$d(n) = \begin{cases} d(\text{INV}) & \text{if n=1} \\ \max\{d(k), d(n-k)\} + d(\text{AND}) & \text{otherwise.} \end{cases}$$

Set $k = n/2$, and it follows that $d(n) = \Theta(\log n)$.

Question: Prove that $\text{DECODER}(n)$ is asymptotically optimal with respect to cost and delay.

Weight of binary strings

Def: The weight of a binary string is defined by

$$\textit{wt}(a[n-1:0]) \triangleq |\{i : a[i] \neq 0\}|.$$

Note that

$$\textit{wt}(a[n-1:0]) = \sum_{i=1}^{n-1} a[i].$$

example:

$$\textit{wt}(01101) = 3.$$

Encoders - Definitions

We define the encoder partial function as follows.
Def: The function

$$\text{ENCODER}_n : \{\vec{y} \in \{0,1\}^{2^n} : \textit{wt}(\vec{y}) = 1\} \rightarrow \{0,1\}^n$$

is defined as follows:

$$\textit{wt}(y) = 1 \Longrightarrow y[\langle \text{ENCODER}_n(\vec{y})\rangle] = 1.$$

Examples:

$$\text{ENCODER}_2(0100) = 10$$
$$\text{ENCODER}_2(0101) = \text{undefined}.$$

Encoder - Defs. - cont.

ENCODER(n) - a comb. circuit with 2^n inputs and n outputs that implements the Boolean function ENCODER$_n$.

- Functionality is not defined for all binary strings.
- encoder $=$ decoder^{-1}.
- ENCODER(n) - an encoder with input length 2^n.

Encoders - Defs. - cont.

ENCODER(n) can be also specified as follows:

Input: $y[2^n - 1 : 0] \in \{0,1\}^{2^n}$.

Output: $x[n-1:0] \in \{0,1\}^n$.

Functionality:

$$\textit{wt}(\vec{y}) = 1 \implies y[\langle \vec{x} \rangle] = 1.$$

If $\textit{wt}(\vec{y}) \neq 1$, then the output $\vec{x}$ is arbitrary.

examples: Consider an encoder ENCODER(3).
- on input 00100000, the output equals 101.

- on input 01101011, the output is not specified.

Encoder - implementation

Plan:

- Design a simple encoder $\text{ENCODER}'(n)$.
- Prove correctness of $\text{ENCODER}'(n)$.
- Improve $\text{ENCODER}'(n)$ to obtain $\text{ENCODER}^*(n)$.
- Prove functional equivalence

$$\text{ENCODER}^*(n) \equiv \text{ENCODER}'(n),$$

  and hence, correctness of $\text{ENCODER}^*(n)$.
- Prove asymptotic optimality of $\text{ENCODER}^*(n)$.

Example: $\text{ENCODER}(3)$ with enable signal

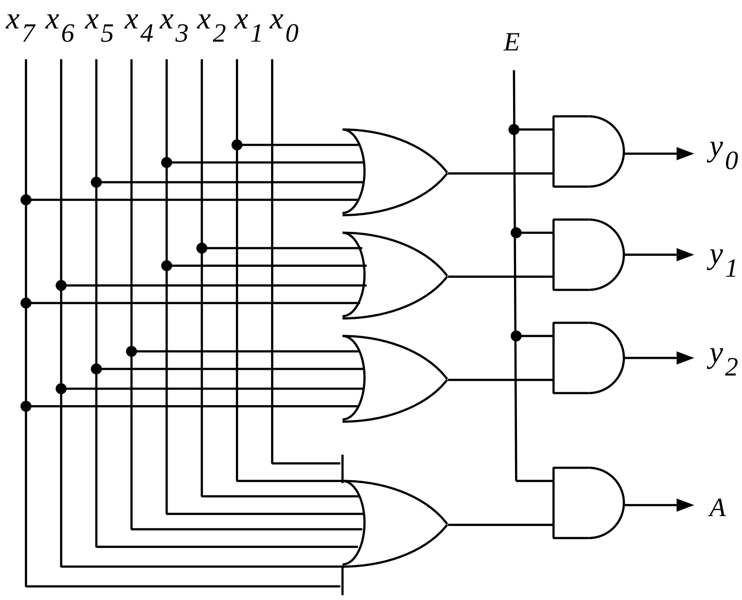


Figure 9.13: IMPLEMENTATION OF AN 8-INPUT BINARY ENCODER.

from: Introduction to Digital Systems, M.D. Ercegovac, T. Lang, and J.H. Moreno, Wiley and Sons, 1998.

ENCODER$'(n)$

Use recursion.

Basis: ENCODER(1): is simply $x[0] \leftarrow y[1]$ (zero cost, zero delay).

Recursion step: Partition input $y[2^n - 1 : 0]$

$$y_L[2^{n-1} - 1 : 0] = y[2^n - 1 : 2^{n-1}]$$
$$y_R[2^{n-1} - 1 : 0] = y[2^{n-1} - 1 : 0].$$

Apply ENCODER$'(n-1)$ to $\vec{y}_L$ and to $\vec{y}_R$.

Problem: if $\vec{y}_L = 0^{2^{n-1}}$, then output of ENCODER$'(n-1)$ is arbitrary, and design can't be correct...
$\Rightarrow$ Must specify output for an all-zeros input.

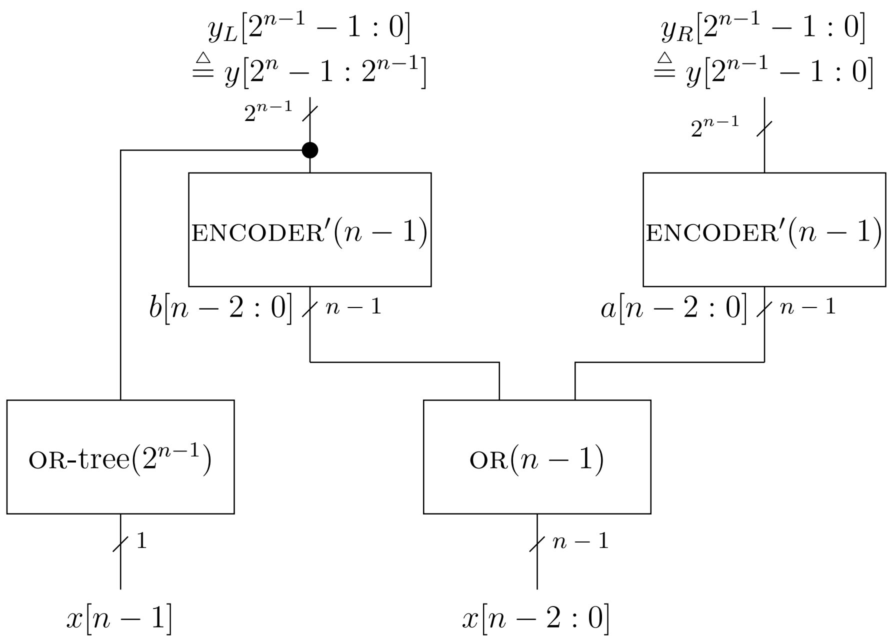
Define ENCODER$_n(0^{2^n})$

We augment the definition of the ENCODER$_n$ function so that its range also includes the all zeros string 0^{2^n}. We define

$$\text{ENCODER}_n(0^{2^n}) \triangleq 0^n.$$

Note: In ENCODER$'(1)$: $x[0] \leftarrow y[1]$, so if input is 00 then output is 0. Hence, ENCODER$'(1)$ also meets this new condition, and the induction basis of the correctness proof holds.

ENCODER$'(n)$ - recursive step

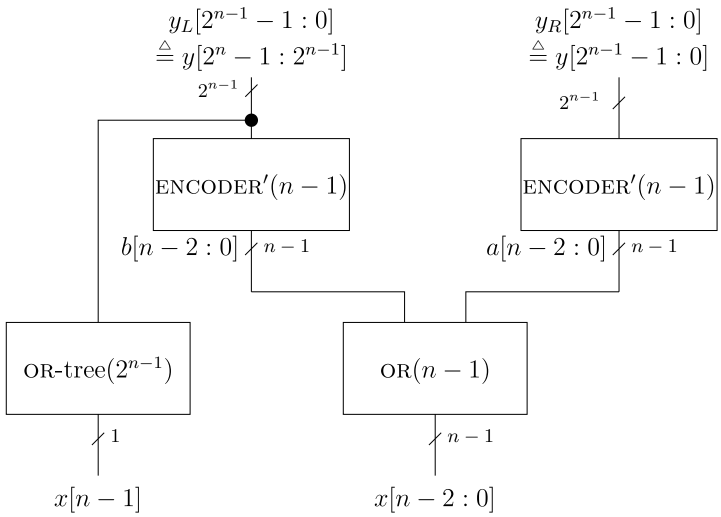


Correctness of ENCODER$'(n)$ - induction step

Three cases, depending on which "half" of $\vec{y}$ contains a 1.

(1) $wt(\vec{y}_L) = 0$ & $wt(\vec{y}_R) = 1$:

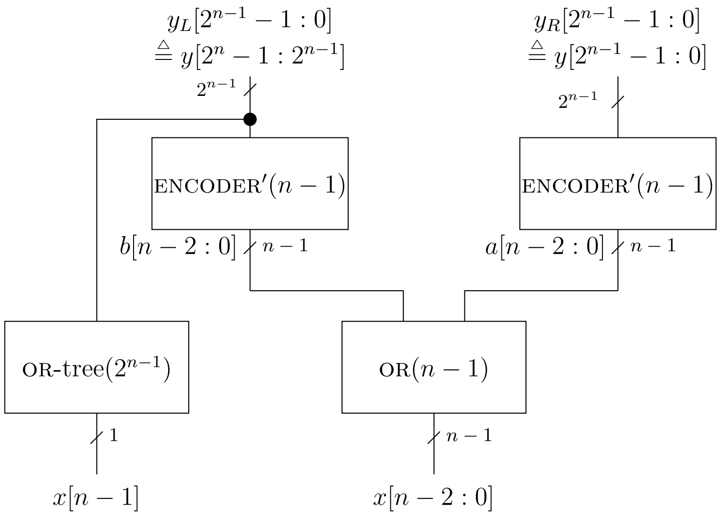
- Ind. Hyp. $\Rightarrow$
  $\vec{b} = 0^{n-1}$ & $y_R[\langle \vec{a} \rangle] = 1$.
- $\Rightarrow$ desired output is
  $\vec{x} = 0 \cdot \vec{a}$.
- actual output is
  $\vec{x} = \text{OR-tree}(\vec{y}_L) \cdot \text{OR}(\vec{b}, \vec{a})$
  $= 0 \cdot \vec{a}$.



Correctness of ENCODER$'(n)$ - induction step - cont.

(2) *wt*$(\vec{y}_L) = 1$ & *wt*$(\vec{y}_R) = 0$:

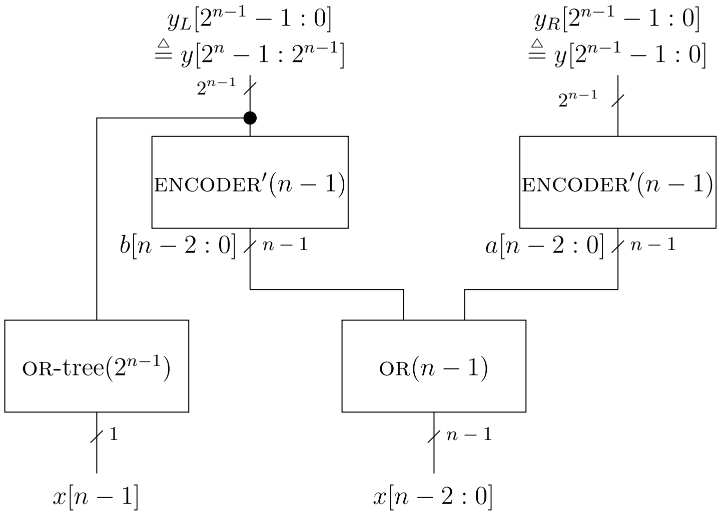
- Ind. Hyp. $\Rightarrow$ $y_L[\langle \vec{b} \rangle] = 1$ & $\vec{a} = 0^{n-1}$.
- $\Rightarrow$ desired output is $\vec{x} = 1 \cdot \vec{b}$.
- actual output is
  $$\vec{x} = \text{OR-tree}(\vec{y}_L) \cdot \text{OR}(\vec{b}, \vec{a})$$
  $$= 1 \cdot \vec{b}.$$



Correctness of ENCODER$'(n)$ - induction step - cont.

(3) *wt*$(\vec{y}_L) = 0$ & *wt*$(\vec{y}_R) = 0$:

- desired output is $\vec{x} = 0^n$.
- Ind. Hyp. $\Rightarrow$ $\vec{b} = 0^{n-1}$ & $\vec{a} = 0^{n-1}$.
- actual output is
  $$\vec{x} = \text{OR-tree}(\vec{y}_L) \cdot \text{OR}(\vec{b}, \vec{a})$$
  $$= 0 \cdot 0^{n-1}.$$



Delay analysis of $\text{ENCODER}'(n)$

Let $d(n)$ denote $t_{pd}(\text{ENCODER}'(n))$.

$$d(n) = \begin{cases} 0 & \text{if n=1} \\ \max\{(n-1) \cdot t_{pd}(\text{OR}), d(n-1) + t_{pd}(\text{OR})\} & \text{otherwise.} \end{cases}$$

Guess the solution: $d(n) = (n-1) \cdot t_{pd}(\text{OR})$.

Cost analysis of $\text{ENCODER}'(n)$

Let $c(n)$ denote $c(\text{ENCODER}'(n))$.

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

Substitute $C(2^n) = c(n)$, and re-write for $N = 2^n$:

$$C(N) = 2 \cdot C(N/2) + \Theta(N).$$

$$\Rightarrow \quad C(N) = \Theta(N \log N)$$

$$\Rightarrow \quad c(\text{ENCODER}'(n)) = \Theta(n \cdot 2^n).$$

Sanity test

$d(\text{ENCODER}'(n)) = \Theta(n)$ & $c(\text{ENCODER}'(n)) = \Theta(n \cdot 2^n)$.

But we can compute each $x[i]$ using a separate OR-tree:

$$x[i] = \text{OR}(\{y[j] : \textit{bin}_n(j)[i] = 1\}).$$

Cost of each tree is $O(2^n)$ and delay is $O(n)$.

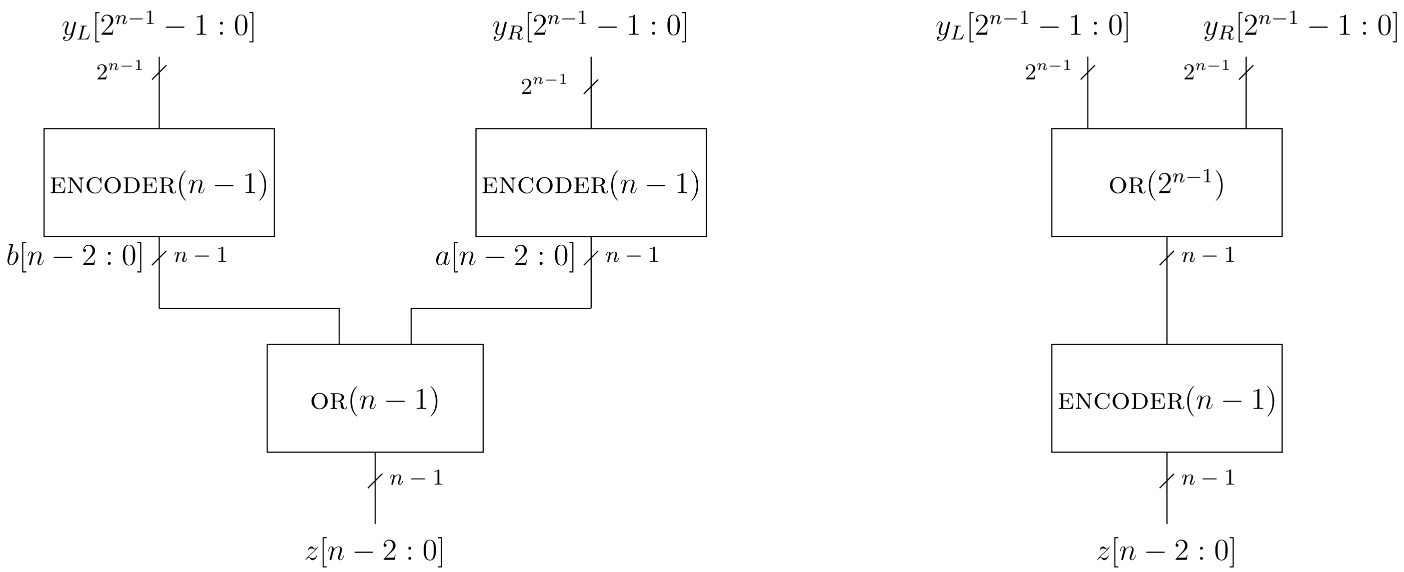
$\Rightarrow$ a trivial design with delay $O(n)$ and cost $O(n \cdot 2^n)$.

We conclude that $\text{ENCODER}'(n)$ is not a good design... Can we fix it?!

Commuting bitwise-OR and ENCODER_{n-1}

claim: If $\textit{wt}(y[2^n - 1 : 0]) \leq 1$, then

$$\text{OR}(\text{ENCODER}_{n-1}(\vec{y}_L), \text{ENCODER}_{n-1}(\vec{y}_R)) = \text{ENCODER}_{n-1}(\text{OR}(\vec{y}_L, \vec{y}_R)).$$



Proof: $\text{OR}(E_{n-1}(\vec{y}_L), E_{n-1}(\vec{y}_R)) = E_{n-1}(\text{OR}(\vec{y}_L, \vec{y}_R))$

- Trivial: $\vec{y} = 0^{2^n}$.
- *wt*$(\vec{y}_L) = 0$ & *wt*$(\vec{y}_R) = 1$:
  - Assume that $y_R[i] = 1$.
  - $\Rightarrow$

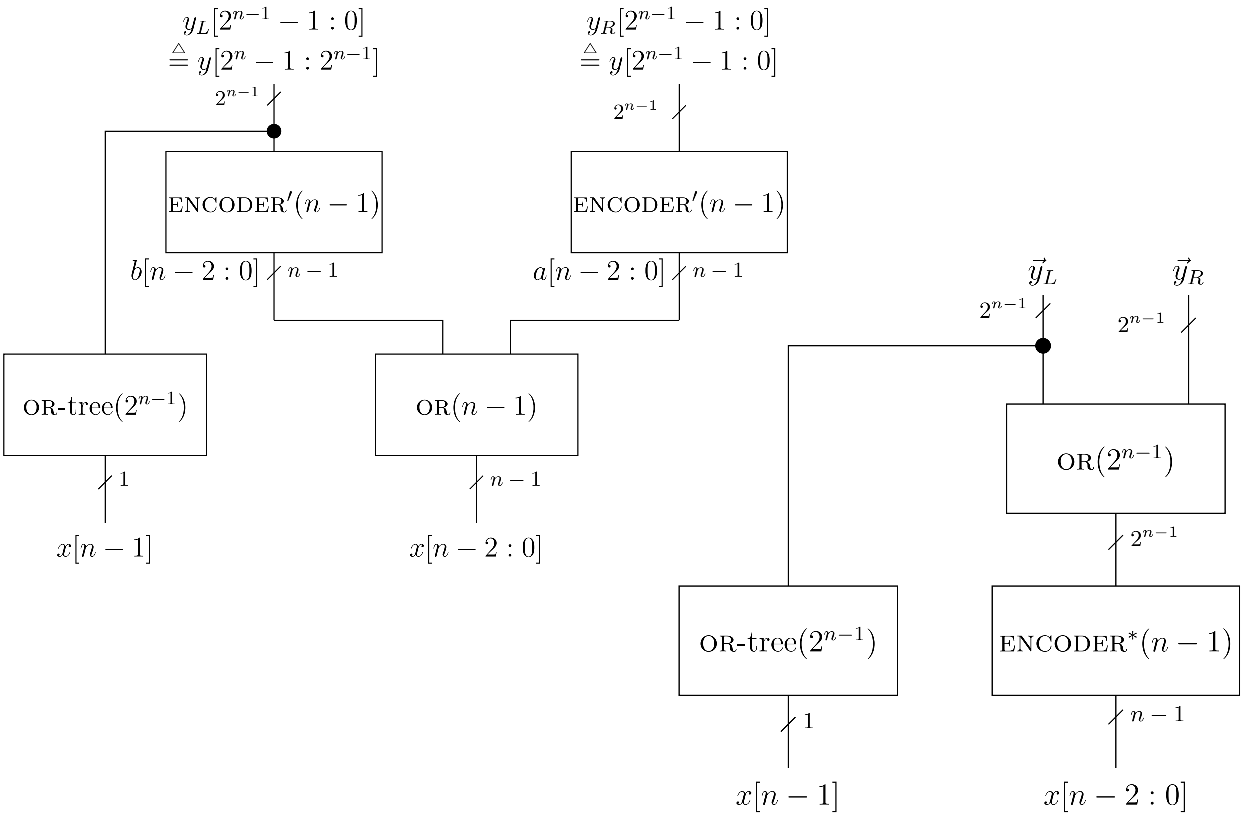
$$\begin{aligned} E_{n-1}(\text{OR}(\vec{y}_L, \vec{y}_R)) &= E_{n-1}(\text{OR}(0^{2^{n-1}}, \vec{y}_R)) \\ &= E_{n-1}(\vec{y}_R). \end{aligned}$$

  - However,

$$\begin{aligned} \text{OR}(E_{n-1}(\vec{y}_L), E_{n-1}(\vec{y}_R)) &= \text{OR}(E_{n-1}(0^{2^{n-1}}), E_{n-1}(\vec{y}_R)) \\ &= \text{OR}(0^{n-1}, E_{n-1}(\vec{y}_R)) \\ &= E_{n-1}(\vec{y}_R), \end{aligned}$$

- (other case is analogous) QED

$\text{ENCODER}'(n) \longrightarrow \text{ENCODER}^*(n)$



Correctness of $\text{ENCODER}^*(n)$

- No need to prove from "the beginning" (although not a hard task).
- Claim proves that $\text{ENCODER}'(n) \equiv \text{ENCODER}^*(n)$.
- We already proved that $\text{ENCODER}'(n)$ is correct.
- $\Rightarrow$ $\text{ENCODER}^*(n)$ is correct!
- Very useful method for hardware design:
  - start with a naive design (e.g. naive divide & conquer)
  - manipulate design (improve but preserve functionality)

Cost analysis of $\text{ENCODER}^*(n)$

$$c(\text{ENCODER}^*(n)) = \begin{cases} 0 & \text{if n=1} \\ c(\text{ENCODER}^*(n-1)) + 2^n \cdot c(\text{OR}) & \text{otherwise.} \end{cases}$$

We expand this recurrence to obtain:

$$\begin{aligned} c(\text{ENCODER}^*(n)) &= c(\text{ENCODER}^*(n-1)) + 2^n \cdot c(\text{OR}) \\ &= (2^n + 2^{n-1} + \ldots + 4) \cdot c(\text{OR}) \\ &= (2 \cdot 2^n - 3) \cdot c(\text{OR})) \\ &= \Theta(2^n). \end{aligned}$$

Delay analysis of $\textsc{encoder}^*(n)$

The delay of $\textsc{encoder}^*(n)$ satisfies the following recurrence equation:

$$d(\textsc{encoder}^*(n)) = \begin{cases} 0 & \text{if n=1} \\ \max\{d(\textsc{or}\text{-tree}(2^{n-1}), & \\ \quad d(\textsc{encoder}^*(n-1) + d(\textsc{or}))\} & \text{otherwise.} \end{cases}$$

Since $d(\textsc{or}\text{-tree}(2^{n-1}) = (n-1) \cdot d(\textsc{or})$, it follows that

$$d(\textsc{encoder}^*(n)) = (n-1) \cdot d(\textsc{or}) = \Theta(n).$$

Optimality: Prove that the cost and delay of $\textsc{encoder}^*(n)$ are asymptotically optimal.

Summary

- vector notation for schematics and binary strings.
- review of binary representation.
- Decoder & Encoder - design, correctness, optimality.
- Techniques:
  - Divide & Conquer
  - Extend specification to make problem easier
  - Evolution - improve naive and correct design while preserving functionality.