מבנה מחשבים

תרגול מספר 5

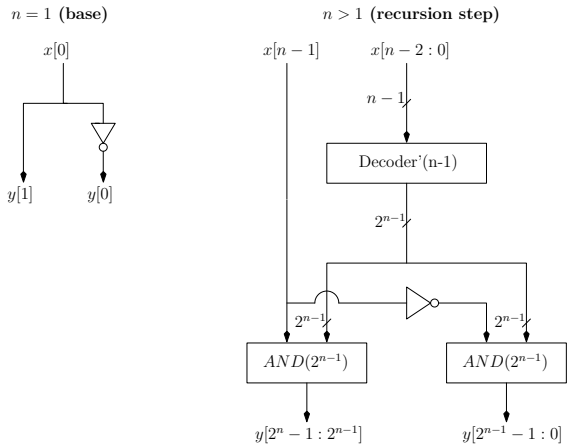
Brute force decoder(n)

- **Input**: A string $\{0,1\}^n$.
- **Output**: A string $\{0,1\}^{2^n}$.
- **Functionality**: For every string s, returns 1 in the bit which is the binary representation of s.
- **Implementation**:
  - Construct a device with n inputs and 1 output, that returns 1 when its input is the number k.
  - The device is implemented using up to n NOT gates, and one AND gate of fan-in n.

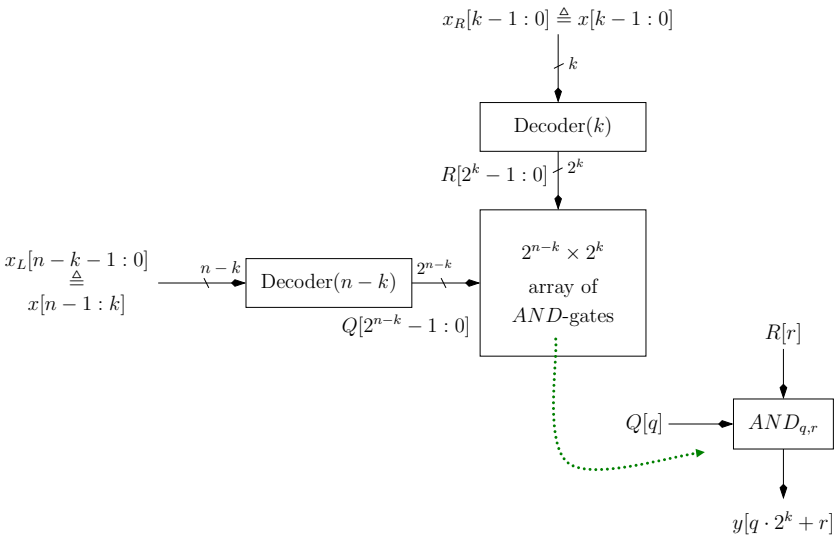
Brute force decoder(n) (cont.)

- **Implementation (cont.)**:
  - For every output bit k, connect the device that implements k.
- **Correctness**: Trivial, it is the implementation of the required boolean function.
- **Size**: $O(n2^n)$.
- **Delay**: O(1), if we use and gates with non-restricted fan-in. If we need to implement with and gates with fan-in = 2, requires O(log n).

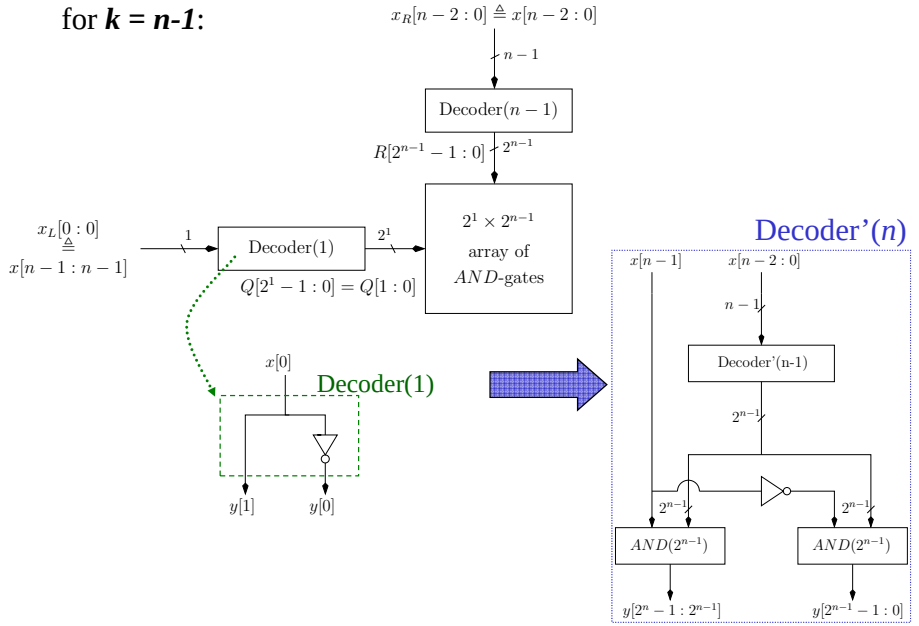
Decoder'(*n*)



Decoder(*n*) design shown and proven in class:



for ***k = n-1***:



Decoder'(*n*) is a private case of Decoder(*n*) design. (*k = n-1* or *k = 1*)

⟹ Decoder'(*n*) is a correct implementation of a decoder.

Cost analysis:

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

Solving the recurrence:

$n > 1$:

$$\begin{aligned} c(n) &= c(n-1) + 2^n \cdot c(AND) + c(INV) \\ &= c(1) + \left(2^n + 2^{n-1} + \ldots + 2^2\right) \cdot c(AND) + (n-1) \cdot c(INV) \\ &= n \cdot c(INV) + 2^2 \frac{2^{n-1}-1}{2-1} \cdot c(AND) \\ &= n \cdot c(INV) + \left(2^{n+1} - 4\right) \cdot c(AND) \\ &= \Theta\left(2^n\right) \end{aligned}$$

asymptotics

Delay analysis:

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

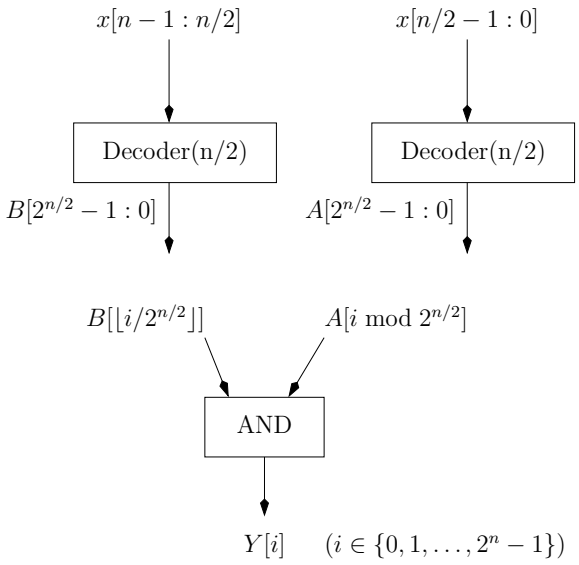
Solving the recurrence:

$n > 1$:

$$\begin{aligned} d(n) &= \max\{d(INV), \quad d(n-1)\} + d(AND) \\ &= d(n-1) + d(AND) \\ &= d(1) + (n-1) d(AND) \\ &= d(INV) + (n-1) d(AND) \\ &= \Theta(n) \end{aligned}$$

Linear Delay!!!

Decoder(n) for k = n/2



Consider the top two recursion steps:

each AND-gate in the "AND-gates array" of the second recursion step, i.e. Decoder(*n/2*), feeds $2^{n/2}$ AND-gates in the "AND-gates array" of the primary recursion step, i.e. Decoder(*n*).

$$\Rightarrow \quad \text{Maximum fanout of a net} = 2^{\frac{n}{2}}$$

$$c(n) = \begin{cases} c(INV) & \text{if } n = 1 \\ 2 \cdot c\left(\frac{n}{2}\right) + 2^n \cdot c(AND) & \text{otherwise} \end{cases} \qquad = \Theta\left(2^n\right)$$

$$d(n) = \begin{cases} d(INV) & \text{if } n = 1 \\ d\left(\frac{n}{2}\right) + d(AND) & \text{otherwise} \end{cases} \qquad = \Theta(\log n)$$

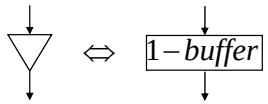
Cost

n \ Design	Decoder(n) k=n/2	Decoder'(n)
1	1	1
2	10	10
4	52	60
8	616	1024
16	132304	262152

Delay

n \ Design	Decoder(n) k=n/2	Decoder'(n)
1	1	1
2	3	3
4	5	7
8	7	15
16	9	31

Buffers



$$c(1-buffer) = 1$$
$$d(1-buffer) = 1$$

$fanout \le 2$

A balanced tree structure minimizes the delay and cost.

The tree is a binary tree due to fan-out limitations.

Notice: * The leaves of the tree feed the gate outputs.

* The root of the tree is fed by the gate input.

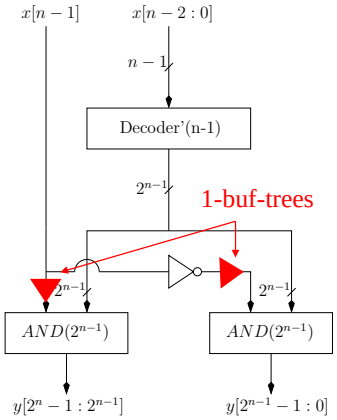
* The first branching is free → no need for a "root" gate.

$$\Rightarrow \quad d(n) = \left(\lceil \log_2 n \rceil - 1\right) \cdot d(1-buffer)$$

$$c(n) = (n-2) \cdot c(1-buffer)$$

Fanout limitation effect on design

$fanout \le 2$



Cost analysis:

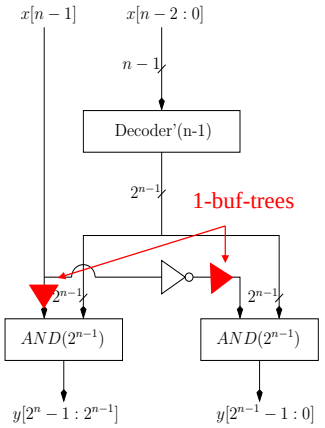
$n=1:$

$$c(1)=c(INV)$$

$n>1:$

$$c(n)=c(n-1)+2^n\cdot c(AND)+c(INV)$$
$$+2\cdot c\left(1-buf-tree\left(2^{n-1}\right)\right)+c(1-buf)$$

$$=c(n-1)+2^n\cdot c(AND)+c(INV)$$
$$+\left(2^n-3\right)\cdot c(1-buf)$$



Delay analysis:

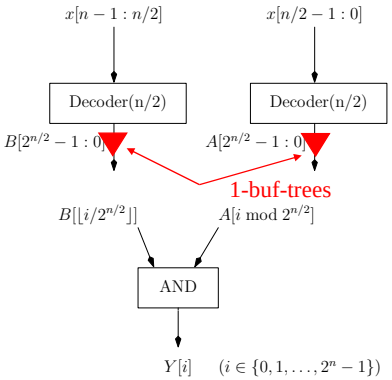
$n=1:$

$$d(1)=d(INV)$$

$n>1:$

$$d(n)=\max\left\{\begin{matrix} d(INV)+d\left(1-buf-tree\left(2^{n-1}\right)\right) \\ d(1-buf)+d\left(1-buf-tree\left(2^{n-1}\right)\right) \\ d(n-1)\end{matrix}\right\}$$
$$+d(AND)$$

$$=\max\left\{\begin{matrix} d(INV)+(n-1)\cdot d(1-buf) \\ n\cdot d(1-buf) \\ d(n-1)\end{matrix}\right\}$$
$$+d(AND)$$



Cost analysis:

$n=1:$ $\quad c(1)=c(INV)$

$n>1:$ $\quad c(n)=2\cdot c\left(\frac{n}{2}\right)+2^n\cdot c(AND)$
$$+2\cdot 2^{\frac{n}{2}}\cdot c\left(1-buf-tree\left(2^{\frac{n}{2}}\right)\right)$$
$$=2\cdot c\left(\frac{n}{2}\right)+2^n\cdot c(AND)$$
$$+2^{\frac{n}{2}+1}\left(2^{\frac{n}{2}}-2\right)\cdot c(1-buf)$$

Delay analysis:

$n=1:$ $\quad d(1)=d(INV)$

$n>1:$ $\quad d(n)=d\left(\frac{n}{2}\right)+d\left(1-buf-tree\left(2^{\frac{n}{2}}\right)\right)+d(AND)$

$$=d\left(\frac{n}{2}\right)+\left(\frac{n}{2}-1\right)d(1-buf)+d(AND)$$

Cost Decoder(n):

n=1:	1
n=2:	10
n=4:	52
n=8:	616
n=16:	132304
n=32:	8.5902e+009
n=64:	3.68935e+019
n=128:	6.80565e+038

Cost Decoder(n), Fanout<=2:

n=1:	1
n=2:	10
n=4:	68
n=8:	1096
n=16:	263312
n=32:	1.71801e+010
n=64:	7.3787e+019
n=128:	1.36113e+039

Cost Decoder'(n):

n=1:	1
n=2:	10
n=4:	60
n=8:	1024
n=16:	262152
n=32:	1.71799e+010
n=64:	7.3787e+019
n=128:	1.36113e+039

Cost Decoder'(n), Fanout<=2:

n=1:	1
n=2:	11
n=4:	79
n=8:	1511
n=16:	393175
n=32:	2.57698e+010
n=64:	1.1068e+020
n=128:	2.04169e+039

Delay Decoder(n):

n=1:	1
n=2:	3
n=4:	5
n=8:	7
n=16:	9
n=32:	11
n=64:	13
n=128:	15

Delay Decoder(n), Fanout<=2:

n=1:	1
n=2:	3
n=4:	6
n=8:	11
n=16:	20
n=32:	37
n=64:	70
n=128:	135

Delay Decoder‘(n):

n=1:	1
n=2:	3
n=4:	7
n=8:	15
n=16:	31
n=32:	63
n=64:	127
n=128:	255

Delay Decoder‘(n), Fanout<=2:

n=1:	1
n=2:	4
n=4:	8
n=8:	16
n=16:	32
n=32:	64
n=64:	128
n=128:	256

Brute force encoder(n)

- **Input**: A string $\{0,1\}^{2^n}$.
- **Output**: A string $\{0,1\}^n$.
- **Functionality**: For every string s with weight 1, returns the binary representation of s.
- **Implementation**:
  - Connect bits $1,3,\ldots 2^n-1$ with an OR gate of size 2^{n-1}, this gate sets bit 1 of the output.
  - Connect bits 2,3,6,7…with an OR gate of size 2^{n-1}, this gate sets bit 2 of the output.
  - And so on for each power of 2.

Brute force encoder(n) (cont.)

- **Correctness**: Trivial, bit j of the output is 1 if and only if any of the inputs in the set S_j is 1. The set S_j is the set of all input bits whose index is such that 2^j is 1 in its binary representation.
- **Size**: $O(n2^n)$.
- **Delay**: O(1), if we use OR gates with non-restricted fan-in. If we need to implement with and gates with fan-in = 2, requires O(n).