Digital Logic Systems

Recitation 8: Lower Bounds on cost and delay, Multiplexers, Decoders, Encoders

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

May 6, 2019

When does a function depend on an input?

Definition

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ depends on its i^{th} input if

$$f_{\restriction x_i = 0} \neq f_{\restriction x_i = 1}.$$

Example

Consider the Boolean function $f(\vec{x}) = \text{XOR}_2(x_1, x_2)$. The function f depends on the ith input for $i = 2$. Indeed, $f_{\restriction x_2 = 1}(x_1) = \text{NOT}(x_1)$ and $f_{\restriction x_2 = 0}(x_1) = x_1$.

The cone of a function

Definition (Cone of a Boolean function)

The cone of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is defined by

$$cone(f) \triangleq \{i : f_{\restriction x_i=0} \neq f_{\restriction x_i=1}\}.$$

Alternative Definition (Cone of a Boolean function)

Let $f : \{0,1\}^n \to \{0,1\}^k$ denote a Boolean function. Then,

$$i \in cone(f) \iff \exists \vec{v} \in \{0,1\}^n : f(v) \neq f(flip_i(\vec{v})).$$

Cone if a function f is a set of all the indices that f depends on.

Example

The cone of the Boolean function $f(\vec{x}) = \mathrm{XOR}_2(x_1, x_2)$ equals $\{1,2\}$ because XOR depends on both inputs.

Example

Example

Consider the following Boolean function:

$$f(\vec{x}) = \begin{cases} 0 & \text{if } \sum_i x_i < 3 \\ 1 & \text{otherwise.} \end{cases}$$

Suppose that one reveals the input bits one by one. As soon as 3 ones are revealed, one can determine the value of $f(\vec{x})$. Nevertheless, the function $f(\vec{x})$ depends on all its inputs, and hence, $cone(f) = \{1, \ldots, n\}$.

Lower Bound Theorems

Theorem

Let C denote a combinational circuit that implements a Boolean function $f : \{0,1\}^n \to \{0,1\}$. If the fan-in of every gate in C is at most 2, then

$$c(C) \geq |\text{cone}(f)| - 1.$$

Theorem

Let $C = (G, \pi)$ denote a combinational circuit that implements a non-constant Boolean function $f : \{0,1\}^n \to \{0,1\}$. If the fan-in of every gate in C is at most k, then

$$t_{pd}(C) \geq \log_k |\text{cone}(f)|.$$

The lower bounds are on the function (based on the cone of your function). Whereas the actual order of growth of the design are on your design

Definition of Decoder

Definition

A decoder with input length n:

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

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

Functionality:

$$y[i] \triangleq \begin{cases} 1 & \text{if } \langle \vec{x} \rangle = i \\ 0 & \text{otherwise.} \end{cases}$$

Number of outputs of a decoder is exponential in the number of inputs. Note also that exactly one bit of the output $\vec{y}$ is set to one. Such a representation of a number is often termed one-hot encoding or 1-out-of-k encoding.

Example

Consider a decoder $\text{DECODER}(3)$. On input $x = 101$, the output y equals 00100000.

An asymptotically optimal decoder design

Base case $\text{DECODER}(1)$:
The circuit $\text{DECODER}(1)$ is simply one inverter where:
$y[0] \leftarrow \text{INV}(x[0])$ and $y[1] \leftarrow x[0]$.
Reduction rule $\text{DECODER}(n)$:
We assume that we know how to design decoders with input length less than n, and design a decoder with input length n.

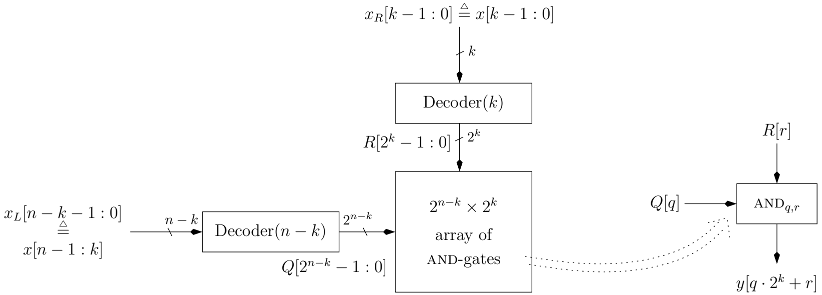


Figure: A recursive implementation of DECODER(n).

Claim (Correctness)

$$y[i] = 1 \quad \Longleftrightarrow \quad \langle x[n-1:0] \rangle = i.$$

Performance Analysis of Decoder

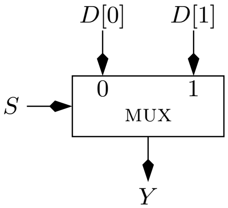
Claim

$c(n) = \Omega(2^n)$ *(regardless of the value of* k*).*
$c(n) = O(2^n)$ *if* $k = \lceil n/2 \rceil$*.*

Claim

$d(n) = \Theta(\log n)$ *if* $k = n/2$*.*

Multiplexer (MUX$2:1$)



Definition

A MUX-gate is a combinational gate that has three inputs $D[0], D[1], S$ and one output Y. The functionality is defined by

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

Note that we could have used the shorter expression $Y = D[S]$ to define the functionality of a MUX-gate.

(n:1)-MUX selects on bit out of n

Definition

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

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

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

Functionality:

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

To simplify the discussion, we will assume in this chapter that n is a power of 2, namely, $n = 2^k$.

Example

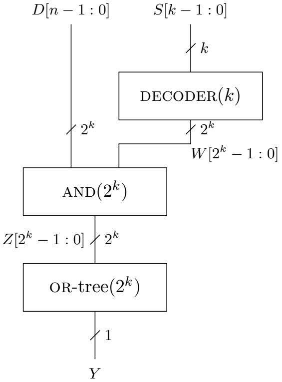
Let $n = 4$ and $D[3:0] = 0101$. If $S[1:0] = 00$, then $Y = D[0] = 1$. If $S[1:0] = 01$, then $Y = D[1] = 0$.

Implementation

We describe two implementations of (n:1)-MUX.

- Decoder based - in the recitation (modular design).
- Tree based - in the lecture (recursive).

Decoder based (n:1)-MUX



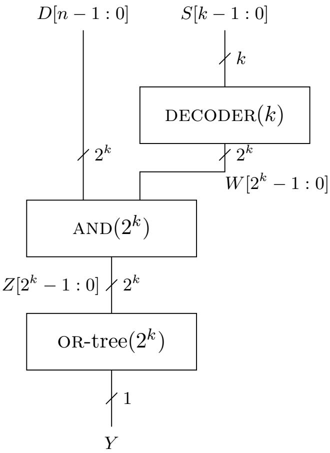
Claim

The (n:1)-MUX design is correct.

Claim

The cost of the (n:1)-MUX design is $\Theta(n)$.

Decoder based (n:1)-MUX - delay



Claim

The delay of the (n:1)-MUX design is $\Theta(\log n)$.

Encoder circuit - definition

Definition

An encoder with input length 2^n and output length n is a combinational circuit that implements the Boolean function ENCODER_n.

We denote an encoder with input length 2^n and output length n by $\text{ENCODER}(n)$. An $\text{ENCODER}(n)$ 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: If $wt(\vec{y}) = 1$, let i denote the index such that $y[i] = 1$. In this case $\vec{x}$ should satisfy $\langle \vec{x} \rangle = i$. Formally:

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

$\text{ENCODER}'(n)$ - a recursive design

For $n = 1$, is simply $x[0] \leftarrow y[1]$.
Reduction step:

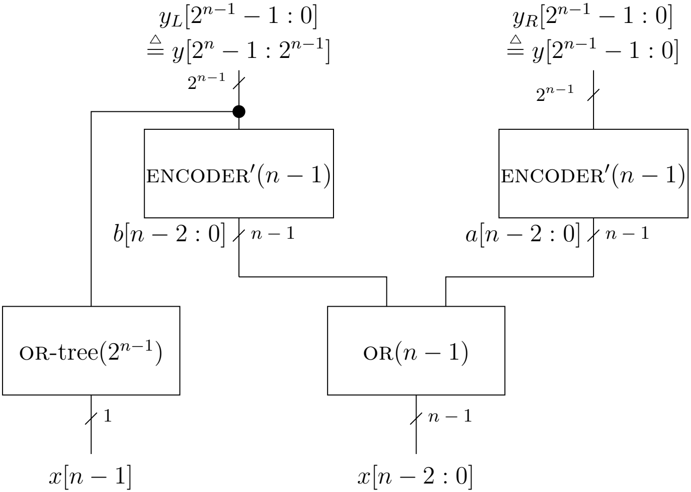
$$y_L[2^{n-1} - 1 : 0] = y[2^n - 1 : 2^{n-1}]$$
$$y_R[2^{n-1} - 1 : 0] = y[2^{n-1} - 1 : 0].$$

Use two $\text{ENCODER}'(n-1)$ with inputs $\vec{y_L}$ and $\vec{y_R}$. But,

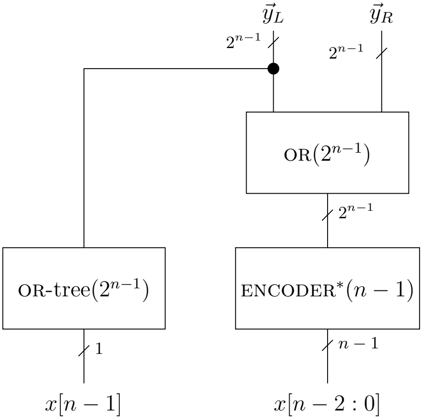
$$wt(\vec{y}) = 1 \Rightarrow (wt(\vec{y_L}) = 0) \vee (wt(\vec{y_R}) = 0).$$

What does an encoder output when input all-zeros?

Reduction step for ENCODER$'(n)$



Reduction step for ENCODER$^*(n)$



Performance Analysis of Encoder

Claim

$c(\text{ENCODER}'(n)) = \Theta(n \cdot 2^n)$.
(asymptotically) equals the cost of the brute force design...

Claim

$c(\text{ENCODER}^*(n)) = \Theta(2^n) \cdot c(\text{OR})$.

Claim

$d(\text{ENCODER}^*(n)) = n \cdot d(\text{OR})$.

Priority Encoder - $\text{PENC}(n)$

A $\text{PENC}(n)$ is a combinational circuit with input length 2^n is defined as follows.

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

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

Functionality: $v = 1 \Leftrightarrow y \neq 0^{2^n}$. Let i denote the highest index i such that $y[i] = 1$. In this case $\vec{x}$ should satisfy $\langle \vec{x} \rangle = i$. Formally:

$$\vec{y} \neq 0^{2^n} \implies y[2^n - 1 : \langle \vec{x} \rangle] = 0^{2^n - 1 - \langle \vec{x} \rangle} \circ 1.$$

In other words

Priority encoder deals with situation where **more than one input is active**. The output will encode the input index with the higher "priority".

Priority Encoder - PENC(n)

Table: The Truth Table of The Priority Encoder(3)

y7	y6	y5	y4	y3	y2	y1	y0	x2	x1	x0	v
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	d.c.	0	0	1	1
0	0	0	0	0	1	d.c.	d.c.	0	1	0	1
0	0	0	0	1	d.c.	d.c.	d.c.	0	1	1	1
0	0	0	1	d.c.	d.c.	d.c.	d.c.	1	0	0	1
0	0	1	d.c.	d.c.	d.c.	d.c.	d.c.	1	0	1	1
0	1	d.c.	d.c.	d.c.	d.c.	d.c.	d.c	1	1	0	1
1	d.c.	d.c.	d.c.	d.c.	d.c.	d.c.	d.c.	1	1	1	1

Implementation Tips

- Use “Divide and Conquer”, similar to Tree-based-(n:1)-MUX.
- Use the “v” signal to choose between the sub-encoders

Question

Design the following circuit

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

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

Functionality:

$$x[i] = OR(\{bin_n(j)[i] \mid y[j] = 1\})$$

For every $0 \leq i \leq n-1$. Where $bin_n(j)$ is a function that return the n-bit binary string that represents j using n bits. $(bin_n : \{0,1,\ldots,2^n-1\} \rightarrow \{0,1\}^n)$

Question

Design the following circuit

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

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

Functionality:

$$x[i] = OR(\{bin_n(j)[i] \mid y[j] = 1\})$$

For every $0 \leq i \leq n-1$. Where $bin_n(j)$ is a function that return the n-bit binary string that represents j using n bits. $(bin_n : \{0,1,\ldots,2^n-1\} \to \{0,1\}^n)$

Answer

An encoder. We were actually asked to implement an extended function of the encoder.