Digital Logic Design: a rigorous approach ©

Chapter 13: Decoders and Encoders

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

May 13, 2020

Book Homepage:
http://www.eng.tau.ac.il/~guy/Even-Medina

Buses

Example

An adder and a register (a memory device). The output of the adder should be stored by the register. Different name to each bit?!

Buses

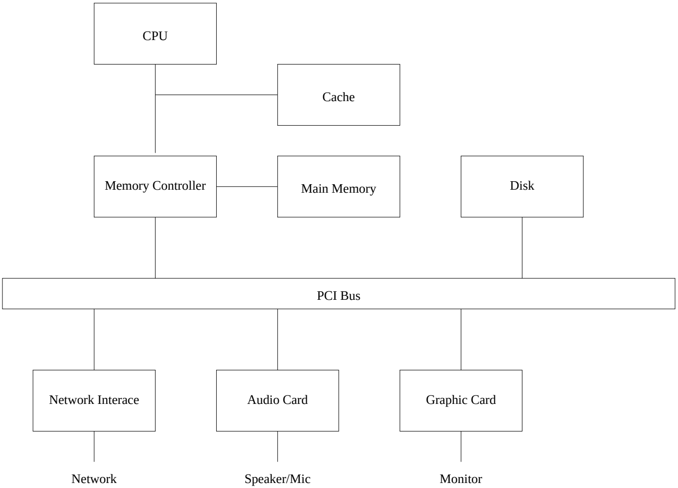
Definition

A *bus* is a set of nets that are connected to the same modules.
The *width* of a bus is the number of nets in the bus.

Buses

Example

PCI bus is data network that connects modules in a computer system.



Indexing conventions

1. Connection of terminals is done by assignment statements: The statement $b[0:3] \leftarrow a[0:3]$ means connect $a[i]$ to $b[i]$.
2. "Reversing" of indexes does not take place unless explicitly stated: $b[i:j] \leftarrow a[i:j]$ and $b[i:j] \leftarrow a[j:i]$, have the same meaning, i.e., $b[i] \leftarrow a[i], \ldots, b[j] \leftarrow a[j]$.
3. "Shifting" is done by default: $a[0:3] \leftarrow b[4:7]$, meaning that $a[0] \leftarrow b[4], a[1] \leftarrow b[5]$, etc. We refer to such an implied re-assignment of indexes as hardwired shifting.

Example - 1

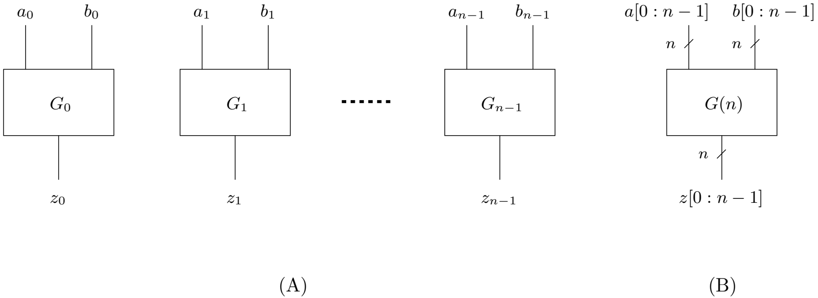


Figure: Vector notation: multiple instances of the same gate. (A) Explicit multiple instances (B) Abbreviated notation.

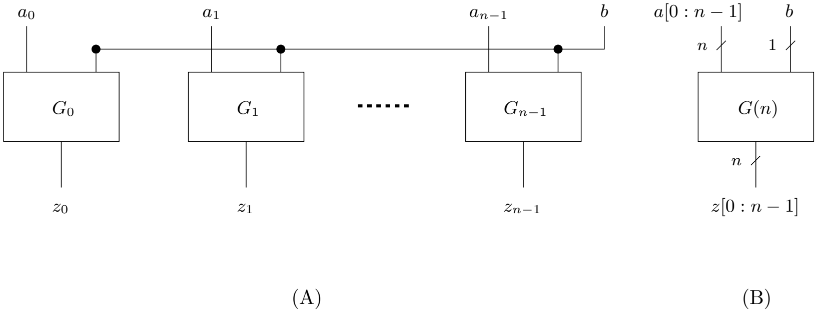


Figure: Vector notation: b feeds all the gates. (A) Explicit multiple instances (B) Abbreviated notation.

Reminder: Binary Representation

Recall that $\langle a[n-1:0]\rangle_n$ denotes the binary number represented by an n-bit vector $\vec{a}$.

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

Definition

Binary representation using n-bits is a function $bin_n : \{0,1,\ldots,2^n-1\} \to \{0,1\}^n$ that is the inverse function of $\langle\cdot\rangle$. Namely, for every $a[n-1:0] \in \{0,1\}^n$,

$$bin_n(\langle a[n-1:0]\rangle_n) = a[n-1:0].$$

Division in Binary Representation

$r = (a \bmod b)$:

$$a = q \cdot b + r, \text{ where } 0 \leq r < b.$$

Claim

Let $s = \langle x[n-1:0] \rangle_n$, and $0 \leq k \leq n-1$. Let q and r denote the quotient and remainder obtained by dividing s by 2^k. Define the binary strings $x_R[k-1:0]$ and $x_L[n-1:n-k-1]$ as follows.

$$x_R[k-1:0] \stackrel{\triangle}{=} x[k-1:0]$$
$$x_L[n-k-1:0] \stackrel{\triangle}{=} x[n-1:k].$$

Then,

$$q = \langle x_L[n-k-1:0] \rangle$$
$$r = \langle x_R[k-1:0] \rangle.$$

Multiplication

Multiplication of $A[n-1:0]$ by $B[n-1:0]$ in binary representation proceeds in two steps:

- compute all the partial products $A[i] \cdot B[j]$
- add the partial products

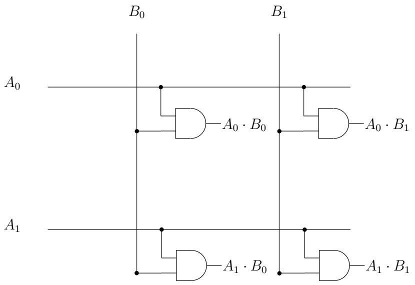
$$
\begin{array}{r}
1011 \\
\times\ 1110 \\
\hline
0000 \\
1011 \\
1011 \\
+\ 1011 \\
\hline
10011010
\end{array}
$$

Computation of Partial Products

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

Output: $C[i,j] \in \{0,1\}^{n^2-1}$ where $(0 \leq i,j \leq n-1)$

Functionality: $C[i,j] = A[i] \cdot B[i]$



We refer to such a circuit as $n \times n$ array of AND gates. Cost is n^2 and delay equals 1 (Q: What is the lower bound?).

Definition

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:

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

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.

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:

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

We denote a decoder with input length n by $\text{DECODER}(n)$.

Example

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

Application of decoders

An example of how a decoder is used is in decoding of controller instructions. Suppose that each instruction is coded by an 4-bit string. Our goal is to determine what instruction is to be executed. For this purpose, we feed the 4 bits to a DECODER(4). There are 16 outputs, exactly one of which will equal 1. This output will activate a module that should be activated in this instruction.

Brute force design

- simplest way: build a separate circuit for every output bit $y[i]$.
- The circuit for $y[i]$ is simply a product of n literals.
- Let $v \triangleq bin_n(i)$, i.e., $v[n-1:0]$ is the binary representation of the index i.
- define the minterm p_v to be $p_v \triangleq (\ell_0^v \cdot \ell_1^v \cdots \ell_{n-1}^v)$, where:

$$\ell_j^v \triangleq \begin{cases} x_j & \text{if } v_j = 1 \\ \bar{x}_j & \text{if } v_j = 0. \end{cases}$$

- define $y[\langle v \rangle] \triangleq \text{AND}_n(\ell_0^v, \ldots, \ell_{n-1}^v)$

Claim

$y[i] = 1$ *iff* $\langle x \rangle = i$.

analysis: brute force design

The brute force decoder circuit consists of:

- n inverters used to compute $\text{INV}(\vec{x})$, and
- a separate $\text{AND}(n)$-tree for every output $y[i]$.
- The delay of the brute force design is $t_{pd}(\text{INV}) + t_{pd}(\text{AND}(n)\text{-tree}) = O(\log_2 n)$.
- The cost of the brute force design is $\Theta(n \cdot 2^n)$, since we have an $\text{AND}(n)$-tree for each of the 2^n outputs.

Wasteful because, if the binary representation of i and j differ in a single bit, then the AND-trees of $y[i]$ and $y[j]$ share all but a single input. Hence the product of $n-1$ bits is computed twice.

We present a systematic way to share hardware between different outputs.

An asymptotically optimal decoder design

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

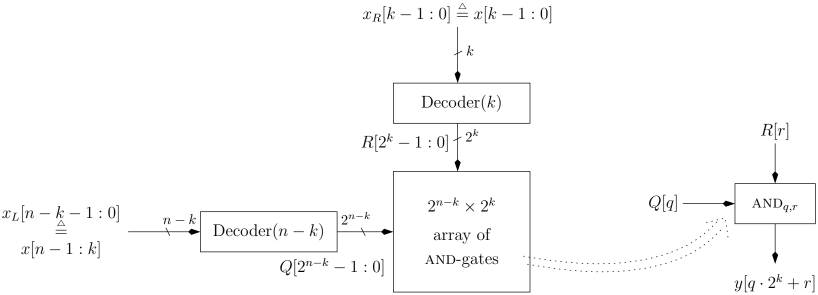


Figure: A recursive implementation of DECODER(n).

Claim (Correctness)

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

Cost analysis

We denote the cost and delay of $\text{DECODER}(n)$ by $c(n)$ and $d(n)$, respectively. 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, up to constant factors

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

Obviously, $c(n) = \Omega(2^n)$ (regardless of the value of k).

Claim

$c(n) = O(2^n)$ if $k = \lceil n/2 \rceil$.

Cost analysis (cont.)

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

Claim

$c(n) = O(2^n)$ *if* $k = \lceil n/2 \rceil$.

Proof.

$c(n) \leq 2 \cdot 2^n$ by complete induction on n.

- basis: check for $n \in \{1, 2, 3\}$.
- step:

$$\begin{aligned} c(n) &= c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + 2^n \\ &\leq 2^{1+\lceil n/2 \rceil} + 2^{1+\lfloor n/2 \rfloor} + 2^n \\ &= 2 \cdot 2^n \cdot (2^{-\lfloor n/2 \rfloor} + 2^{-\lceil n/2 \rceil} + 1/2) \end{aligned}$$

□

Delay analysis.

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

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

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

Asymptotic Optimality

Theorem

For every decoder G of input length n:

$$d(G) = \Omega(\log n)$$
$$c(G) = \Omega(2^n).$$

Proof.

1. lower bound on delay : use log delay lower bound theorem.
2. lower bound on cost? The proof is based on the following observations:
   - Computing each output bit requires at least one nontrivial gate.
   - No two output bits are identical.

□

Encoders

- An encoder implements the inverse Boolean function implemented by a decoder.
- the Boolean function implemented by a decoder is not surjective.
- the range of the Boolean function implemented by a decoder is the set of binary vectors in which exactly one bit equals 1.
- It follows that an encoder implements a partial Boolean function (i.e., a function that is not defined for every binary string).

Hamming Distance and Weight

Definition

The Hamming distance between two binary strings $u, v \in \{0,1\}^n$ is defined by

$$dist(u,v) \triangleq |\{i \mid u_i \neq v_i\}|.$$

Definition

The Hamming weight of a binary string $u \in \{0,1\}^n$ equals $dist(u, 0^n)$. Namely, the number of non-zero symbols in the string.

We denote the Hamming weight of a binary string $\vec{a}$ by $wt(\vec{a})$, namely,

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

Concatenation of strings

Recall that the concatenation of the strings a and b is denoted by $a \circ b$.

Definition

The binary string obtained by i concatenations of the string a is denoted by a^i.

Consider the following examples of string concatenation:

- If $a = 01$ and $b = 10$, then $a \circ b = 0110$.
- If $a = 1$ and $i = 5$, then $a^i = 11111$.
- If $a = 01$ and $i = 3$, then $a^i = 010101$.
- We denote the zeros string of length n by 0^n.

Definition of Encoder function

We define the encoder partial function as follows.

Definition

The function $\text{ENCODER}_n : \{\vec{y} \in \{0,1\}^{2^n} : wt(\vec{y}) = 1\} \rightarrow \{0,1\}^n$ is defined as follows: $\langle \text{ENCODER}_n(\vec{y}) \rangle$ equals the index of the bit of $y[2^n - 1 : 0]$ that equals one. Formally,

$$\text{ENCODER}_n(0^{2^n-k-1} \circ 1 \circ 0^k) = bin_n(k)$$

Examples:

1. $\text{ENCODER}_2(0001) = 00$, $\text{ENCODER}_2(0010) = 01$, $\text{ENCODER}_2(0100) = 10$, $\text{ENCODER}_2(1000) = 11$.

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:

$$\vec{x} = \text{ENCODER}_n(\vec{y}) \ .$$

Encoder - remarks

- functionality is not specified for all inputs $\vec{y}$.
- functionality is only specified for inputs whose Hamming weight equals one.
- Since an encoder is a combinational circuit, it implements a Boolean function. This means that it outputs a digital value even if $wt(y) \neq 1$. Thus, two encoders must agree only with respect to inputs whose Hamming weight equals one.
- If $\vec{y}$ is output by a decoder, then $wt(\vec{y}) = 1$, and hence an encoder implements the inverse function of a decoder.

Brute Force Implementation

Recall that $bin_n(i)[j]$ denotes the jth bit in the binary representation of i. Let A_j denote the set

$$A_j \triangleq \{i \in [0 : 2^n - 1] \mid bin_n(i)[j] = 1\}.$$

Claim

If $\mathrm{wt}(y) = 1$, *then* $x[j] = \bigvee_{i \in A_j} y[i]$.

Brute Force Implementation - cont

Claim

If $\text{wt}(y) = 1$, *then* $x[j] = \bigvee_{i \in A_j} y[i]$.

Implementing an $\textsc{Encoder}(n)$:

- For each output x_j, use a separate $\textsc{Or}$-tree whose inputs are $\{y[i] \mid i \in A_j\}$.
- Each such $\textsc{Or}$-tree has at most 2^n inputs.
- the cost of each $\textsc{Or}$-tree is $O(2^n)$.
- total cost is $O(n \cdot 2^n)$.
- The delay of each $\textsc{Or}$-tree is $O(\log 2^n) = O(n)$.

Can we do better?

- We will prove that in every combinational circuit E that implements an encoder, the cardinality of the graphical cone of the first output $X[0]$ is at least $2^n/2$.
- So for every encoder E: $c(E) = \Omega(2^n)$ and $d(E) = \Omega(n)$.
- The brute force design is not that bad. Can we reduce the cost?
- Let's try...

$\textsc{Encoder}'(n)$ - 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 $\textsc{Encoder}'(n-1)$ 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?

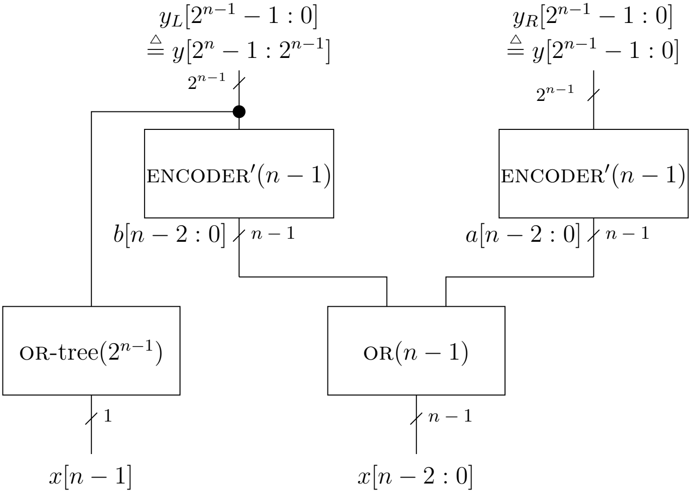
Augmenting functionality

Augment the definition of the ENCODER_n function so that its domain also includes the all-zeros string 0^{2^n}. We define

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

Note that $\text{ENCODER}'(1)$ (i.e., $x[0] \leftarrow y[1]$) also meets this new condition, so the induction basis of the correctness proof holds.

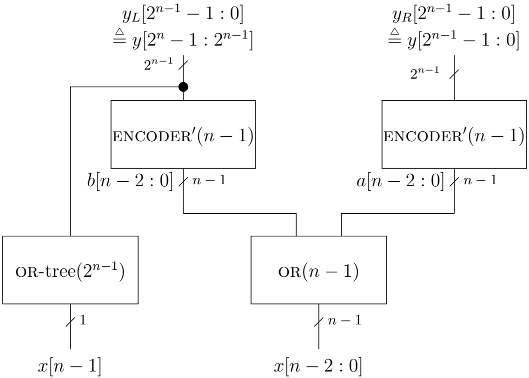
Reduction step for ENCODER$'(n)$



Correctness

Claim

The circuit $\text{ENCODER}'(n)$ *implements the Boolean function* ENCODER_n.



Cost Analysis

$$c(\textsc{Encoder}'(n)) = \begin{cases} 0 & \text{if } n = 1 \\ 2 \cdot c(\textsc{Encoder}'(n-1)) & \\ +c(\textsc{Or}\text{-tree}(2^{n-1})) & \\ +(n-1) \cdot c(\textsc{Or}) & \text{if } n > 1. \end{cases}$$

Let $c(n) \triangleq c(\textsc{Encoder}'(n))/c(\textsc{Or})$.

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

Claim

$c(n) = \Theta(n \cdot 2^n)$.

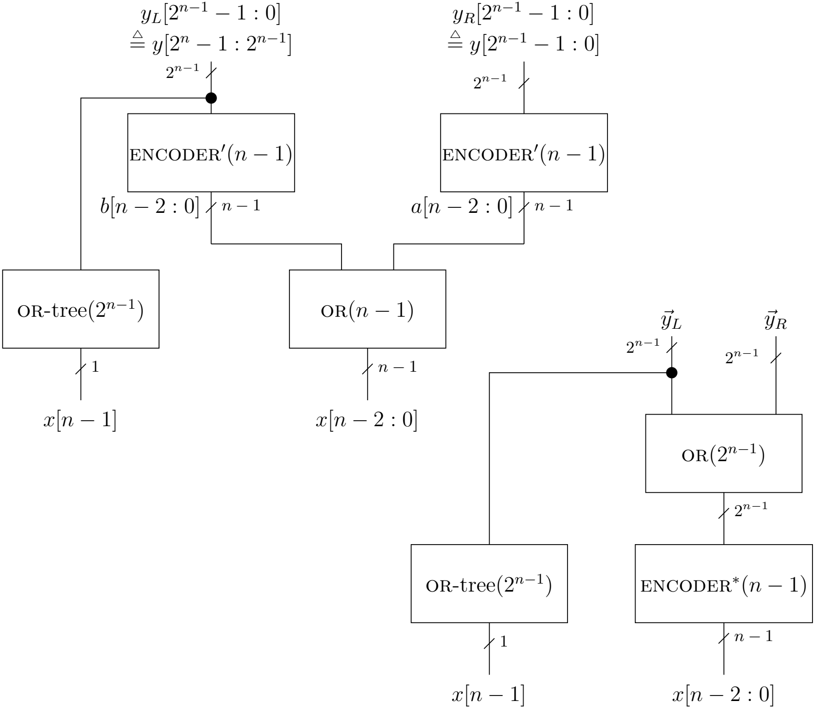
So $c(\textsc{Encoder}'(n))$ (asymptotically) equals the cost of the brute force design...

Reducing The Cost

Claim

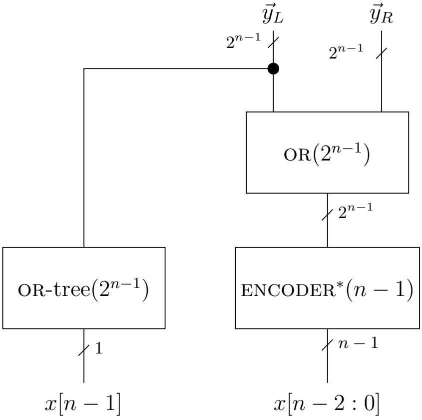
If $\text{wt}(y[2^n - 1 : 0]) \leq 1$, *then*

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



$y_L[2^{n-1}-1:0]$
$\triangleq y[2^n-1:2^{n-1}]$
$y_R[2^{n-1}-1:0]$
$\triangleq y[2^{n-1}-1:0]$
2^{n-1}
2^{n-1}
ENCODER$'(n-1)$
ENCODER$'(n-1)$
$b[n-2:0]$
$n-1$
$a[n-2:0]$
$n-1$
OR-tree(2^{n-1})
OR$(n-1)$
1
$n-1$
$x[n-1]$
$x[n-2:0]$
$\vec{y}_L$
$\vec{y}_R$
2^{n-1}
2^{n-1}
OR(2^{n-1})
2^{n-1}
OR-tree(2^{n-1})
ENCODER$^*(n-1)$
1
$n-1$
$x[n-1]$
$x[n-2:0]$

Correctness?



Functional Equivalence

Definition

Two combinational circuits are functionally equivalent if they implement the same Boolean function.

Claim

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

$$\textsc{encoder}_{n-1}(\textsc{or}(\vec{y}_L, \vec{y}_R)) = \textsc{or}(\textsc{encoder}_{n-1}(\vec{y}_L), \textsc{encoder}_{n-1}(\vec{y}_R)).$$

Claim

$\textsc{encoder}'(n)$ *and* $\textsc{encoder}^*(n)$ *are functionally equivalent.*

Corollary

$\textsc{encoder}^*(n)$ *implements the* $\textsc{encoder}_n$ *function.*

Cost analysis

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

$$c(\textsc{Encoder}^*(n)) = \begin{cases} 0 & \text{if n=1} \\ c(\textsc{Encoder}^*(n-1)) + (2^n - 1) \cdot c(\textsc{Or}) & \text{otherwis} \end{cases}$$

$C(2^k) \triangleq c(\textsc{Encoder}^*(k))/c(\textsc{Or})$. Then,

$$C(2^k) = \begin{cases} 0 & \text{if k=0} \\ C(2^{k-1}) + (2^k - 1) & \text{otherwise.} \end{cases}$$

we conclude that $C(2^k) = \Theta(2^k)$.

Claim

$c(\textsc{Encoder}^*(n)) = \Theta(2^n) \cdot c(\textsc{Or})$.

Delay analysis

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

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

Asymptotic Optimality

Theorem

For every encoder E of input length n:

$$d(E) = \Omega(n)$$
$$c(E) = \Omega(2^n).$$

Wrong Proof:

Focus on the output $x[0]$ and the Boolean function f_0 that corresponds to $x[0]$. Tempting to claim that $|cone(f_0)| \geq 2^{n-1}$, and hence the lower bounds follow.

But, this is not a valid argument because the specification of f_0 is a partial function (domain consists only of inputs whose Hamming weight equals one)... must come up with a correct proof!

Asymptotic Optimality

Theorem

For every encoder E of input length n:

$$d(E) = \Omega(n)$$
$$c(E) = \Omega(2^n).$$

Proof.

Consider the output $x[0]$. We claim that the graphical cone satisfies:

$$|cone_G(x[0])| \geq \frac{1}{2} \cdot 2^n.$$

Otherwise, there exists an even index i and an odd index j such that $\{i, j\} \cap cone_G(x[0]) = \emptyset$. Now consider two inputs: e_i (a unit vector with a one in position i) and e_j. The output $x[0]$ is the same for e_i, $0^{2^n} = flip_i(e_i) = flip_j(e_j)$ and e_j. This implies that $x[0]$ errs for at least of the inputs e_i or e_j. □

Parametric Specification

- The specification of $\text{DECODER}(n)$ and $\text{ENCODER}(n)$ uses the parameter n.
- The parameter n specifies the length of the input in the case of a decoder and the length of the output in an encoder.
- $\text{DECODER}(8)$ and $\text{DECODER}(16)$ are completely different circuits.
- $\{\text{DECODER}(n)\}_{n=1}^{\infty}$ is a family of circuits, one for each input length.

Summary - 1

We discussed:

- buses
- decoders
- encoders

Summary - 2

Three main techniques were used in this chapter.

- Divide & Conquer - a recursive design methodology.
- Extend specification to make problem easier. Adding restrictions to the specification made the task easier since we were able to add assumptions in our recursive designs.
- Evolution. Naive, correct, costly design. Improved while preserving functionality to obtain a cheaper design.