Reliable Design

3 Classes of design

1. Self checking systems - faults automatically detected by a subsystem (checker)

2. Fault-tolerant circuits - faults do not cause system malfunction

3. Easily testable systems - test generation problem is simplified
If only $2^k < 2^8$ output configurations can occur during normal operation, the occurrence of any of the $2^8 - k$ unallowable configurations indicates a malfunction.

A hardware checker can be used to automatically detect such faults.
Example

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a)

Figure 5.1 Automatic Fault Checking for Example 5.1

Output $f_1 = f_2 = 1$ never occurs. Checker detects this situation and generates fault indicator = 1 when it occurs.

$2^k$ normal outputs (valid) code words

$2^k - q$ unallowable outputs

$\Rightarrow$ invalid or non-code words.
Error Detecting and Correcting Codes

Classification - ability to detect errors

$e$ (bit) error detecting

Any fault affecting $\leq e$ bits can be detected.

$e$ (bit) error correcting - any fault affecting $\leq e$ bits can be corrected (set of faulty bits can be determined)

(d) Hamming distance of a code - minimum number of bits in which any two code words differ
Relation between Hamming distance and error detecting/corrected capability of code.

<table>
<thead>
<tr>
<th>$d$</th>
<th>capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>1-error detection, 0-error correction</td>
</tr>
<tr>
<td>3</td>
<td>2-error detection, 1-error correction</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$e + 1$</td>
<td>$e$-error detection, $\left\lfloor \frac{e}{2} \right\rfloor$-error correction</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$2e + 1$</td>
<td>$2e$-error detection, $e$-error correction</td>
</tr>
</tbody>
</table>

Figure 5.2  Capability of a Code with Distance $d$

* Smallest integer $\geq e/2$

Codes must use extra bits called check bits. Simplest is parity check code (only one check bit). Odd or even parity Single bit error detecting
Example

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.3

\[ Y_e \text{ is even parity check bit} \]
\[ Y_o \text{ is odd parity check bit} \]

\( f_1, f_2 \) are called information bits

\[ Y_o = Y_e. \]

Generalized class of parity check codes

8 information bits

\( c \) check bits

\[ 2^c \geq 2q + c + 1 \]

- Single bit error correcting

\[
\begin{array}{c|c}
q & c \\
---&---
1 & 2 \\
4 & 3 \\
11 & 4 \\
26 & 5 \\
57 & 6 \\
120 & 7 \\
\end{array}
\]
With use of codes can design circuits which automatically detect or correct errors.

Figure 5.5 (a) Error Detecting Circuit (b) Error Correcting Circuit

Different circuits require different type of codes.

Transmission $\Rightarrow$ parity check code may be adequate

Arithmetic $\Rightarrow$ parity check code may not be adequate

(Cannot predict parity check bit of $A + B$, $P(A+B)$ from parity of $A$ and parity of $B$, $P(A), P(B)$.}

Thus parity check bit for $A + B$ must be recomputed after addition.

Residue codes - for arithmetic operations (addition, multiplication), parity check bits of result can be determined from check bits of operands. $\Rightarrow$ independent checking for even parity.
**Definition of Residue Code**

Rightmost $p$ bits are check bits.

- **Information bits**
- **p check bits**

Define number $N$

Define number $C$

$$C = N \mod m$$

$m$ is a parameter called the **residue** of the code

$$p = \lceil \log_2 m \rceil$$
**Example**

In this example, the table demonstrates the relationship between the input bits and the output bits. The table shows the input values (I, I, I) and the corresponding output values (N, C, C, C).

<table>
<thead>
<tr>
<th>I</th>
<th>I</th>
<th>I</th>
<th>N</th>
<th>C</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.7 A 3-Bit Residue Code with \( m = 3 \)

\[ C = N \mod 3 \]

Residue = 3
Theorem 5.1: Let \( \{ a_i \} \) be a set of operands with check bits \( C^i = (a_i) \mod m \). The residue of the sum is \( (\sum a_i) \mod m \) and the residue of the product is \( (\prod a_i) \mod m \). Then

(a) \( (\sum a_i) \mod m = (\Sigma (a_i) \mod m) \mod m \). (Check bits of sum equal sum of check bits modulo \( m \).)

(b) \( (\prod a_i) \mod m = (\Pi (a_i) \mod m) \mod m \). (Check bits of product equal product of check bits modulo \( m \).)

Proof: Let \( a_i = k_{i1}m + k_{i2} \) where \( 0 \leq k_{i2} < m \).

(a) Then \( (\sum a_i) \mod m = (\Sigma (k_{i1}m + k_{i2}) \mod m = (\Sigma k_{i2}) \mod m \mod m \).

(b) Then \( (\prod a_i) \mod m = (\Pi (k_{i1}m + k_{i2}) \mod m = (\Pi k_{i2}) \mod m \mod m \). \qed
Use of Residue Codes to Check Addition

\[ A \quad B \]
\[ \text{Adder} \]
\[ A+B \]
\[ \text{Residue Calculator} \]
\[ C(A+B) \]

\[ C(A) \quad C(B) \]
\[ \text{Mod}_m \quad \text{Adder} \]
\[ (C(A)+C(B)) \mod m \]
\[ \text{Comparison} \quad \text{Error Indicator} \]

Figure 5.8
**Theorem:** In a residue code with \( m \) odd, all single bit errors are detected.

**Proof:** We must consider two cases.

**Case 1:** The single erroneous bit is an information bit. Then \( N' = N \pm 2^i \) if bit \( i \) of the information segment is in error. Since there is just a single error \( C = C' \). Thus \( C' = (N') \mod m \) if and only if \( (N') \mod m = (N) \mod m \). This implies that \( (N \pm 2^i) \mod m = (N) \mod m \). But \( (N \pm 2^i) \mod m = ((N) \mod m \pm (2^i) \mod m) \mod m \). For \( m \) odd, \( (2^i) \mod m \neq 0 \) and hence the error is detected.

**Case 2:** The single erroneous bit is a check bit. Then \( N = N' \) and \( C' = (C \pm 2^i) \mod m \) where bit \( i \) of the check segment is in error. The error is detected unless \( C' = C \). This implies that \( C = (C \pm 2^i) \mod m \) which is impossible for \( m \) odd and hence all single errors are detected. \( \Box \)
m odd → some single bit errors may be indistinguishable

m even → some single bit errors may be undetectable

Example

\[ m = 3 \]
\[ \overline{110100} \text{ code word 1} \]
\[ 84321 \text{ check bits} \]

Error in bit 4

Result in \[ \overline{100100} \]

Error is detected since

\[ 4 \mod 3 \neq 0 \]

\[ \overline{000100} \text{ code word 2} \]

Error in bit 5

Result in \[ \overline{100100} \]

Is detected.

However, the two cases cannot be distinguished since result of single bit error is the same
Let \( m = 2 \)
\[ 1 1 0 1 0 \]
\[ \text{code word} \]
\[ \text{check bit 6+} \]

error in bit 3
results in 1 0 0 1 0

This is a valid code word
since \( 4 \mod 2 = 0 \)
error is not detected

\( \Rightarrow m \) must be odd to
detect all single bit
errors
Other types of errors

Unidirectional errors - all erroneous bits have same value

Adjacent bit errors - all bits affected by an error form a connected string of bits

Other Codes

$k\choose m$ (k out of m) code - m bit words, each code word has exactly k 1's.

(Will detect all unidirectional errors)

Modified residue check code - m-1 check bits, defined so total # of 1-bits in a code word is a multiple of m.

(Detects all unidirectional errors affecting $\leq m$ bits)

Berger Code

$k$ nr bits, $[\log_2 (k+1)]$ check bits
Checking Circuits

Advantages of Hardware checking circuits

(1) Intermittent faults are detected.
(2) Errors are immediately detected upon occurrence. This prevents corrup-
tion of data.
(3) The distribution of checkers throughout a digital system provides a
good measure of the location of a fault by the location at which it is
detected.
(4) Software diagnostic program is eliminated, or at least simplified.

Disadvantages of Hardware Checking Circuits

(1) More hardware is required, including a hardware checker.
(2) Additional hardware must be checked or tested (checking the checker
problem). In general, some faults cannot be automatically detected.
Faults in this hardcore must still be tested by a diagnostic program.
Fault Tolerant Design

Redundancy techniques used to mask faults, system remains operational.
Used when repair is impossible or unfeasible.

Goal: Increase expected life of system (Mean time before failure)

\[ MTBF = \int_0^\infty -t dR(t) \]

\[ R(t) = \text{probability of survival until time } t \]

\[ MTBF = \frac{1}{\lambda} \text{ (a constant)} \]

\[ \Rightarrow R(t) = e^{-\lambda t} \]

\[ \begin{array}{c}
\text{1} \\
\text{0} \\
\text{t}
\end{array} \]

\[ \begin{array}{c}
\text{0} \\
\text{t}
\end{array} \]
Triplicated Modular Redundancy (TMR)

![TMR Diagram](image)

Based on error correcting code

\[
\begin{align*}
000 & \rightarrow \text{distance 3} \\
111 & \\
\end{align*}
\]

Assuming probability of failure of M is very small relative to L1, L2, L3

\[
R(t) = \frac{e^{-\lambda t} e^{-\lambda t} e^{-\lambda t}}{3 e^{-2\lambda t} (1 - e^{-\lambda t})}
\]

- Probability of all three of the subsystems surviving
- Probability of any two of the subsystems surviving

\[
= 3e^{-3\lambda t} - 2e^{-2\lambda t}.
\]

Two of three Li must fail for the same input for error to occur.
To find $t_0$:

$$3e^{-2\lambda t_0} - 2e^{-3\lambda t_0} - e^{-\lambda t_0} = 0$$

$$\Rightarrow t_0 \approx \frac{7}{\lambda}$$

MTBF = $\frac{5}{6\lambda}$

(less than that of single system)

$\Rightarrow$ increased reliability over a relatively short period of time.
TMR Concept can also be applied at the subsystem level.

![Figure 5.23](a) A System Containing Three Modules (b) A TMR Subsystem for L, (c) System Designed from TMR Subsystems

This is called multiplexing.

Will also correct errors in subsystem majority gates.

Will correct multiple errors occurring in different subsystems.
N Modular Redundancy

Generalization to $N$ copies of system, $N$ odd > 3

$N = 5, 7, 9, \ldots$

Failure does not occur until

$S = (N+1)/2$ modules fail

Static Redundancy

Basic System Configuration Never Changes. All copies of system active. System Reliability decreases with age.

Dynamic Redundancy

System reconfigures after each fault. System reliability fairly constant with age.
Hybrid Redundancy

Example of dynamic redundancy

3 Active Modules
N-3 Standby Modules (Inactive)

Detected error in an active module causes it to be automatically switched to inactive status and replaced in active status by one of the standby modules.
If no fault occurs in the switching network, system error doesn't occur until < 2 fault-free copies exist.
Faults in switching networks have two types of effect:

1. Some good (fault-free) module cannot be switched in (to active status).

or

2. Some faulty active status module cannot be switched out (to inactive status).

Type 1 \Rightarrow \text{equivalent to reduction of one standby module}\nN \Rightarrow N-1

Type 2 \Rightarrow \text{system with one (permanently) faulty active module. Next fault makes error correction impossible.}

One type of fault much more serious than the other. Design system to prevent serious fault.
$C_1 + C_2$ independent control signals.

This is an example of fail safe design.

Used when one type of fault much more costly than another.

Example

Intersection with two red lights

or Intersection with two green lights.
Fail Safe Design

A system is fail safe if no single fault can produce an incorrect 0.
(1 fail safe if no single fault can produce an incorrect 1)

More realistically, probability of incorrect 0 very very low.

Use error detecting code

Figure 5.25 Fail Safe System
Example

\[
\begin{array}{c|ccc|ccc}
Z_1 & Z_2 & Z_3 & Z_4 & Z_5 & Z_6 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

(a)

Given functions \(Z_1, Z_2\) define parity check bit \(Z_3\) for single bit error detection.

Want \(Z_1\) to be 0 fail safe
Want \(Z_2\) to be 1 fail safe

\[
\begin{array}{c|ccc|ccc|c}
\text{Z} & \text{Z} & \text{Z} & \text{Z} & \text{Z} & \text{Z} & \text{Q} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

(b)

Define \(Q_1 = Z_3\), for even parity
\(Q_1 = 0\) for odd parity
\(Q_2 = Z_4\) for even parity, \(Q_2 = 1\) for odd parity
Must design G so that for any input, any fault affects at most one output value.
Mismatch leads to attempt to diagnose error using software diagnostic program. Successful diagnosis leads to attempt to repair faulty unit.

$1/\lambda_1 \Rightarrow MTBF$

$1/\lambda_2 \Rightarrow MTBR$ (Mean Time Before Repair)

$p = \text{coverage} \Rightarrow \% \text{ of faults successfully diagnosed.}$
51 State representing two fault-free modules
52 State representing one fault-free module
53 State representing both modules inoperative (faulty)

Figure 5.28 (a) Idealized Model of System (b) Model of System with Coverage $p$
MTBF \approx \frac{\lambda_2}{2 \lambda_1^2} \left(1 + \frac{\lambda_2}{\lambda_1} (1-p)\right)

Assuming \(\frac{1}{\lambda_1} = 3\) months
\(\frac{1}{\lambda_2} = 2\) hours

<table>
<thead>
<tr>
<th>(p)</th>
<th>MTBF (years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>100</td>
</tr>
<tr>
<td>.995</td>
<td>20</td>
</tr>
<tr>
<td>.99</td>
<td>10</td>
</tr>
<tr>
<td>.95</td>
<td>2.5</td>
</tr>
<tr>
<td>.9</td>
<td>1</td>
</tr>
</tbody>
</table>

Imperfect coverage is the dominant factor in system failure when
\[1 - p > \frac{\lambda_1}{\lambda_2}\]
Design to Simplify Testing and Test Generation

Observation Points

Figure 5.36 (a) A Combinational Circuit (b) The Same Circuit with Observation Point Added

C Requires 5 tests to detect all single stuck faults
C' Requires 4 tests to detect all single stuck faults

In C to detect $x_2$ S-a-1
\[ T = x_1 \overline{x_2} x_3 (\overline{x_4} + \overline{x_5}) \]

In C' to detect $x_2$ S-a-1
\[ T = x_1 \overline{x_2} x_3 \]

Test $(1,0,1,1,1)$ detects $x_2$ S-a-1 in C', as well as $x_4, x_5$ S-a-0.
Control Points

Figure 5.37 (a) A Combinational Circuit (b) The Same Circuit with a Control Point and Observation Points

C' Requires 4 tests to detect all single stuck faults

Adding observation point at output of $G_1 \implies 4$ tests still required.

Adding Control Signal C + 2 Observation points $\implies 3$ tests sufficient

<table>
<thead>
<tr>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1, - , -)</td>
<td>(1, 1, - , 0)</td>
<td>(1, 1, 1, -)</td>
</tr>
<tr>
<td>(1, 0, - , -)</td>
<td>(0, 1, - , 0)</td>
<td>(- , - , 0, 1)</td>
</tr>
<tr>
<td>(1, 1, - , -)</td>
<td>(1, 0, - , 1)</td>
<td>(0, 1, 1, 0)</td>
</tr>
</tbody>
</table>

Combined (0,1,1,0), (1,0,0,1), (1,1,1,0)
Application to Sequential Circuits

Design with Embedded Shift Registers

Level Scan Sensitive Design

LSSD

Figure 5.41  Sequential Circuit with Observable and Controllable Feedback Paths

Use control signal C to shift from normal operation (C=0) to testing mode (C=1). When C=1 FF's interconnect to form a shift register to make state initialization easy.
SR Flip-Flops

\[ S_i = \overline{c_i} (x, y) + c_i \]
\[ R_i = \overline{c_i} (x, y) + \overline{c_i} \]
\[ S_i = \overline{c_i} (x, y) + \overline{c_i} \overline{c_i} \]  \{ for all \( i, 2 \leq i \leq n \) \}
\[ R_i = \overline{c_i} (x, y) + c_i \overline{c_i} \]
\[ z_i = \overline{c_i} (x, y) + c_i \]
\[ z_i = \overline{c_i} (x, y) + \overline{c_i} \]

When \( C = 1 \)

\( X_i \rightarrow F \rightarrow F \)
\( Y_i \rightarrow Y_2 \)
\( Y_2 \rightarrow Y_2 \)
\( Y_{n-1} \rightarrow Y_n \)
\( Y_n \rightarrow (Z_1, Z_2) \)

Shift Register

When \( C = 0 \) \( \Rightarrow \) Normal behavior
For each fault in C test pattern consists of state y and input x

Procedure 5.1 (Testing embedded shift register circuit):
(1) Set c = 1 to switch circuit to shift register mode.
(2) Check operation of shift register.
(3) Select a test pattern and set shift register to desired initial state by sequentially shifting in appropriate inputs.
(4) Set c = 0 to return to normal mode.
(5) Apply test pattern input to generate incorrect state (or output) by clocking the flip-flops.
(6) Set c = 1 to return to shift register mode.
(7) Shift out final state to detect fault.
(8) Repeat (3)-(7) for all other test patterns.