Positional Number System

- A number is represented by a string of digits where each digit position has an associated weight.
- The weight is based on the radix of the number system.
- Some common radices:
  - Decimal.
  - Binary.
  - Hexadecimal.
Notation

- **Decimal.**
  - $W = 123_{10} = 123d = 123$

- **Binary.**
  - $X = 102 = 10b$

- **Hexadecimal.**
  - $Z = 0A3_{16} = 0A3h = 0xA3$
Radix 10 Numbers

Decimal.
- Here the base or radix is 10.

Digits used.
- 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

\[ D = 217 \]
\[ D = 200 + 10 + 7 \]
\[ D = 2 \times 10^2 + 1 \times 10^1 + 7 \times 10^0 \]
Radix 2 Numbers

- Binary.
  - Here the base or radix is 2.

- Digits used:
  - 0, and 1.

\[
B = 101.1_2
\]

\[
B = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}
\]

\[
B = 5.5
\]
Radix 16 Numbers

Hexadecimal.

- Here the base or radix is 16.

Digits used:

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

\[
H = 3C_{16} \quad \text{or} \quad H = 3 \times 16^1 + C \times 16^0 + 8 \times 16^{-1}
\]

\[
H = 60.5
\]
Radix $r$ Numbers

$$A = a_n \ldots a_1 a_0 \cdot a_{-1} \ldots a_{-m}$$

$$A = a_n r^n + \ldots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \ldots + a_{-m} r^{-m}$$

$$A = \sum_{i=-m}^{n} a_i r^i$$
## Conversion Table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>15</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>16</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>17</td>
<td>F</td>
</tr>
</tbody>
</table>
Binary to Octal, Octal to Binary Conversions

Substitute a three binary digit string with an octal digit.

\[ \begin{align*}
1 & \quad 001 \quad . \quad 1_2 = 11 \quad . \quad 4_8 \\
754_8 & = 111_2 \quad 101_2 \quad 100_2
\end{align*} \]
Binary to Hexadecimal, Hexadecimal to Binary Conversions

Substitute a four binary digit string, called a nibble, with a hexadecimal digit.

\[ 8 \text{A}.C_{16} = 100001010.11_2 \]
\[ 11110101112 = 1\text{EB}_{16} \]
The conversions just described are simple due to the fact that the radices are all powers of two.

- $2^1 = \text{binary}$.
- $8 = 2^3 = \text{octal}$.
- $16 = 2^4 = \text{hexadecimal}$.
Radix $r$ to Decimal Conversions

\[ A = \sum_{i = -m}^{n} a_i r^i \]

\[ A = a_n r^n + \ldots + a_{-m} r^{-m} \]

\[ \text{Ex:} \]

- $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$
- $432_5 = 4 \times 5^2 + 3 \times 5^1 + 2 \times 5^0 = 117$
- $75_8 = 7 \times 8^1 + 5 \times 8^0 = 61$
- $1A_{16} = 1 \times 16^1 + 10 \times 16^0 = 26$
Decimal to Radix r Conversions

\[ A = \sum_{i=-m}^{n} a_i r^i \]
\[ A = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_0 r^0 + a_{-1} r^{-1} + \ldots + a_{-m} r^{-m} \]
\[ A = A_I + A_F \]

Integral part
\[ A_I = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_0 r^0 \]

Fractional part
\[ A_F = a_{-1} r^{-1} + \ldots + a_{-m} r^{-m} \]
Decimal to Radix $r$ Conversions

Integral part

\[ A_I = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_1 r^1 + a_0 r^0 \]

\[ A_I = ((a_n r + a_{n-1}) r + \ldots + a_1) r + a_0 \]

\[ A_I / r = (a_n r + a_{n-1}) r + \ldots + a_1 \] as the quotient and \( a_0 \) as the remainder.

Divide the result repeatedly until a zero quotients is reached.

The remainders of the consecutive divisions form the numbers in base $r$. 
# Decimal to Radix r Conversions

Convert 26 decimal to binary.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>26/2</td>
<td>13</td>
</tr>
<tr>
<td>13/2</td>
<td>6</td>
</tr>
<tr>
<td>6/2</td>
<td>3</td>
</tr>
<tr>
<td>3/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

$26 = 11010_2$
Decimal to Radix r Conversions

Fractional Part

\[ A_F = a_{-1}r^{-1} + a_{-2}r^{-2} + \ldots + a_{-m}r^{-m} \]

\[ A_F = r^{-1}(a_{-1} + r^{-1}(a_{-2} + \ldots + (r^{-1}a_{-m}))) \]

\[ A_F \times r = r^{-1}(a_{-1}+ r^{-1}(a_{-2} +\ldots)) \times r \]

\[ = a_{-1} + r^{-1}(a_{-2} +\ldots) \]

Multiplying the fractional part by \( r \) results in a mixed number.

The integral part of this mixed number is the conversion’s required digit.
Decimal to Radix $r$ Conversions

This algorithm is not guaranteed to terminate, since a finite fraction in one number system may correspond to an infinite one in another number system.
Decimal to Radix r Conversions

Convert 0.75 decimal to binary.

<table>
<thead>
<tr>
<th>Mixed no.</th>
<th>Integral part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.75 X 2</td>
<td>1.5</td>
</tr>
<tr>
<td>0.5 X 2</td>
<td>1.0</td>
</tr>
</tbody>
</table>

0.75 = 0.11<sub>2</sub>
Negative Number Representation

- Signed magnitude.
- Complement number system:
  - Radix-complement.
  - Diminished radix-complement.
Signed Magnitude Representation

\[ A = a_{n-1} \ a_{n-2} \ \ldots \ a_0 \]

- \( A \) is a \( n \) digit number, where \( a_{n-1} \) is the sign, and the remaining \( n-1 \) bits are the magnitude.

- \( a_{n-1} \)
  - Positive if equal to 0.
  - Negative if equal to 1 or \( r-1 \) for \( r \) greater than 2.

- Range is \(- (2^{n-1} - 1)\) to \((2^{n-1} - 1)\).

- Two zero representations:
  - \( 00\ldots0_2 = +0 \)
  - \( 10\ldots0_2 = -0 \)
Addition and Subtraction

**Check sign.**

- If equal.
  - Add magnitudes and give the result the same sign.

- If different.
  - Compare magnitudes.
  - Subtract the smaller from the larger and give the result the sign of the larger.
Complement Representation

- Negation is accomplished by taking the complement of the numbers.
- Two numbers in complement representation may be added or subtracted directly without sign and magnitude checks.
- Numbers in complement representation may be sign extended.
Diminished Radix Complement

- Also known as:
  - One’s complement ($r = 2$).
  - Nine’s complement ($r = 10$).

- 1’s Complement of A is $B = (2^n - 1) - A$
  - $n$ is the numbers of digits in A.
Diminished Radix Complement

Example: Compute the complement of $A$, for $A = 11100010_2$

$B = \text{Complement of } A$

$B = (2^8 - 1) - 11100010_2$

$B = (100000000_2 - 1_2) - 11100010_2$

$B = 11111111_2 - 11100010_2$

$B = 00011101_2$
Diminished Radix Complement

The easy way to get the one’s complement of a binary number is by complementing each individual digit of that number.

Ex:
- $A = 11100010_2$
- $B = \text{One’s complement of } A$
- $B = 00011101_2$
Diminished Radix Complement

- Range is $-(2^{n-1} - 1)$ to $(2^{n-1} - 1)$.
- Two zero representations:
  - $00...0_2 = +0$
  - $11...1_2 = -0$
Radix Complement

Also known as:
- Two’s complement \((r = 2)\).
- Ten’s complement \((r = 10)\).

2’s Complement of \(A = 2^n - A\)
- \(n\) is the numbers of digits in \(A\).

\[ B = 2^n - A - 1 + 1 \]
\[ B = (2^n - 1 - A) + 1 \]
Radix Complement

Ex:

- $A = 11100010_2$
- $B = \text{Two's complement of } A$
- $B = 00011101_2 + 1_2$
- $B = 00011110_2$
Radix Complement

The easy way to get the two’s complement of a binary number is to search that number starting with the LSB until you find the rightmost one digit. Leave that digit and all other digits to the right of it unchanged. Complement all digits to the left of that one digit.

Ex:
- A = 11101000₂
- B = Two’s complement of A
- B = 00011000₂
Radix Complement

- Range is $-2^{n-1}$ to $2^{n-1} - 1$.
- One zero representation:
  - $00...0_2 = 0$
**Two’s Complement Addition**

- Simply add the numbers ignoring any carries beyond the MSB.
- The result will always be correct as long as the range of the number system is not exceeded.

**Example:**

\[
\begin{align*}
2 & \quad 0010 & \quad - & \quad 2 & \quad 1110 \\
5 & \quad 0101 & \quad + & \quad 5 & \quad 0101 \\
7 & \quad 0111 & \quad + & \quad 3 & \quad 0011
\end{align*}
\]
Overflow

Overflow occurs when the result of an addition exceeds the range of the number system.

So, will overflow occur when numbers with different signs are added?

Detection of an overflow:

- Compare the carries into and out of the sign bit, if they are different an overflow has occurred.
- The sign of the addends are the same and they differ from the sums sign.
- **Ex:**
  - $-4 + (-5) = -9$
  - $4 + 5 = 9$
Overflow

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>-4</td>
<td>1100</td>
<td>5</td>
<td>0101</td>
<td>-5</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>-9</td>
<td>0111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two’s Complement Subtraction

Subtraction is performed by negating the subtrahend by taking its two’s complement, then adding it to the minuend using normal two’s complement addition rules.

Ex:

\[
\begin{array}{cccc}
4 & 0100 & + & 0100 \\
2 & 0010 & + & 1110 \\
2 & 0010 & + & 0010 \\
\end{array}
\]
One’s Complement Addition

- Perform addition; If there is a carry out of the sign bit, add one to the result.
- This is called the end-around carry.

Ex:
- $-3 + (-4) = -7$
One’s Complement Subtraction

Subtraction is performed by negating the subtrahend by taking its one’s complement, then adding it to the minuend using normal one’s complement addition rules.

Ex:

- 3 – ( -4 ) = 1
Boolean Algebra

1854, George Boole created a two valued algebraic system which is now called Boolean algebra.

1938, Claude Shannon adapted Boolean algebra to analyze and describe the behavior of circuits built with relays. This adaptation is called switching algebra.
In switching algebra the condition of a logic signal is represented by symbolic variables, such as x, y, and/or z, and these variables can only have two values, 0 or 1.

Two possible conventions:

- Positive Logic.
  - Where LOW = 0 and HIGH = 1.
- Negative Logic.
  - Where LOW = 1 and HIGH = 0.
Axioms

The axioms or postulates of a mathematical system are a minimum set of basic definitions that are assumed to be true, and from which all other information about the system can be derived.

The axioms stated below embody the “digital abstraction” by formally stating that X can take on only one of two values.

- (A1) \( X = 0 \) if \( X \neq 1 \)
- (A1') \( X = 1 \) if \( X \neq 0 \)
Axioms

Complement.

- (A2) If $X = 0$, then $X' = 1$.
- (A2') If $X = 1$, then $X' = 0$.

Notation.

\[ \neg X \]
\[ \sim X \]
\[ X' \]
\[ X \]
Axioms

Logical multiplication ($\cdot$).

- (A3) $0 \cdot 0 = 0$
- (A4) $1 \cdot 1 = 1$
- (A5) $0 \cdot 1 = 1 \cdot 0 = 0$

Logical addition (+).

- (A3') $1 + 1 = 1$
- (A4') $0 + 0 = 0$
- (A5') $1 + 0 = 0 + 1 = 1$
Precedence

By convention, the precedence of operations in a logic expression is the following:

- Parentheses.
- Complement.
- Multiplication.
- Addition.
Theorems

Theorems are statements, known to be always true, that are used to manipulate algebraic expressions to allow simpler analysis or more efficient synthesis of circuits.

Identities.

- (T1) \( X + 0 = X \)
- (T1') \( X \cdot 1 = X \)
Theorems

Null elements.
- (T2) $X + 1 = 1$
- (T2') $X \cdot 0 = 0$

Idempotency.
- (T3) $X + X = X$
- (T3') $X \cdot X = X$
Theorems

ΔInvolution.
- \((T4) (X')' = X\)

ΔComplements.
- \((T5) X + X' = 1\)
- \((T5') X \cdot X' = 0\)
Theorems

Proofs.

- Theorems T1 through T5' can be proved by using a technique called perfect induction.
- Since a switching variable can take on only two different values, 0 or 1 by Axiom A1, we can prove a theorem involving a single variable by showing that the theorem is true for both X=0 and X=1.
Theorems

Proof of theorem (T2).

\[ X + 1 = 1 \]

Two cases:

- \( X = 0 \)
  - \( 0 + 1 = 1 \) is true according to A5’.

- \( X = 1 \)
  - \( 1 + 1 = 1 \) is true according to A3’.
Theorems

◆ Commutativity.
  - (T6) $X + Y = Y + X$
  - (T6') $X \cdot Y = Y \cdot X$

◆ Associativity.
  - (T7) $(X + Y) + Z = X + (Y + Z)$
  - (T7') $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

◆ Distributivity.
  - (T8) $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$
  - (T8') $(X + Y) \cdot (X + Z) = X + Y \cdot Z$
Theorems

- **Covering (absorption).**
  - (T9) $X + X \cdot Y = X$
  - (T9') $X \cdot (X + Y) = X$

- **Combining.**
  - (T10) $X \cdot Y + X \cdot Y' = X$
  - (T10') $(X + Y) \cdot (X + Y') = X$
Theorems

◆ Consensus.

- \((\text{T11})\) \(X \cdot Y + X' \cdot Z + Y \cdot Z\)
  \[= X \cdot Y + X' \cdot Z\]
- \((\text{T11'})\) \((X + Y) \cdot (X' + Z) \cdot (Y + Z)\)
  \[= (X + Y) \cdot (X' + Z)\]

◆ Comments.

- T9 and T10 are used in minimization of logic functions.
- T10 used to eliminate timing hazards and to find prime implicants (iterative consensus method).
Theorems

Proof of theorem (T9).

\[ X + X \cdot Y = X \quad (T1') \]
\[ X \cdot 1 + X \cdot Y = X \quad (T8) \]
\[ X \cdot (1 + Y) = X \quad (T2) \]
\[ X \cdot 1 = X \quad (T1') \]
\[ X = X \]

Proof of theorem (T10).

\[ X \cdot Y + X \cdot Y' = X \quad (T8) \]
\[ X \cdot (Y + Y') = X \quad (T5) \]
\[ X \cdot 1 = X \quad (T1') \]
\[ X = X \]
Theorems

Any expression can be substituted for X, Y and Z in the previous theorem.

For example:

Simplify \( W = A'BC + A' \).

Substitute \( X = A' \) and \( Y = BC \).

\[
W = XY + X
\]

According to theorem (T9)

\[
XY + X = X
\]

Therefore

\[
W = X = A'
\]
Theorems

Simplify:

\[ W = [A + B'C + DEF] \cdot [A + B'C + (DEF)'] \]

Substitute \( X = A + B'C \) and \( Y = DEF \)

\[ W = [X + Y] \cdot [X + Y'] \]

According to theorem (T10')

\[ [X + Y] \cdot [X + Y'] = X \]

Therefore

\[ W = X = A + B'C \]
Theorems

Generalized Idempotency.

- (T12) $X + X + \ldots + X = X$
- (T12') $X \cdot X \cdot \ldots \cdot X = X$

DeMorgan’s Theorem.

- (T13) $(X_1 \cdot X_2 \cdot \ldots \cdot X_n)' = X_1' + X_2' + \ldots + X_n'$
- (T13') $(X_1 + X_2 + \ldots + X_n)' = X_1' \cdot X_2' \cdot \ldots \cdot X_n'$
Theorems

Generalized DeMorgan’s Theorem.

- (T14) \([F(X_1, X_2, \ldots, X_n, +, \cdot)]' = [F(X_1', X_2', \ldots, X_n', +, \cdot)]\]

Example:

- 
  \((X \cdot Y + W \cdot Z)' = (X' + Y') \cdot (W' + Z')\)
Shannon’s Expansion Theorem.

- (T15) \( F(X_1, X_2, \ldots, X_n) = X_1 \cdot F(1, X_2, \ldots, X_n) + X_1' \cdot F(0, X_2, \ldots, X_n) \)

- (T15’) \( F(X_1, X_2, \ldots, X_n) = X_1 + F(0, X_2, \ldots, X_n) \cdot X_1' + F(1, X_2, \ldots, X_n) \)
Theorems

Another technique used to prove theorems is the finite induction one. With finite induction, first you prove that the theorem is true for the case where $n = 2$ (basis step), then you prove that if the theorem is true for $n = i$, then it is also true for $n = i + 1$ (induction step).
Ex: Prove theorem (T12)

\[ X + X + \ldots + X = X \]

Basis step.

\[ X + X = X \text{ true by (T3)}. \]

Induction step.

\[ X + X + X = X \]
\[ (X + X) + X = X \]
\[ (X) + X = X \]
\[ X = X + X + X \]
Classification of Digital Circuits

- **Combinational logic circuits.**
  - Output depends only on present input.

- **Sequential circuits.**
  - Output depends on present input and present state of the circuit.
Combinational Logic Design

Procedure

Start with the problem statement.

Determine the number of inputs variables and the required number of output variables.

Derive a truth table that defines the required relationship between input and output.

Simplify each output function (Karnaugh maps).

Draw the logic diagram.
Half Adder Design Example

A half adder computes the sum of two one bit Boolean inputs, which can be at most 102. This requires two outputs.

Inputs: X, and Y.

Outputs: S and C.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Half Adder Design Example

- $C = XY$
- $S = X \oplus Y$
Latches and Flip-Flops

These sequential devices differ in the way their outputs are changed:

- The output of a latch changes independent of a clocking signal.
- The output of a flip-flop changes at specific times determined by a clocking signal.
S-R Latch

- SR latch based on NOR gates.
- The S input sets the Q output to 1 while R reset it to 0.
- When R=S=0 then the output keeps the previous value.
- When R=S=1 then Q=Q’=0, and the latch may go to an unpredictable next state.
S’-R’ Latch

- S’R’ latch based on NAND gates.
- The S’ input sets the Q output to 1 while R’ reset it to 0.
- When R’=S’=1 then the output keeps the previous value.
- When R’=S’=1 then Q=Q’=1, and the latch may go to an unpredictable next state.

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q0</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Q_1</td>
<td>Q_1</td>
</tr>
</tbody>
</table>
D Latch

- This latch eliminates the problem that occurs in the S’R’ latch when R=S=0.
- C is an enable input:
  - When C=1 then the output follows the input D and the latch is said to be open.
  - When C=0 then the output retains its last value and the latch is said to be closed.

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>$Q_0$</th>
<th>$\overline{Q_0}$</th>
<th>$Q_{-1}$</th>
<th>$\overline{Q}_{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td></td>
<td></td>
<td>$Q_{-1}$</td>
<td>$\overline{Q}_{-1}$</td>
</tr>
</tbody>
</table>
Edge Triggered J-K Flip-Flop

The operation of inputs J and K in the J-K flip-flop is similar to the operation of inputs S and R in the S-R flip-flop. The difference arises when J and K are asserted simultaneously. In this situation the output of the J-K flip-flop inverts its current state.
T Flip-Flop

- Also known as the toggle flip-flop.
- When input $T = 0$ the output $Q$ retain its previous value.
- When input $T = 1$ the output $Q$ inverts on every tick of the clock.
- When inputs $J$ and $K$ of a J-K flip-flop are connected together, the J-K flip-flop will behave like a T flip-flop.
Sequential Logic Design

Procedure

- Derive a state/output table from the problem specification.
- Minimize the number of states in the state/output table by eliminating equivalent states.
- Choose a set of state variables. Assign to each state a unique combination from the set derived above.
- Create a transition/output table.
Sequential Logic Design

Procedure

Choose a flip-flop type and construct its excitation table.

Using the excitation table fill the values for the input excitation function columns on the transition/output table.

Derive the excitation and output equations.

Draw logic diagram.
Sequence Detector Design

Example

Design a sequential circuit with one input (I) and one output (Z). The output is asserted when the input sequence 0-1-1 is received.

See state/output table below.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0</td>
<td>S₀</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Init</td>
<td>0</td>
</tr>
<tr>
<td>S₀</td>
<td>0</td>
<td>S₀</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>S₀₁</td>
<td>0</td>
</tr>
<tr>
<td>S₀₁</td>
<td>0</td>
<td>S₀</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>S₀₁₁</td>
<td>0</td>
</tr>
<tr>
<td>S₀₁₁</td>
<td>0</td>
<td>S₀₁₁</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>S₀₁₁</td>
<td>1</td>
</tr>
</tbody>
</table>
Sequence Detector Design

Example

Set of state variables and their unique assignment to the different states.

<table>
<thead>
<tr>
<th>State</th>
<th>Q1</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Init}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$S_{01}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$S_{011}$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
## Sequence Detector Design Example

### Transition/ output table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>$Q_0$</td>
<td>$I$</td>
<td>$Q_1^*$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Sequence Detector Design
Example

See excitation table below.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>D</th>
<th>J</th>
<th>K</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Sequence Detector Design

Example

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
<th>Input Excitation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Q_1$</td>
<td>$Q_0$</td>
<td>$I$</td>
<td>$Q_1^*$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Equations derived from the table above:

- $J_1 = IQ_0$
- $K_1 = I'Q_0$
- $J_0 = I'Q_1$
- $K_0 = IQ_1$
- $Z = Q_1Q_0'$
Sequence Detector Design
Example

See the logic diagram for the circuit below.
Transition Time

- Time interval between two reference points on a waveform. These reference points are usually 10% and 90% of the voltage change.
  - *Rise time* \( t_r \) - Time interval when waveform is changing from a logic low to a logic high level.
  - *Fall time* \( t_f \) - Time interval when waveform is changing from a logic high to a logic low level.
Propagation Delay

-Time it takes for a change at the input of a device to produce a change at the output of the same.

- $t_{plH}$ is the propagation delay when the output changes from LOW to HIGH.
- $t_{pHL}$ is the propagation delay when the output changes from HIGH to LOW.
- $t_{plH}$ and $t_{pHL}$ are not necessarily equal, and their values depends on the logic family.
Propagation Delay and Transition Time
DC Noise Margins

- The maximum amount of voltage variation (noise) that may be permitted for LOW or HIGH voltage levels.
- $V_{OH\text{Min}}$: the minimum output voltage in the HIGH state.
- $V_{IH\text{Min}}$: the minimum input voltage guaranteed to be recognized as a HIGH.
- $V_{IL\text{Max}}$: the maximum input voltage guaranteed to be recognized as a LOW.
- $V_{OL\text{Max}}$: the maximum output voltage in the LOW state.
DC Noise Margins

- High-State = $V_{OH\text{Min}} - V_{IH\text{Min}}$
- Low-State = $V_{IL\text{Max}} - V_{OL\text{Max}}$