BINARY CODED DECIMAL: B.C.D.

- ANOTHER METHOD TO REPRESENT DECIMAL NUMBERS
- USEFUL BECAUSE MANY DIGITAL DEVICES PROCESS + DISPLAY NUMBERS IN TENS

IN BCD EACH NUMBER IS DEFINED BY A BINARY CODE OF 4 BITS.

*** 8 – 4 – 2 – 1 MOST COMMON CODE

8 – 4 – 2 – 1 CODE INDICATES THE WEIGHT OF EACH BIT $2^3 – 2^2 – 2^1 – 2^0$

E.G. 934 = 1001 0011 0100

9 3 4

FOR EACH DIGIT A BINARY [NORMAL] CODE IS ALLOCATED.

OTHER REPRESENTATION FORMS ARE 2-4-2-1 AND EXCESS-3
We will use 8-4-2-1 BCD.

Decimal numbers > 9 may be obtained when adding two decimal digits (range: 0-18). I.e. 0 + 0 ÷ 9 + 9. Only 0→9 have the correct BCD code.

We need to correct the others.

<table>
<thead>
<tr>
<th>BINARY</th>
<th>8-4-2-1</th>
<th>2-4-2-1</th>
<th>EXCESS-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>NOT USED</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>NOT USED</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>NOT USED</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>NOT USED</td>
<td>2</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>NOT USED</td>
<td>3</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>NOT USED</td>
<td>4</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>NOT USED</td>
<td>5</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>NOT USED</td>
<td>6</td>
</tr>
<tr>
<td>1010</td>
<td>NOT USED</td>
<td>NOT USED</td>
<td>7</td>
</tr>
<tr>
<td>1011</td>
<td>NOT USED</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>1100</td>
<td>NOT USED</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>1101</td>
<td>NOT USED</td>
<td>7</td>
<td>NOT USED</td>
</tr>
<tr>
<td>1110</td>
<td>NOT USED</td>
<td>8</td>
<td>NOT USED</td>
</tr>
<tr>
<td>1111</td>
<td>NOT USED</td>
<td>9</td>
<td>NOT USED</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>UNCORRECTED BCD SUM</td>
<td>CORRECTED BCD SUM</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( C'_3 S'_3 S'_2 S'_1 S'_0 )</td>
<td>( C_N S_3 S_2 S_1 S_0 )</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 0 1 0</td>
<td>1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 0 1 1</td>
<td>1 0 0 0 1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1 1 0 0</td>
<td>1 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1 1 0 1</td>
<td>1 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1 1 1 0</td>
<td>1 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1 1 1 1</td>
<td>1 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1 0 0 0 0</td>
<td>1 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1 0 0 0 1</td>
<td>1 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1 0 0 1 0</td>
<td>1 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1 0 0 1 1</td>
<td>1 1 0 0 1</td>
<td></td>
</tr>
</tbody>
</table>

- 0→9 ONLY LEGAL CODES
  
  E.G. 19 = 1 \ 9 = 0001 1001 = 11001

THUS, FOR SUMS BETWEEN 10 → 18 MUST SUBTRACT 10 AND PRODUCE A CARRY

SUBTRACT 10 = 1010₂ >> ADD 2’s COMPLEMENT = 0110
4-BIT BCD ADDER

TO ADD TWO DIGITS

FOR SUMS > 9 WE NEED TO ADD 2’s COMPLEMENT of $10_{10}$ TO THE UNCORRECTED RESULT ($S_3' S_2' S_1' S_0'$)

CORRECTION IS ALSO NEEDED WHEN A CARRY OUT ($C_3'$) IS GENERATED [NUMBERS 16 → 18]

>>> A DECODER IS REQUIRED TO DETECT WHEN CARRY OUT ($C_N'$) TO THE NEXT STAGE IS NEEDED

K-MAP FOR $C_N$

<table>
<thead>
<tr>
<th></th>
<th>$S_1'$</th>
<th>$S_0'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_3'$</td>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1 1</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>1 0</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

>>> $C_N = C_3' + S_3' S_2' + S_3' S_1'$
TO IMPLEMENT A 4_BIT BCD ADDER WE NEED TWO 4-BIT FULL ADDERS, ONE TO ADD TWO 4-BIT BCD NUMBERS AND THE OTHER FULL ADDER TO ADD 2’s COMPLEMENT OF 1010 TO THE RESULT IF C_N = 1

ALSO WE NEED 2 AND GATES AND ONE OR GATE TO GENERATE C_N

- ADD 0110 WHEN C_N=1
- ADD 0000 WHEN C_N=0
BCD SUBTRACTION

9’s COMPLEMENT

THE 9’s COMPLEMENT OF A DECIMAL NUMBER IS FOUND BY SUBTRACTING EACH DIGIT IN THE NUMBER FROM 9

<table>
<thead>
<tr>
<th>DECIMAL DIGIT</th>
<th>9’s COMPLEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

E.G. 9’s COMPLEMENT of 28 = 99 – 28 = 71

9’s COMPLEMENT of 562 = 999 – 562 = 437

SUBTRACTION OF A SMALLER DECIMAL NUMBER FROM A LARGER ONE CAN BE DONE BY ADDING THE 9’s COMPLEMENT OF THE SMALLER NUMBER TO THE LARGER NUMBER AND THEN ADDING THE CARRY TO THE RESULT (END AROUND CARRY).
WHEN SUBTRACTING A LARGER
NUMBER FROM A SMALLER ONE
THERE IS NO CARRY AND THE RESULT
IS IN 9’s COMPLEMENT FORM AND NEGATIVE.

EXAMPLES:

(a) \[ \begin{array}{c}
+8 \\
-3 \\
\hline
5
\end{array} \quad \begin{array}{c}
+8 \\
+6 \\
\hline
(1) 4 \\
\hline
+1 \\
\hline
5
\end{array} \quad 9’s \text{ COMP. OF 3} \]
END AROUND CARRY

(b) \[ \begin{array}{c}
54 \\
-21 \\
\hline
33
\end{array} \quad \begin{array}{c}
54 \\
78 \\
\hline
(1) 32 \\
\hline
+1 \\
\hline
33
\end{array} \quad 9’s \text{ COMP. OF 3} \]
END AROUND CARRY

(c) \[ \begin{array}{c}
15 \\
-28 \\
-13 \\
\hline
86
\end{array} \quad \begin{array}{c}
15 \\
+71 \\
\hline
86 \\
\hline
-13
\end{array} \quad 9’s \text{ COMP. OF 3} \]

NO CARRY >>> NEGATIVE RESULT

\[ 86 - 99 = -13 \]
BCD SUBTRACTION

RECALL FOR DECIMAL SUBTRACTION:

$$A - B = A + [9's\ COMPLEMENT\ OF\ B]$$

• SIMILARLY FOR BCD

RULES:

(a) ADD 9’s COMP. OF B TO A

(b) IF RESULT > 9, CORRECT BY ADDING 0110

(c) IF MOST SIGNIFICANT CARRY IS PRODUCED [i.e. =1] THEN THE RESULT IS POSITIVE AND THE END ARROUND CARRY MUST BE ADDED.

(d) IF MOST SIGNIFICANT CARRY IS 0 [i.e. NO CARRY] THEN THE RESULT IS NEGATIVE AND WE GET THE 9’s COMP. OF THE RESULT.
**E.G.**  \( 8 - 3 = 8 + [9\text{'s COMP. OF } 3] \)
\[
= 8 + 6
\]
\[
\begin{array}{c}
1000 \\
0110 \\
1110 \leftarrow \text{INVALID (>9)}
\end{array}
\]
\[
0110 \leftarrow \text{CORRECTION}
\]

(1)  \[
\begin{array}{c}
0100 \\
\hline
1
\end{array}
\]
\[
\begin{array}{c}
\rightarrow
\end{array}
\]
\[
\begin{array}{c}
0101 = 5
\end{array}
\]

(b)  \( 3 - 8 = -5 \)
\[
\begin{array}{c}
0011 \\
0001 \\
0100
\end{array}
\]

**NO CARRY >>> NEGATIVE**

9’s COMP. OF 0100 = 0101 = -5

(c)  \( 87 - 39 \quad >>> \quad 87 + [9\text{'s COMP OF } 39] \)

\[
\begin{array}{c}
8 7 \\
6 0
\end{array}
\]
\[
\begin{array}{c}
1000 \\
0110 \\
\hline
1110
\end{array}
\]
\[
\begin{array}{c}
0111
\end{array}
\]
\[
\begin{array}{c}
\text{INVALID}
\end{array}
\]
\[
\begin{array}{c}
0110
\end{array}
\]

(1)  \[
\begin{array}{c}
0100 \\
\hline
0111
\end{array}
\]
\[
\begin{array}{c}
1
\end{array}
\]
\[
\begin{array}{c}
\rightarrow
\end{array}
\]
\[
\begin{array}{c}
0100 \\
1000
\end{array}
\]
\[
\begin{array}{c}
\hline
1
\end{array}
\]
\[
\begin{array}{c}
0100 \\
1000
\end{array}
\]
\[
\begin{array}{c}
\hline
1
\end{array}
\]
\[
\begin{array}{c}
= \quad 4 \quad 8
\end{array}
\]
(d) \[ 18 - 72 \ggg 18 + [27] \]

\[
\begin{array}{ll}
0001 & 1000 \\
0010 & 0111 \\
\hline
0011 & 1111 \rightarrow \text{NO CARRY NEGATIVE} \\
0001 & 0110 \\
\hline
0100 & (1) 0101 \rightarrow \text{CORRECTION} \\
\end{array}
\]

\[
4 \quad 5 = -54
\]

OUTPUT IS A NEGATIVE NUMBER \ggg

THE RESULT IS IN 9’s COMP. FORM

(e) \[ 65 - 12 \ggg 65 + [87] \]

\[
\begin{array}{ll}
0110 & 0101 \\
1000 & 0111 \\
\hline
1110 & 1100 \rightarrow \text{INVALID CORRECTION} \\
0110 & 0110 \\
\hline
(1) 0100 & (1) 0010 \\
\hline
1 & 1 \rightarrow \text{END AROUND CARRY} \\
0101 & 0011 \\
\hline
5 & 3
\end{array}
\]
9's COMP

BCD ADDER

C₀

TRUE/9's COMP

0 TRUE
1 COMP.

BCD RESULT

SIGN:
0 POSITIVE
1 NEGATIVE
9’s COMPLEMENT

9’s COMPLEMENT OF A NUMBER

= 9 – NUMBER

BUT SUBTRACTORS ARE NOT WIDELY AVAILABLE >>> WE GENERATE THE 9’s COMPLEMENT BY ADDING 1010 TO THE INVERTED NUMBER

<table>
<thead>
<tr>
<th>BCD DIGIT</th>
<th>Digit</th>
<th>Digit + 1010 = 9’s COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1111</td>
<td>1001</td>
</tr>
<tr>
<td>0001</td>
<td>1110</td>
<td>1000</td>
</tr>
<tr>
<td>0010</td>
<td>1101</td>
<td>0111</td>
</tr>
<tr>
<td>0011</td>
<td>1100</td>
<td>0110</td>
</tr>
<tr>
<td>0100</td>
<td>1011</td>
<td>0101</td>
</tr>
<tr>
<td>0101</td>
<td>1010</td>
<td>0100</td>
</tr>
<tr>
<td>0110</td>
<td>1001</td>
<td>0011</td>
</tr>
<tr>
<td>0111</td>
<td>1000</td>
<td>0010</td>
</tr>
<tr>
<td>1000</td>
<td>0111</td>
<td>0001</td>
</tr>
<tr>
<td>1001</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

WE IGNORE THE CARRY OUT
EXERCISE: DESIGN A 9’s COMPLEMENT GENERATOR FOR A BCD DIGIT:

![Diagram of 4-bit adder with 9's complement generator]
OCTAL & HEXADECIMAL NUMBERS

TWO MORE NUMBER SYSTEMS

OCTAL >>> BASE 8   E.G. 3₈

HEXADECIMAL >>> BASE 16   E.G. 5₁₆

THE OCTAL NUMBER SYSTEM IS COMPOSED OF 8 DIGITS:

    0, 1, 2, 3, 4, 5, 6, 7

TO COUNT ABOVE 7, WE BEGIN ANOTHER COLUMN AND START OVER:

    10, 11, 12, 13, 14, 15, 16, 17, 20, 21,…

>>>> COUNTING IN OCTAL IS SAME AS COUNTING IN DECIMAL EXCEPT ANY NUMBERS WITH 8 OR 9 WHICH ARE OMITTED.
THE HEXADECIMAL SYSTEM IS COMPOSED OF 16 DIGITS AND CHARACTERS. EACH HEXADECIMAL CHARACTER REPRESENTS a 4-BIT BINARY NUMBER

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>OCTAL</th>
<th>HEXADECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Decimal</td>
<td>Octal</td>
<td>Remainder</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>32</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

**DECIMAL >>> OCTAL**

**DIVIDE BY 8, GET REMAINDERS AND READ UP.**

\[
\begin{align*}
123_{10} : 8 & \\
15 & \quad \downarrow \quad 3 & \quad \text{READ UP} \\
1 & \quad \downarrow \quad 7 \\
0 & \quad \downarrow \quad 1 \\
\end{align*}
\]

\[\text{>>> } 123_{10} = 173_8\]
(b) \[ 100_{10} : 8 \]
\[
\begin{array}{c|c}
12 : 8 & 4 \\
1 : 8 & 4 \\
0 & 1 \\
\end{array}
\]

>>> \[ 100_{10} = 144_{8} \]

**OCTAL >>> DECIMAL**

MULTIPLY BY SUCCESSIVE POWERS OF 8

\[ 173_{8} = 1 \times 8^{2} + 7 \times 8^{1} + 3 \times 8^{0} \]
\[ = 64 + 56 + 3 = 123_{10} \]

**NOTE:** WE BEGIN WITH RIGHT-MOST POWER OF 8 AS \( 8^{0} \)

**DECIMAL >>> HEXADECIMAL**

DIVIDE BY 16, GET REMAINDERS AND READ UP

**NOTE:** REMAINDER = 13

>>> USE CHARACTER D
E.G. \[123_{10} : 16\]
\[\begin{align*}
7 & : 16 \quad (11) \leftarrow B \\
0 & \quad 7
\end{align*}\]

>>> \[123_{10} = 7B_{16}\]

**HEXADECIMAL >>> DECIMAL**

MULTIPLY BY SUCCESSIVE POWERS OF 16

**NOTE: WE CONVERT A CHARACTER TO ITS DECIMAL VALUE BEFORE WE MULTIPLY**

E.G. \[6A_{16} = 6 \times 16^1 + A \times 16^0\]
\[= 6 \times 16^1 + 10 \times 16^0\]
\[= 96 + 10 = 106_{10}\]
SEQUENTIAL LOGIC

UNTIL NOW WE HAVE DEALT WITH COMBINATIONAL LOGIC – WHERE THE OUTPUT DEPENDED ON THE PRESENT COMBINATION OF INPUTS

IN SEQUENTIAL LOGIC THE OUTPUT DEPENDS ON THE PAST INPUTS AS WELL AS PRESENT INPUTS

E.G. MEMORY DEVICES USED TO REMEMBER PAST VALUES OF INPUTS

FLIP-FLOPS

DIGITAL MEMORIES, COUNTERS AND SHIFT REGISTERS ARE BASED ON THE FLIP-FLOP. THIS IS A DEVICE THAT CAN STORE 2 STATES 0 OR 1
**R-S FLIP-FLOP**

The basic RS [reset- set] flip-flop consists of 2 nor gates with the outputs cross-coupled to the inputs.

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No change</td>
<td></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>Not allowed</td>
<td></td>
</tr>
</tbody>
</table>

Start with a known state:

- \(R = 0, S = 1 \implies Q = 1\) set
- \(R = 1, S = 0 \implies Q = 0\) reset
WHEN $R=S=0$, $Q$ AND $\overline{Q}$ REMAIN AS THEY WHERE BEFORE THE INPUT WAS CHANGED

$R=S=1$ IS NOT ALLOWED AS IT CAUSES $Q=\overline{Q}=0$ >>> THEY HAVE TO BE OPPOSITE TO EACH OTHER

APPLICATION: MEMORY CELL

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$Q_N$</th>
<th>$\overline{Q}_N$</th>
<th>$Q_{N+1}$</th>
<th>$\overline{Q}_{N+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$N =$ PREVIOUS STAGE (PAST)
$N+1 =$ PRESENT STAGE

THE CIRCUIT IS SAID TO BE **BISTABLE**
>>> IT CAN ASSUME TWO STABLE STATES.
RS FLIP-FLOP USING NAND GATES

** TWO CROSS-COUPLED NAND GATES

![NAND gates diagram]

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( \overline{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>NOT ALLOWED</td>
<td>SET</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>NO CHANGE</td>
<td></td>
</tr>
</tbody>
</table>

ACTIVE LOW RS

THE RS FLIP-FLOP CAN BE USED TO ELLIMINATE “CONTACT BOUNCES” IN SWITCHES.
ELECTRICAL CONTACTS OFTEN PROVIDE INPUT SIGNALS TO LOGIC SYSTEMS >>> THESE CONTACTS MAY BE A SOURCE OF NOISE DUE TO THEIR MECHANICAL PROPERTIES.

SWITCH MOVES FROM 1 TO 2 Q CHANGES FROM 0 TO 1 AS SOON AS THE FIRST CONTACT IS MADE (EVEN IF THE SWITCH BOUNCES Q STAYS AT 1)

WHEN THE SWITCH MOVES FROM 2 TO 1 Q CHANGES FROM 1 TO 0 AS SOON AS THE FIRST CONTACT IS MADE.
SYNCHRONOUS [CLOCKED] RS FLIP-FLOP

WE OFTEN NEED TO SET OR RESET A FLIP-FLOP UNDER THE CONTROL OF A CLOCK PULSE

\[
S_n \quad S' \quad Q \\
R_n \quad R' \quad \overline{Q}
\]

REMEMBER: \( S' = R' = 1 \) NO OUTPUT CHANGE

WHEN CLOCK APPLIED

<table>
<thead>
<tr>
<th>( S_N )</th>
<th>( R_N )</th>
<th>( Q_{n+1} )</th>
<th>( \overline{Q}_{n+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( Q_n )</td>
<td>( \overline{Q}_n )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>NOT ALLOWED</td>
<td></td>
</tr>
</tbody>
</table>

\( n = \text{TIME BEFORE THE CLOCK PULSE} \)
\( n+1 = \text{TIME AFTER THE CLOCK PULSE} \)
HERE THE FLIP-FLOP IS SET OR RESET UNDER THE CONTROL OF THE CLOCK PULSE

\[ \text{CLOCK PULSE} \]

\[ n \quad n+1 \]

WHEN CLK IS AT “0” NO CHANGE IN OUTPUTS IRRESPECTIVE OF THE STATE OF THE INPUTS (S,R).

**D FLIP-FLOP**

D [DELAY] FLIP-FLOP IS A SIMPLE EXTENSION OF THE CLOCKED RS FLIP-FLOP WITH THE D INPUT CONNECTED TO THE “SET” INPUT AND \( \overline{D} \) CONNECTED TO THE “RESET” INPUT.
D=1 >>> SET
D=0 >> RESET

ACTS AS 1-BIT DTORAGE ELEMENT
INPUT TRANSFERRED TO OUTPUT
>>> CALLED A LATCH.
J-K FLIP-FLOP

J-K FLIP-FLOP ELIMINATES NON-DEFINED (NOT ALLOWED) STATE THAT OCCURRED IN THE RS FLIP-FLOP (R=S=1).

MOST VERSATILE AND WIDELY USED.

<table>
<thead>
<tr>
<th>(Q_n)</th>
<th>J</th>
<th>K</th>
<th>(Q_{n+1})</th>
</tr>
</thead>
<tbody>
<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>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NO CHANGE
RESET
SET
TOGGLE
NO CHANGE
RESET
SET
TOGGLE

WHEN CLOCK ACTIVE
DIFFERS TO R-S BY USING ADDITIONAL FEEDBACK

WHEN CLOCK ACTIVE

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>$Q_{n+1}$</th>
<th>$Q_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>NO CHANGE</td>
<td>$Q_n$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>RESET</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SET</td>
<td>$Q_n$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>TOGGLE</td>
<td>$Q_n$</td>
</tr>
</tbody>
</table>

SAME AS R-S FLIP-FLOP EXCEPT WHEN $J=K=1$. 
E.G. \( Q=0 \ [\overline{Q}=1] \) >>> RESET

THEN CLOCK GOES FROM 0 \( \rightarrow \) 1

>>> \( A \rightarrow 0 \) >>> \( Q=1 \)

\( B \rightarrow 1 \) >>> \( Q=1 \)

>>> SET

MASTER SLAVE J-K FLIP-FLOP

PROBLEM WITH J-K FLIP-FLOP >>>

OUTPUTS MAY CHANGE IF THE DATA INPUTS \([J,K,Q,\overline{Q}]\) CHANGE WHILE THE CLOCK IS HIGH

>>> WE NEED TO MAKE SURE THAT \( Q \) AND \( \overline{Q} \) DO NOT ASSUME THEIR NEW VALUES UNTIL AFTER THE TRAILING EDGE ON THE CLOCK PULSE i.e. WHEN PULSE \( \rightarrow 0 \)
WE DO THIS BY USING A MASTER-SLAVE [M-S] FLIP-FLOP >>> DIVIDE THE FLIP-FLOP INTO MASTER + SLAVE FLIP-FLOP.

![Diagram of Master-Slave Flip-Flop]

WHEN THE CLOCK PULSE IS HIGH THE J-K INPUTS ARE APPLIED TO THE MASTER FLIP-FLOP AND THE SLAVE INPUT IS NOT AFFECTED (CLK=0)

WHEN THE CLOCK PULSE FALLS TO ZERO, THE DATA FROM THE MASTER IS APPLIED TO THE SLAVE INPUTS AND THE OUTPUTS Q AND Q GET THEIR NEW VALUES.

ANOTHER RESTRICTION: THE INPUTS J AND K DO NOT CHANGE WHILE CLK=1
A **R-S** MASTER SLAVE FLIP-FLOP MAY ALSO BE CONSTRUCTED IN A SIMILAR FASHION, THE ONLY DIFFERENCE IS THAT WHEN THE TWO INPUTS J=K=1 THE OUTPUT \( Q \) CHANGES IS STATE ON RECEIPT OF THE CLOCK PULSE i.e. THE FLIP-FLOP TOGGLES.

\[
\begin{array}{|c|c|c|}
\hline
J & K & Q_{n+1} \\
\hline
0 & 0 & Q_n \quad \text{UNCAHGED} \\
0 & 1 & 0 \quad \text{RESET} \\
1 & 0 & 1 \quad \text{SET} \\
1 & 1 & \overline{Q_n} \quad \text{TOGGLE} \\
\hline
\end{array}
\]

**DIFFERENT APPROACH:**

WE WILL NOW LOOK AT THE J-K FLIP FLOP IN A SLIGHTLY DIFFERENT WAY

IF \( Q_n = 0 \) AND J = 0 \( \implies \) \( Q_{n+1} = 0 \)
REGARDLESS OF K

IF \( Q_n = 0 \) AND J = 1 \( \implies \) \( Q_{n+1} = 1 \)
REGARDLESS OF K
IF $Q_n = 1$ AND $K = 0$ $\implies Q_{n+1} = 1$
REGARDLESS OF $J$

IF $Q_n = 1$ AND $K = 1$ $\implies Q_{n+1} = 0$
REGARDLESS OF $J$

<table>
<thead>
<tr>
<th>$J$</th>
<th>$K$</th>
<th>$Q_n$</th>
<th>$Q_{n+1}$</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

“$X$” $\rightarrow$ DEFINES A DON’T CARE TERM
$\rightarrow$ USEFUL IN DESIGNING LOGIC SYSTEMS WITH J-K FLIP FLOPS

THE MASTER-SLAVE FLIP-FLOPS ARE SYNCRONOUS DEVICES $\implies$ THE OUTPUT CHANGES STATE AT A SPECIFIED POINT ON A CLOCK INPUT.
J-K FLIP-FLOP WITH ASYNCHRONOUS SET AND RESET INPUTS

SEPARATE “SET” AND “RESET” INPUTS MAY BE ADDED TO THE J-K FLIP FLOP.

THESE SET & RESET INPUTS [\(\bar{S}, \bar{R}\)] OVER-RIDE THE J,K INPUTS AND ARE ASYNCHRONOUS [i.e. OVER-RIDE THE CLOCK ALSO].

NOTE: WHEN \(\bar{S} = \bar{R} = 1\) >>> WE HAVE NORMAL FLIP-FLOP OPERATION
**TRUTH TABLE:**

<table>
<thead>
<tr>
<th>$\bar{S}$</th>
<th>$\bar{R}$</th>
<th>J</th>
<th>K</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>NOT ALLOWED</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Q_n$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\bar{Q}_n$</td>
</tr>
</tbody>
</table>

NORMAL OPERATION
FLIP-FLOP APPLICATIONS

REGISTERS:

GROUPS OF FLIP-FLOPS ARRANGED TO PROVIDE DATA STORAGE FOR SEVERAL DATA BITS.

1. DATA REGISTER

A J-K FLIP-FLOP CAN BE USED AS A 1-BIT MEMORY [D FLIP-FLOP] BY APPLYING THE BIT TO BE STORED [D] TO J AND ITS INVERSE [D'] TO K. AN “n” BIT BINARY WORD CAN BE STORED BY “n” SUCH FLIP-FLOPS >>>> CALLED A n-BIT REGISTER.

PARALLEL DATA STORAGE, i.e. SEVERAL BITS ON PARALLEL LINES STORED SIMULTANEOUSLY.
DATA BITS ($D_3 \rightarrow D_0$) ARE STORED ON THE RECEIPT OF THE CLOCK PULSE.

2. SHIFT REGISTER

SERIAL DATA TRANSFER [i.e. ONE BIT AT A TIME], A WORD OF N-BITS READ INTO N-FLIP-FLOP SHIFT REGISTER

THE FIRST FLIP-FLOP HOLDS THE WORD’S MSB AND IS CONNECTED AS A D FLIP-FLOP.
E.G. 4 –BIT SHIFT REGISTER

Q3 = MSB

REMEMBER:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

OUTPUT FOLLOWS J-INPUT WHEN J = \overline{K}

>>> SHIFT OCCURS WHEN CLOCK APPLIED.
SHIFT REGISTER OPERATION:

LSB OF SERIAL DATA IS APPLIED TO THE SERIAL INPUT \([J_3]\). THE CLOCK PULSE IS APPLIED AND LSB GOES INTO \(Q_3\) \([= J_2]\). THE NEXT LSB IS THEN APPLIED TO \(J_3\) AND AFTER A CLOCK PULSE IT IS STORED IN \(Q_3\), WHILE AT THE SAME TIME THE LSB IS SHIFTED INTO \(Q_2\).

AFTER 4 CLOCK PULSES THE SERIAL DATA IS STORED IN THE SHIFT REGISTER.

E.G. STORE THE WORD: 0 1 0 1

<table>
<thead>
<tr>
<th>CLK</th>
<th>SERIAL INPUT</th>
<th>Q3</th>
<th>Q2</th>
<th>Q1</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

ASSUME THE STORED WORD IS INITIALLY 0 0 0 0.
3. **SHIFT REGISTERS WITH PARALLEL INPUTS**

[PARALLEL TO SERIAL CONVERTER]

- **LOAD BITS IN PARALLEL FORM AND SHIFT THEM IN SERIAL FORM.**

- **DATA IS LOADED BY APPLYING SIGNALS TO ASYNCHRONOUS SET/RESET INPUTS OF THE J-K FLIP-FLOP.**

![Shift Register Diagram]

PARALLEL DATA IS LOADED BY APPLYING A CLOCK PULSE TO “PARALLEL LOAD” SIGNAL.
WHEN “PARALLEL LOAD” IS LOW THE CIRCUIT OPERATES AS A NORMAL SHIFT REGISTER \([\bar{S} = \bar{R} = 1]\)

WHEN “PARALLEL LOAD” IS HIGH THEN PARALLEL DATA IS LOADED ASYNCRONOUSLY [INDEPENDENT OF THE CLOCK] AND SHIFTED OUT SYNCHRONOUSLY.

4. RIGHT-LEFT SHIFT REGISTER

SHIFT REGISTERS CAN ALSO BE USED TO TRANSFER DATA FROM RIGHT TO LEFT, SHIFT LEFT, BY CONNECTING THE OUTPUT OF A FLIP-FLOP BACK TO THE INPUT OF THE FLIP-FLOP ON ITS LEFT

A SHIFT FROM LEFT TO RIGHT, SHIFT RIGHT, CAN BE CARRIED OUT DURING NORMAL OPERATION OF THE SHIFT REGISTER.
A shift right and shift left register can be combined by suitable gating and control signals.

Note $\overline{R/L} >>> \overline{R/L} = 0 >>> \text{shift right}$
$\overline{R/L} = 1 >>> \text{shift left}$