Combinational Arithmetic Circuits

• Addition:
  – Half Adder (HA).
  – Full Adder (FA).
  – Carry Ripple Adders.
  – Carry Look-Ahead Adders.

• Subtraction:
  – Half Subtractor.
  – Full Subtractor.
  – Borrow Ripple Subtractors.
  – Subtraction using adders.

• Multiplication:
  – Combinational Array Multipliers.
Half Adder

- Adding two single-bit binary values, X, Y produces a sum S bit and a carry out C-out bit.
- This operation is called half addition and the circuit to realize it is called a half adder.

### Half Adder Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- $S(X,Y) = \sum (1,2)$
- $S = X'Y + XY'$
- $S = X \oplus Y$

- $C-out(x, y, C-in) = \sum (3)$
- $C-out = XY$

![Half Adder Circuit Diagram]
Full Adder

- Adding two single-bit binary values, X, Y with a carry input bit C-in produces a sum bit S and a carry out C-out bit.

**Full Adder Truth Table**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Sum S**

\[
S = X'Y'(C-in) + XY'(C-in)' + XY'(C-in)' + XY(C-in)
\]

\[
S = X \oplus Y \oplus (C-in)
\]

**Carry C-out**

\[
C-out = XY + X(C-in) + Y(C-in)
\]

\[
C-out = XY + X(C-in) + Y(C-in)
\]
Full Adder Circuit Using AND-OR

X \rightarrow X'
Y \rightarrow Y'
C-in \rightarrow C-in'

C-out \leftarrow \text{Full Adder} \rightarrow C-in

X \rightarrow Y \rightarrow S

X'Y'C-in
XY'C-in'
X'Y'C-in'
XY'C-in'

Sum S

C-out

XC-in
YC-in

EECC341 - Shaaban

#4  Lec # 11  Winter 2001  1-16-2002
Full Adder Circuit Using XOR

X Y C-in S C-out

X Y
C-in

Sum S

X Y
C-in

XC-in

YC-in

C-out
An n-bit adder used to add two n-bit binary numbers can be built by connecting in series n full adders.

- Each full adder represents a bit position j (from 0 to n-1).
- Each carry out C-out from a full adder at position j is connected to the carry in C-in of the full adder at the higher position j+1.

The output of a full adder at position j is given by:

\[ S_j = X_j \oplus Y_j \oplus C_j \]

\[ C_{j+1} = X_j \cdot Y_j + X_j \cdot C_j + Y \cdot C_j \]

- In the expression of the sum C_j must be generated by the full adder at the lower position j-1.
- The propagation delay in each full adder to produce the carry is equal to two gate delays = 2Δ
- Since the generation of the sum requires the propagation of the carry from the lowest position to the highest position, the total propagation delay of the adder is approximately:

\[ \text{Total Propagation delay} = 2nΔ \]
4-bit Carry Ripple Adder

Adds two 4-bit numbers:
\[ X = X_3 \ X_2 \ X_1 \ X_0 \]
\[ Y = Y_3 \ Y_2 \ Y_1 \ Y_0 \]
producing the sum \[ S = S_3 \ S_2 \ S_1 \ S_0 \]
\[ C\text{-out} = C_4 \] from the most significant position \( j=3 \)

Total Propagation delay \( = 2 \ n\Delta = 8\Delta \)
or 8 gate delays

Data inputs to be added

\[ \begin{align*}
C_4 & \quad \text{Full Adder} \quad X_3 \quad Y_3 \\
S_3 & \quad \text{C\text{-out}} \\
C_3 & \quad \text{C\text{-in}} \\
\end{align*} \]

\[ \begin{align*}
C_0 = 0 & \quad \text{Full Adder} \\
\text{Sum output} & \quad X_0 \quad Y_0 \\
\end{align*} \]

\[ \begin{align*}
C_4 & \quad \text{Full Adder} \quad X_2 \quad Y_2 \\
S_2 & \quad \text{C\text{-out}} \\
C_2 & \quad \text{C\text{-in}} \\
\end{align*} \]

\[ \begin{align*}
C_1 & \quad \text{Full Adder} \quad X_1 \quad Y_1 \\
S_1 & \quad \text{C\text{-out}} \\
C_1 & \quad \text{C\text{-in}} \\
\end{align*} \]

\[ \begin{align*}
C_0 = 0 & \quad \text{Full Adder} \\
\text{Sum output} & \quad X_0 \quad Y_0 \\
\end{align*} \]
Larger Adders

- Example: 16-bit adder using 4, 4-bit adders
- Adds two 16-bit inputs \(X\) (bits \(X_0\) to \(X_{15}\)), \(Y\) (bits \(Y_0\) to \(Y_{15}\)) producing a 16-bit Sum \(S\) (bits \(S_0\) to \(S_{15}\)) and a carry out \(C_{16}\) from most significant position.

Data inputs to be added \(X\) (\(X_0\) to \(X_{15}\)), \(Y\) (\(Y_0\) to \(Y_{15}\))

Sum output \(S\) (\(S_0\) to \(S_{15}\))

Propagation delay for 16-bit adder = \(4 \times \text{propagation delay of 4-bit adder}\)

\[= 4 \times 2^n\Delta = 4 \times 8\Delta = 32\Delta\]

or 32 gate delays
Carry Look-Ahead Adders

- The disadvantage of the ripple carry adder is that the propagation delay of adder \((2 \cdot n \Delta)\) increases as the size of the adder, \(n\) is increased due to the carry ripple through all the full adders.

- Carry look-ahead adders use a different method to create the needed carry bits for each full adder with a lower constant delay equal to three gate delays.

- The carry out \(C\)-out from the full adder at position \(i\) or \(C_{j+1}\) is given by:

\[
C\text{-out} = C_{i+1} = X_i \cdot Y_i + (X_i + Y_i) \cdot C_i
\]

- By defining:
  - \(G_i = X_i \cdot Y_i\) as the carry generate function for position \(i\) (one gate delay)  
    (If \(G_i = 1\), \(C_{i+1}\) will be generated regardless of the value \(C_i\))
  - \(P_i = X_i + Y_i\) as the carry propagate function for position \(i\) (one gate delay)  
    (If \(P_i = 1\), \(C_i\) will be propagated to \(C_{i+1}\))

- By using the carry generate function \(G_i\) and carry propagate function \(P_i\), then \(C_{i+1}\) can be written as:

\[
C\text{-out} = C_{i+1} = G_i + P_i \cdot C_i
\]

- To eliminate carry ripple the term \(C_i\) is recursively expanded and by multiplying out, we obtain a 2-level AND-OR expression for each \(C_{i+1}\)
For a 4-bit carry look-ahead adder the expanded expressions for all carry bits are given by:

\[ C_1 = G_0 + P_0 \cdot C_0 \]

\[ C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \]

\[ C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0 \]

\[ C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0 \]

where     \[ G_i = X_i \cdot Y_i \]
            \[ P_i = X_i + Y_i \]

The additional circuits needed to realize the expressions are usually referred to as the carry look-ahead logic.

Using carry-ahead logic all carry bits are available after three gate delays regardless of the size of the adder.
C_i = G_{i-1} + P_{i-1} \cdot G_{i-2} + \ldots + P_{i-1} \cdot P_{i-2} \cdot \ldots P_1 \cdot G_0 + P_{i-1} \cdot P_{i-2} \cdot \ldots P_0 \cdot C_0
Binary Arithmetic Operations

Subtraction

- Two binary numbers are subtracted by subtracting each pair of bits together with borrowing, where needed.
- Subtraction Example:

\[
\begin{array}{cccccccccccc}
X & 229 & & & & & & & & & & \text{Borrow} \\
\hline
Y & - & 46 & & & & & & & & & \\
\hline
183 & - & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{cccccccccccc}
& & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\hline
& & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
\hline
& & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
& & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]
Half Subtractor

- Subtracting a single-bit binary value \( Y \) from another \( X \) (i.e. \( X - Y \)) produces a difference bit \( D \) and a borrow out bit \( B\text{-out} \).
- This operation is called half subtraction and the circuit to realize it is called a half subtractor.

### Half Subtractor Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( Y )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
D(X,Y) = \Sigma (1,2)
\]

\[
D = X'Y + XY'
\]

\[
D = X \oplus Y
\]

\[
B\text{-out}(x, y, C\text{-in}) = \Sigma (1)
\]

\[
B\text{-out} = X'Y
\]
Full Subtractor

- Subtracting two single-bit binary values, \( Y \), \( B\text{-in} \) from a single-bit value \( X \) produces a difference bit \( D \) and a borrow out \( B\text{-out} \) bit. This is called full subtraction.

**Full Subtractor Truth Table**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( Y )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Difference \( D \)**

\[
D = X'Y'(B\text{-in}) + XY'(B\text{-out})' + XY'(B\text{-in})' + XY(B\text{-in})
\]

**Borrow \( B\text{-out} \)**

\[
B\text{-out} = X'Y + X'(B\text{-in}) + Y(B\text{-in})
\]

\[
S(X,Y, C\text{-in}) = \Sigma (1,2,4,7)
\]

\[
C\text{-out}(x, y, C\text{-in}) = \Sigma (1,2,3,7)
\]
Full Subtractor Circuit Using AND-OR

\[ \begin{align*}
X'Y'B-IN' & \quad \text{Difference D} \\
XYB-IN & \quad \text{B-in}
\end{align*} \]
Full Subtractor Circuit Using XOR

Full Subtractor

X
Y
B-in

D

X
Y
B-in

B-out

X'
Y
B-in

X'
Y
B-in

B-out

X'
Y
B-in

Y
B-in

Difference D

B-out
n-bit Subtractors

An n-bit subtractor used to subtract an n-bit number Y from another n-bit number X (i.e. X-Y) can be built in one of two ways:

- By using n full subtractors and connecting them in series, creating a borrow ripple subtractor:
  - Each borrow out B-out from a full subtractor at position j is connected to the borrow in B-in of the full subtractor at the higher position j+1.

- By using an n-bit adder and n inverters:
  - Find two’s complement of Y by:
    - Inverting all the bits of Y using the n inverters.
    - Adding 1 by setting the carry in of the least significant position to 1
  - The original subtraction (X - Y) now becomes an addition of X to two’s complement of Y using the n-bit adder.
4-bit Borrow Ripple Subtractor

Subtracts two 4-bit numbers:
\[ Y = Y_3 \ Y_2 \ Y_1 \ Y_0 \] from
\[ X = X_3 \ X_2 \ X_1 \ X_0 \]
yielding the difference \( D = D_3 \ D_2 \ D_1 \ D_0 \),
\( B\)-out = \( B_4 \) from the most significant position \( j=3 \)

Data inputs to be subtracted

\[ X_3 \ Y_3 \]
\[ X_2 \ Y_2 \]
\[ X_1 \ Y_1 \]
\[ X_0 \ Y_0 \]

\( B_4 \leftarrow \) Full Subtractor
\( D_3 \)
\( B_3 \leftarrow \) Full Subtractor
\( D_2 \)
\( B_2 \leftarrow \) Full Subtractor
\( D_1 \)
\( B_1 \leftarrow \) Full Subtractor
\( D_0 \)

\( B_0 = 0 \)

\( X_3 X_2 X_1 X_0 \)
\( Y_3 Y_2 Y_1 Y_0 \)

Input

4-bit Subtractor

B-out

\( D_3 \ D_2 \ D_1 \ D_0 \)

B-in

\( B_4 \leftarrow \)

\( B_0 = 0 \)

Difference Output \( D \)
4-bit Subtractor Using 4-bit Adder

Inputs to be subtracted

C4
C-out

4-bit Adder

C-in

C0 = 1

S3 S2 S1 S0

D3 D2 D1 D0

Difference Output
### Binary Multiplication

- Multiplication is achieved by adding a list of shifted multiplicands according to the digits of the multiplier.

- **Ex.** (unsigned)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>multiplicand (4 bits)</th>
<th></th>
<th></th>
<th></th>
<th>multiplier (4 bits)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1 0 1 1</td>
<td>X</td>
<td>1 1 0 1</td>
<td>x</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>1 0 1 1</td>
<td>X</td>
<td>3.0 1 1</td>
<td>X</td>
<td>3.0 1 1</td>
<td>X</td>
<td>3.0 1 1</td>
<td>X</td>
<td>3.0 1 1</td>
</tr>
<tr>
<td>11</td>
<td>0 0 0 0</td>
<td>x</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td></td>
</tr>
<tr>
<td>__</td>
<td>1 0 1 1</td>
<td>x</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td>Y</td>
<td>3.0 1 1</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>1 0 1 1</td>
<td>P7</td>
<td>P6</td>
<td>P5</td>
<td>P4</td>
<td>P3</td>
<td>P2</td>
<td>P1</td>
<td>P0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P7</td>
<td>P6</td>
<td>P5</td>
<td>P4</td>
<td>P3</td>
<td>P2</td>
<td>P1</td>
</tr>
</tbody>
</table>

- **Product (8 bits)**

- An \( n \)-bit \( X \) \( n \)-bit multiplier can be realized in combinational circuitry by using an array of \( n-1 \) \( n \)-bit adders where is adder is shifted by one position.

- For each adder one input is the multiplied by 0 or 1 (using AND gates) depending on the multiplier bit, the other input is \( n \) partial product bits.