Adder Circuits
Ivor Page[1]

3.1 The Ripple Carry Adder

The ripple carry adder is probably the simplest parallel binary adder. It is made up of k full-adder stages, where each full-adder can be conveniently made from two half-adders and an OR gate. The truth table of the half-adder is as follows:

Inputs		Outputs	
x_i	y_i	c_{i+1}	z_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 1 shows the block diagram of the adder.

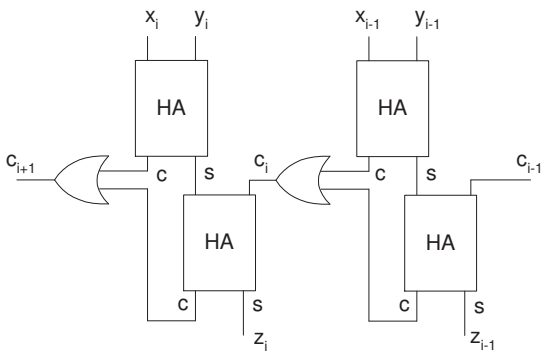


Figure 1: Ripple Carry Adder

Each half-adder can be constructed from four 2-input nand gates.

[1]University of Texas at Dallas

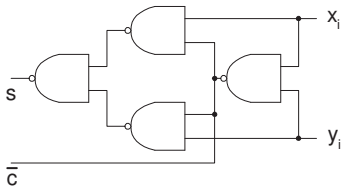


Figure 2: Half-Adder in Nand Gates

Figure 2 shows the logic diagram. The complement of the carry signal is produced, which is exactly what is needed if the inter-stage OR gate is replaced by a nand gate: $\overline{\overline{c_1} \cdot \overline{c_2}} = c_1 + c_2$.

Figure 3 is from Weste & Eshraghian[2]. It shows one of many possible CMOS implementations of a full-adder. This circuit is generally referred to as 28T since it has 28 transistors, including two for each inverter.

The delay through this circuit from x, y to c_{out} comes from the two series transistor pull-ups and pull-downs in the first stage and an inverter delay. For the *sum* output, there are two cascaded stages, each with 2 series transistor pull-ups and pull-downs, followed by an inverter.

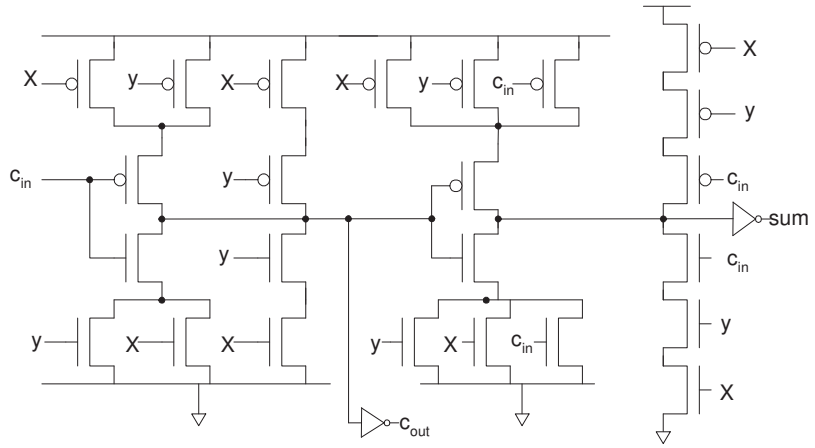


Figure 3: CMOS Full-Adder

Figure 4 shows one possible layout using simple transistor sequencing. It does not include the two inverters.

In this layout, dark blue is used for metal-1 lines and light blue for metal-2. P-Diffusion is yellow and N-diffusion is green. Polysilicon lines are pink. Each time a poly line crosses

[2]Principles of CMOS VLSI Design, Weste & Eshraghian, Addison Wesley

P-Diff, a PMOS transistor is formed, and each time it crosses N-Diff, an NMOS transistor is formed.

It is common to modify the 28T circuit to simplify the layout. To do so, disconnect the source of the top transistor from V_{ss} in the output chain for $\overline{sum}$ and connect it instead to the drains of the top three parallel transistors in the penultimate stage. Make a corresponding change at the bottom end of the final chain. This change enables a more compact layout which has shorter connections.

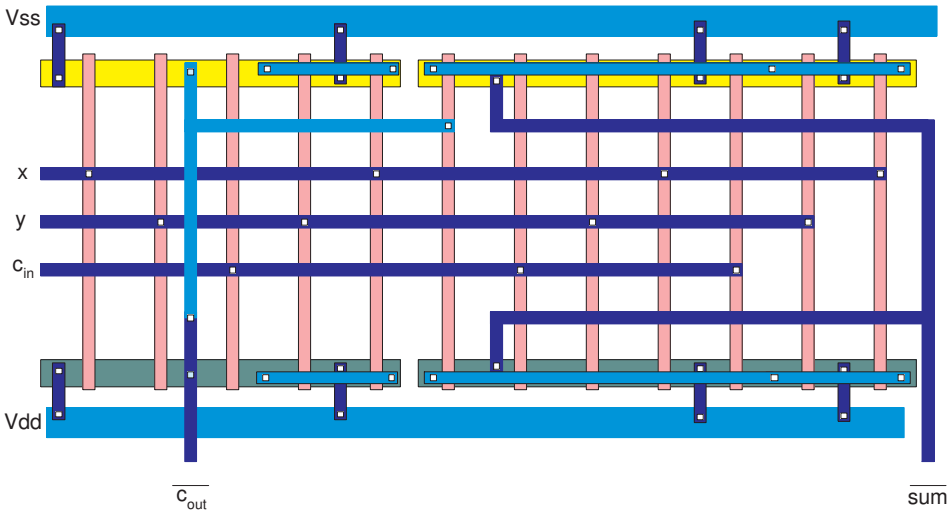


Figure 4: CMOS Layout of Full-Adder

Figure 5 shows the circuit of the adder arranged according to the layout of Figure 4.

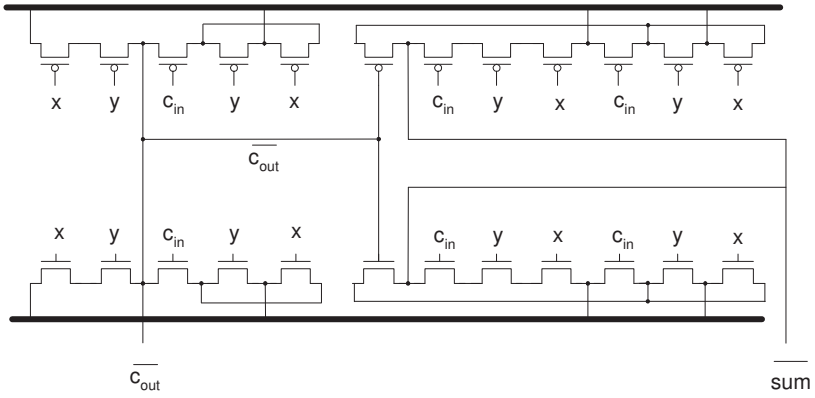


Figure 5: CMOS Scematic of Layout of Full-Adder

3.2 Delay Through Ripple Carry Adder

The worst case delay through a k stage ripple-carry adder, from inputs x_0, y_0 to output c_k, using nand-based circuits is $(2k+1)D$, where D is the normalized gate delay.

We consider the average delay since the probability of the worst case is very small and, assuming that it is possible to detect that carry propagation has terminated, the computer system need not wait for the worst case delay time on every addition. Unfortunately, taking advantage of a variable time adder would seriously complicate the design of a clocked system. It is more practical to base the design on worst case delay assumptions.

First we consider the mean carry-chain length, assuming that the two values being added have random bit-patterns. This assumption is clearly not true of most computer programs. Most integer calculations involve the addition or subtraction of small constants.

Consider a simple example, where the transpose of a matrix is being calculated:

```
// Matrix Transpose Example

   int in_matrix[4][4], out_matrix[4][4];
   . . .
   for(int i=0;i<4;i++)
      for(int j=0;j<4;j++)
         out_matrix[j][i] = in_matrix[i][j];
```

Although contrived, the example does reflect what takes place in real programs: much of the arithmetic activity is devoted to loop-counting and computing addresses. These operations require the addition and subtraction of small integer constants. The Burroughs B1700 computer was based on this premise. It enabled storage of, and arithmetic on, variable-length integers, reflecting the propensity of operations with small integer constants. Let's continue with the assumption that the two operands have random bit patterns. For each operand, every possible value within the range of the number system is equally likely. The following statements are then true for an arbitrary stage:

- Probability of carry generation = 1/4
- Probability of carry annihilation = 1/4
- Probability of carry propagation = 1/2

The first case corresponds to $x_i.y_i = 1$. The second requires $x_i + y_i = 0$, and the third requires $x_i \oplus y_i = 1$.

For a carry chain to begin at stage i and end at stage j, $j > i$, it must be generated by stage i, propagated by stages $i+1, i+2, \cdots j-1$, and stopped at stage j. To stop the carry chain, stage j must either annihilate the incoming carry, or generate a carry, thereby beginning new chain.

The probability that a carry generated at position i ends at position j, for $j > i$ is:

$$P_{carry\_chain}(j,i) = 2^{-(j-i-1)} \times 1/2 = 2^{-(j-i)} \quad j > i$$

The equation does not include the probability that stage i generates a carry. The term $1/2$ comes from the combined probabilities that stage j generates a carry or annihilates the incoming carry.

To obtain the mean length of a carry chain beginning at stage i we sum the lengths of all possible chains times their probabilities.

$$\begin{aligned}
&= \sum_{j=i+1}^{k-1} \left[(j-i)2^{-(j-i)}\right] + (k-i)2^{-(k-1-i)} \\
&= \sum_{l=1}^{k-1-i} \left[l2^{-l}\right] + (k-i)2^{-(k-1-i)} \\
&= 2 - (k-i-1)2^{-(k-1-i)} + (k-i)2^{-(k-1-i)} \\
&= 2 - 2^{-(k-i-1)}
\end{aligned}$$

The summation includes all chains that stop before or at stage $k-1$. The term $(k-i)2^{-(k-1-i)}$ is for a carry chain of length $k-i$, that must end since there is no stage k. The change of variable, $l = j - i$, in the second line facilitates the use of the theorem:

$$\sum_{l=1}^{p} \frac{l}{2^l} = 2 - \frac{(p+2)}{2^p}$$

For $k >> i$, the mean carry-chain length is 2, quite short. Now we consider the mean of the longest carry-chain lengths when adding pairs of arbitrary integers. That is, if we add one million pairs of randomly generated integers and tabulate the length of the longest carry-chain in each of these sums, what will be the mean of those tabulated values?

Let $\eta_k(h)$ be the probability that the longest carry-chain in a k bit addition is of length h or more. The probability that the longest carry chain is exactly of length h is $\eta_k(h) - \eta_k(h+1)$. We can form a recurrence relation by using the following conditions that enable a carry chain of length h or more:

c1: The least significant $k-1$ bits have a carry length of h or more.

c2: The least significant $k-1$ bits do not have such a carry chain, but the most significant h bits, including the last bit, have a carry chain of length exactly h.

Conditions c1 and c2 are mutually exclusive.

$$\text{In general: } \eta_k(h) = \eta_{k-1}(h) + 2^{-(h+1)} \times P(condition\ 2)$$

The term $2^{-(h+1)}$ represents the probability of a carry being generated at stage $k-h$, times the probability of that carry being propagated over (the most significant) $h-2$ stages.

Since we cannot know the probability that the second condition will occur, we set its probability to 1, change the equals sign to less-than-or-equals, and unwind the recurrence:

$$\begin{aligned}
\eta_k(h) &\leq \eta_{k-1}(h) + 2^{-(h+1)} \\
&\leq \eta_{k-2}(h) + 2 \times 2^{-(h+1)} \\
&\leq \eta_{k-3}(h) + 3 \times 2^{-(h+1)} \\
&\leq \eta_{k-4}(h) + 4 \times 2^{-(h+1)} \\
&\ldots
\end{aligned}$$

Then, since $\eta_i(h) = 0$ for $i < h$,

$$\begin{aligned}
\eta_k(h) &\leq (k-h+1)2^{-(h+1)} \\
&\leq k2^{-(h+1)}
\end{aligned}$$

Note that $\eta_k(h)$ is a probability, $\eta_k(h) \leq 1$, but the RHS of the above inequality is ≥ 1 for $h \leq \lfloor log_2\ k \rfloor - 1$.

To compute the mean of the longest carry-chain, λ, we sum the lengths times their probabilities:

$$\begin{aligned}
\lambda &= \sum_{h=1}^{k} h\left[\eta_k(h) - \eta_k(h+1)\right] \\
&= [\eta_k(1) - \eta_k(2)] + 2[\eta_k(2) - \eta_k(3)] + \cdots + k[\eta_k(k) - 0] \\
&= \sum_{h=1}^{k} \eta_k(h)
\end{aligned}$$

Next we partition the sum into two parts. The first $\gamma = \lfloor log_2\ k \rfloor - 1$ terms and the remaining $k - \gamma$ terms. We set the values of the first γ terms to 1, the largest they can be since they are probabilities. Each of the remaining terms is bounded above by $k2^{-(h+1)}$.

$$\lambda = \sum_{h=1}^{k} \eta_k(h) \leq \sum_{h=1}^{\gamma} 1 + \sum_{h=\gamma+1}^{k} k2^{-(h+1)} < \gamma + k2^{-(\gamma+1)}$$

Now let $\epsilon = log_2\ k - \lfloor log_2\ k \rfloor$ or $\gamma = log_2\ k - 1 - \epsilon$, where $0 \leq \epsilon < 1$, then, noting $2^{log_2\ k} = k$ and $2^{\epsilon} < 1 + \epsilon$,

$$\lambda < log_2\ k - 1 - \epsilon + 2^{\epsilon} < log_2\ k$$

This concludes the proof that the mean delay through a ripple-carry adder is $O(log\ k)$.

To make use of this information, it would be necessary to add logic to the adder to detect when the output was stable, i.e. when all the carry propagation had been completed. **Carry-Completion Logic** provides this facility.

In the Carry Completion Logic, two signals are generated in each stage in addition to the sum and carry signals produced by the full adders:

b_i	c_i	Comment
0	0	Carry not yet known
0	1	Carry known to be 1
1	0	Carry known to be 0

For stages where the inputs $(x_i, y_i) = (0, 0)$ or $(1, 1)$, $(b_{i+1}, c_{i+1}) = (\overline{x_i + y_i}, x_i y_i)$.

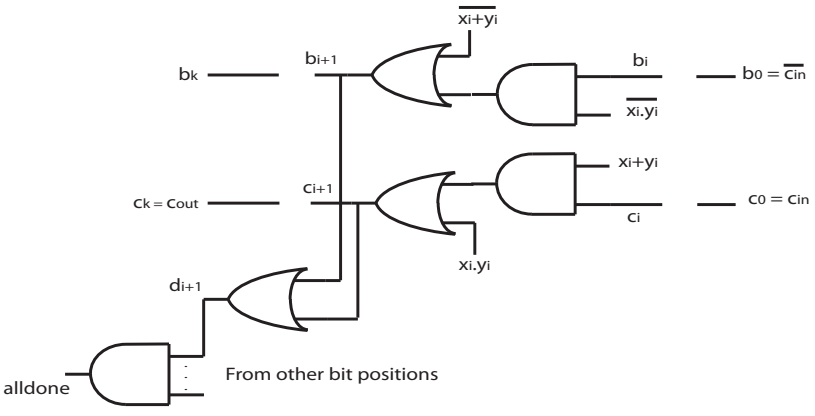
For the remaining stages, $(b_{i+1}, c_{i+1}) = (b_i \overline{x_i y_i}, c_i(x_i + y_i))$

At the right hand end of the adder, $b_0 = \overline{c_{in}}$ and $c_0 = c_{in}$. When every b_i, c_i pair is 1,0 or 0,1, all carries are assumed to be complete. An OR gate at each stage generates $d_i = b_{i+1} + c_{i+1}$ and a global *alldone* signal is generated from the logical AND of all the d_i signals.

Combining these equations we get:

$$\begin{aligned} b_0 &= \overline{c_{in}} \\ c_0 &= c_{in} \\ b_{i+1} &= \overline{x_i + y_i} + b_i \overline{x_i y_i} \\ c_{i+1} &= x_i y_i + c_i(x_i + y_i) \\ d_i &= b_i + c_i \\ alldone &= \prod_{i=1,k} d_i \end{aligned}$$

Here is the logoc diagram for stage i.



The latency of the carry-completion logic is the same as the latency of the adder. It ranges from constant time if no carries are propagated to $(2k + 1)$ gate delays in the worst case.

3.3 Manchester Carry Chain

As we saw in the above proof, each stage can generate, propagate, or annihilate a carry:

$$\begin{aligned} g_i &= x_i y_i \\ p_i &= x_i \oplus y_i \\ a_i &= \overline{x_i + y_i} \\ t_i &= \overline{a_i} = x_i + y_i \end{aligned}$$

We have added a *transfer* signal t_i that can be used in preference to p_i since it is easier and quicker to generate.

From these equations a simple recurrence results:

$$c_{i+1} = g_i + c_i p_i = g_i + c_i t_i$$

In the carry-lookahead adder, the recurrence is unwound m times to form a block carry system. In the Manchester carry chain, the equation is implemented by three switches per stage:

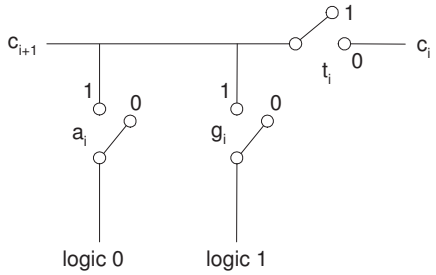


Figure 6: Manchester Carry Chain

If this circuit is implemented with pass transistors and a buffer gate is used after every m stages, the delay would be proportional to $(k/m)m^2$.