Addition Operation

In this section, we study algorithms for computing the sum of two k-bit integers A and B. Let A_i and B_i for $i = 1, 2, \ldots, k-1$ represent the bits of the integers A and B, respectively. We would like to compute the sum bits S_i for $i = 1, 2, \ldots, k-1$ and the final carry-out C_k as follows:

$$\begin{array}{cccccc} & A_{k-1} & A_{k-2} & \cdots & A_1 & A_0 \\ + & B_{k-1} & B_{k-2} & \cdots & B_1 & B_0 \\ \hline C_k & S_{k-1} & S_{k-2} & \cdots & S_1 & S_0 \end{array}$$

We will study the following algorithms: the carry propagate adder (CPA), the carry completion sensing adder (CCSA), the carry look-ahead adder (CLA), the carry save adder (CSA), and the carry delayed adder (CDA) for computing the sum and the final carry-out.

Full-Adder and Half-Adder Cells

The building blocks of these adders are the full-adder (FA) and half-adder (HA) cells. Thus, we briefly introduce them here. A full-adder is a combinational circuit with 3 input and 2 outputs. The inputs A_i, B_i, C_i and the outputs S_i and C_{i+1} are boolean variables. It is assumed that A_i and B_i are the ith bits of the integers A and B, respectively, and C_i is the carry bit received by the ith position. The FA cell computes the sum bit S_i and the carry-out bit C_{i+1} which is to be received by the next cell. The truth table of the FA cell is as follows:

A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The boolean functions of the output values are as

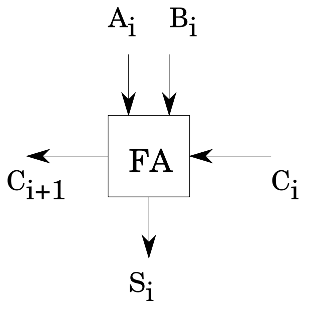
$$\begin{aligned} C_{i+1} &= A_iB_i + A_iC_i + B_iC_i \ , \\ S_i &= A_i \oplus B_i \oplus C_i \ . \end{aligned}$$

Similarly, an half-adder is a combinational circuit with 2 inputs and 2 outputs. The inputs A_i, B_i and the outputs S_i and C_{i+1} are boolean variables. It is assumed that A_i

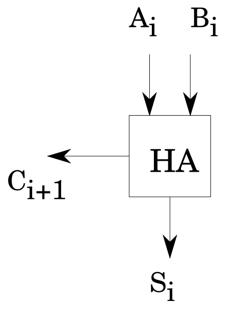
and B_i are the ith bits of the integers A and B, respectively. The HA cell computes the sum bit S_i and the carry-out bit C_{i+1}. Thus, an half-adder is easily obtained by setting the third input bit C_i to zero. The truth table of the HA cell is as follows:

A_i	B_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The boolean functions of the output values are as $C_{i+1} = A_i B_i$ and $S_i = A_i \oplus B_i$, which can be obtained by setting the carry bit input C_i of the FA cell to zero. The following figure illustrates the FA and HA cells.



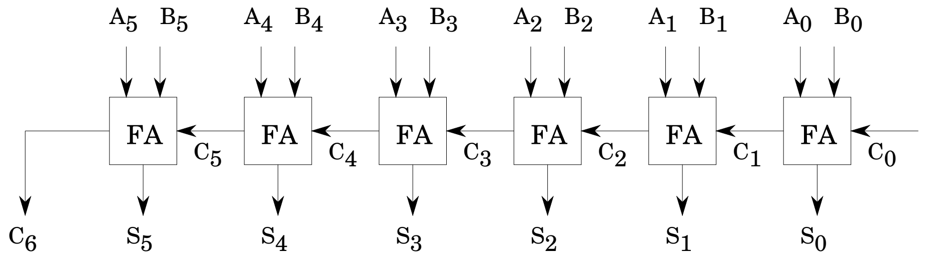
Full-Adder Cell



Half-Adder Cell

Carry Propagate Adder

The carry propagate adder is a linearly connected array of full-adder (FA) cells. The topology of the CPA is illustrated below for $k = 8$.



The total delay of the carry propagate adder is k times the delay of a single full-adder cell. This is because the ith cell needs to receive the correct value of the carry-in bit C_i in order to compute its correct outputs. Tracing back to the 0th cell, we conclude that a total of k full-adder delays is needed to compute the sum vector S and the final carry-out C_k. Furthermore, the total area of the k-bit CPA is equal to k times a single full-adder cell area. The CPA scales up very easily, by adding additional cells starting from the most significant.

The subtraction operation can be performed on a carry propagate adder by using 2's complement arithmetic. Assuming we have a k-bit CPA available, we encode the positive numbers in the range $[0, 2^{k-1} - 1]$ as k-bit binary vectors with the most significant bit being 0. A negative number is then represented with its most significant bit as 1. This is accomplished as follows: Let $x \in [0, 2^{k-1}]$, then $-x$ is represented by computing $2^k - x$. For example, for $k = 3$, the positive numbers are $0, 1, 2, 3$ encoded as $000, 001, 010, 011$, respectively. The negative 1 is computed as $2^3 - 1 = 8 - 1 = 7 = 111$. Similarly, -2, -3, and -4 are encoded as 110, 101, and 100, respectively. This encoding system has two advantages which are relevant in performing modular arithmetic operations:

- The sign detection is easy: the most significant bit gives the sign.
- The subtraction is easy: In order to compute $x - y$, we first represent $-y$ using 2's complement encoding, and then add x to $-y$.

The CPA has several advantages but one clear disadvantage: the computation time is too long for our application, in which the operand size is in the order of several hundreds, up to 2048 bits. Thus, we need to explore other techniques with the hope of building circuits which require less time without significantly increasing the area.

Carry Completion Sensing Adder

The carry completion sensing adder is an asynchronous circuit with area requirement proportional to k. It is based on the observation that the average time required for the carry propagation process to complete is much less than the worst case which is k full-adder delays. For example, the addition of 15213 by 19989 produces the longest carry length as 5, as shown below:

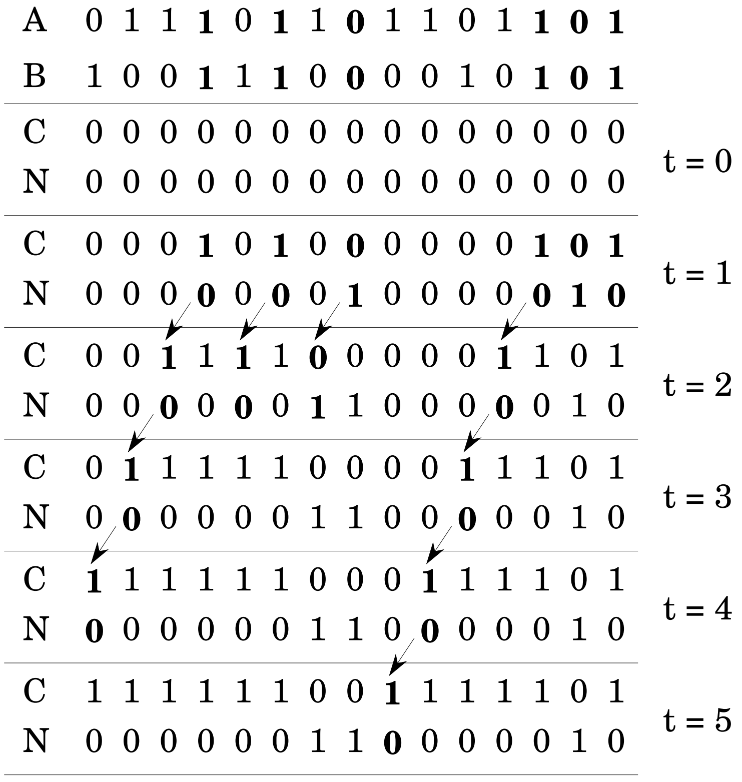
$$\begin{array}{l} \mathrm{A} = 0\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \\ \mathrm{B} = 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1 \\ \hline \xleftarrow[4]{\qquad\qquad} \ \xleftarrow[1]{\quad} \ \xleftarrow[5]{\qquad\qquad\quad} \ \xleftarrow[1]{\quad} \end{array}$$

A statistical analysis shows that the average longest carry sequence is approximately 4.6 for a 40-bit adder [?]. In general, the average longest carry produced by the addition of two k-bit integers is upper bounded by $\log_2 k$. Thus, we can design a circuit which detects the completion of all carry propagation processes, and completes in $\log_2 k$ time in the average.

In order to accomplish this task, we introduce a new variable N in addition to the carry variable C. The value of C and N for ith position is computed using the values of A and B for the ith position, and the previous C and N values, as follows:

$$\begin{aligned}
(A_i, B_i) = (0,0) &\implies (C_i, N_i) = (0,1)\\
(A_i, B_i) = (1,1) &\implies (C_i, N_i) = (1,0)\\
(A_i, B_i) = (0,1) &\implies (C_i, N_i) = (C_{i-1}, N_{i-1})\\
(A_i, B_i) = (1,0) &\implies (C_i, N_i) = (C_{i-1}, N_{i-1})
\end{aligned}$$

Initially, the C and N vectors are set to zero. The cells which produce C and N values start working as soon as the values of A and B are applied to them in parallel. The output of a cell (C_i, N_i) settles when its inputs (C_{i-1}, N_{i-1}) are settled. When all carry propagation processes are complete, we have either $(C_i, N_i) = (0,1)$ or $(C_i, N_i) = (1,0)$ for all $i = 1, 2, \ldots, k$. Thus, the end of carry completion is detected when all $X_i = C_i + N_i = 1$ for all $i = 1, 2, \ldots, k$, which can be accomplished by using a k-input AND gate.



A	0	1	1	**1**	0	**1**	1	**0**	1	1	0	1	**1**	**0**	**1**	
B	1	0	0	**1**	1	**1**	0	**0**	0	0	1	0	**1**	**0**	**1**	
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t = 0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
C	0	0	0	**1**	0	**1**	0	**0**	0	0	0	0	**1**	**0**	**1**	t = 1
N	0	0	0	**0**	0	**0**	0	**1**	0	0	0	0	**0**	**1**	**0**	
C	0	0	**1**	1	**1**	1	**0**	0	0	0	0	**1**	1	0	1	t = 2
N	0	0	**0**	0	**0**	0	**1**	1	0	0	0	**0**	0	1	0	
C	0	**1**	1	1	1	1	0	0	0	0	**1**	1	1	0	1	t = 3
N	0	**0**	0	0	0	0	1	1	0	0	**0**	0	0	1	0	
C	**1**	1	1	1	1	1	0	0	0	**1**	1	1	1	0	1	t = 4
N	**0**	0	0	0	0	0	1	1	0	**0**	0	0	0	1	0	
C	1	1	1	1	1	1	0	0	**1**	1	1	1	1	0	1	t = 5
N	0	0	0	0	0	0	1	1	**0**	0	0	0	0	1	0	