ECE380 Digital Logic

Number Representation and Arithmetic Circuits:

Number Representation and Unsigned Addition

Positional representation

- First consider integers
  - Begin with positive only descriptions and expand to include negative numbers
  - Numbers that are positive only are ***unsigned*** and numbers that can also assume negative values are ***signed***
- For the decimal system:
  - A number consists of digits having ten possible values (0-9)
  - Each digit represents a multiple of a power of 10

    $(123)_{10}=1x10^2+2x10^1+3x10^0$
- In general, an integer is represented by *n* decimal digits

  $D=d_{n-1}d_{n-2}. . .d_1d_0$
- Representing the value

  $V(D)=d_{n-1}x10^{n-1} + d_{n-2}x10^{n-2} +. . .+ d_1x10^1 + d_0x10^0$

Positional representation

- Because the digits have 10 possible values and each digit is weighted as a power of 10, we say that decimal numbers are *base*-10 or *radix*-10 numbers
- In digital systems we commonly use the ***binary***, or *base*-2, number system in which digits can be 0 or 1
  - Each digit is called a ***bit***
- The positional representation is

  $B=b_{n-1}b_{n-2}...b_1b_0$
- Representing a integer with the value

  $V(B)=b_{n-1}x2^{n-1} + b_{n-2}x2^{n-2} +...+ b_1x2^1 + b_0x2^0$

Positional representation

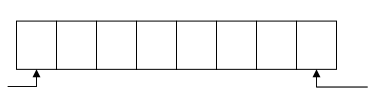
- The binary number 1101 represents the value

  $V=1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$

  $V=8+4+1=13$
- So

  $(1101)_2=(13)_{10}$
- The range of numbers that can be represented by a binary number depends of the number of bits used
- In general, using *n* bits allows a representation of positive integers in the range 0 to 2^n-1

Most significant Bit (MSB)



Least significant Bit (LSB)

Decimal/Binary conversion

- A binary number can be converted to a decimal number directly by evaluating the expression
  $V(B)=b_{n-1}x2^{n-1} + b_{n-2}x2^{n-2} +. . .+ b_1x2^1 + b_0x2^0$
- using decimal arithmetic (by expansion)
- Converting from a decimal to a binary number can be preformed by successively dividing the decimal number by 2 as follows
  - Divide the decimal number (D) by 2 producing a quotient D/2 and a remainder. The remainder will be 0 or 1 (since we divide by 2) and will represent a single bit (the LSB) of the binary equivalent
  - Repeatedly divide the generated quotient by 2 until the quotient=0. For each divide, the remainder represents one of the binary digits (bits) of the binary equivalent

Decimal/Binary conversion

Convert $(857)_{10}$

			Remainder	
$857 \div 2$	$=$	428	1	LSB
$428 \div 2$	$=$	214	0	
$214 \div 2$	$=$	107	0	
$107 \div 2$	$=$	53	1	
$53 \div 2$	$=$	26	1	
$26 \div 2$	$=$	13	0	
$13 \div 2$	$=$	6	1	
$6 \div 2$	$=$	3	0	
$3 \div 2$	$=$	1	1	
$1 \div 2$	$=$	0	1	MSB

Result is $(1101011001)_2$

Octal and hexadecimal numbers

- Positional notation can be used for any radix (base). If the radix is *r*, then the number
  $K = k_{n-1}k_{n-2}...k_1k_0$
  has the value

$$V(K) = \sum_{i=0}^{n-1} k_i \times r^i$$

- Numbers with radix-8 are called ***octal*** and numbers with radix-16 are called ***hexadecimal*** (or hex)
  – For octal, digit values range from 0 to 7
  – For hex, digital values range from 0-9 and A-F

Numbers in different systems

Decimal	Binary	Octal	Hex
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Decimal	Binary	Octal	Hex
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary to hex or octal conversion

- **Group binary digits into groups of four and assign each group a hexadecimal digit.**

0110	**1011**	**0111**
6	**B**	**7**

  - **Binary-to-octal:**

011	**010**	**110**	**111**
3	**2**	**6**	**7**

  - **Hexadecimal-to-binary:**

A	**1**	**9**
1010	**0001**	**1001**

  - **Octal-to-binary:**

5	**0**	**3**	**1**
101	**000**	**011**	**001**

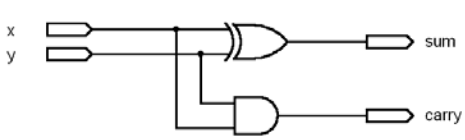
Unsigned number addition

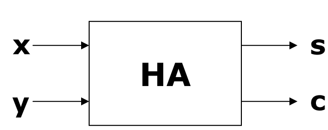
- Additional of two 1-bit numbers gives four possible combinations

$$\begin{array}{r} x \\ +\ y \\ \hline c\ \ s \end{array} \qquad \begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ \ 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ \ 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ \ 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 1\ \ 0 \end{array}$$

carry sum

x	*y*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





Unsigned number addition

- Larger numbers have more bits involved
  - There is still the need to add each pair of bits
  - But, for each bit position *i*, the addition operation may include a ***carry-in*** from bit position *i-1*

$X=x_4x_3x_2x_1x_0$	0 1 1 1 1	$(15)_{10}$
$Y=y_4y_3y_2y_1y_0$	0 1 0 1 0	$(10)_{10}$
	1 1 1 0	Generated carries
$S=s_4s_3s_2s_1s_0$	1 1 0 0 1	$(25)_{10}$

Full adder circuit

c_i	x_i	y_i	c_{i+1}	s_{i+1}
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

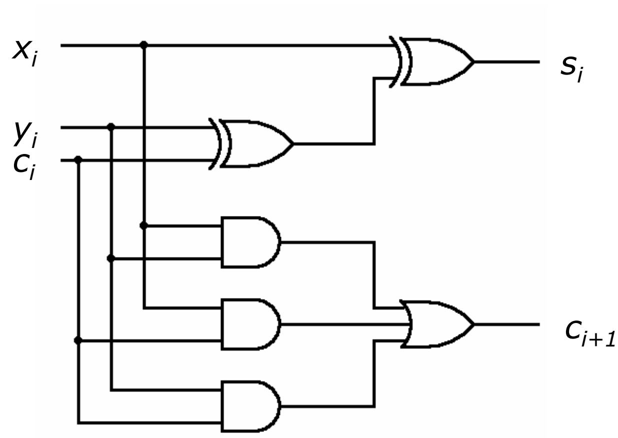
c_i \ x_iy_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$s_i=x_i\oplus y_i\oplus c_i$

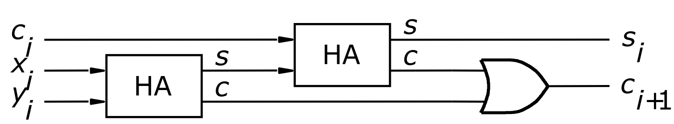
c_i \ x_iy_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$c_i=x_iy_i+y_ic_i+x_ic_i$

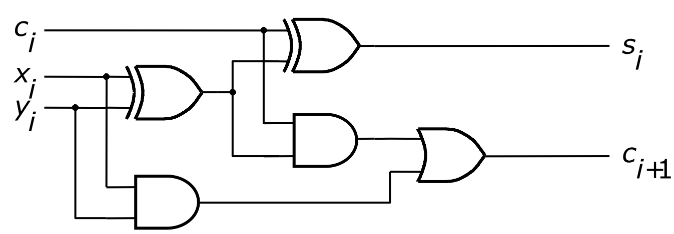
Full adder circuit



Full adder circuit (decomposed)



Block diagram

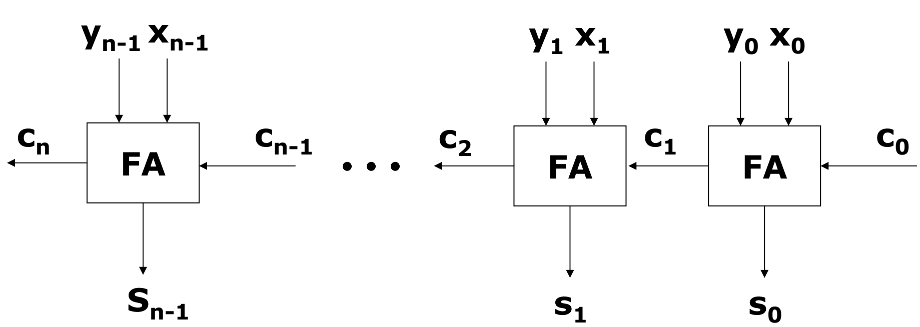


Detailed diagram

Ripple-carry adder

- In performing addition, we start from the least significant digit and add pairs of digits progressing to the most significant digit
- If a carry is produced in position *i*, it is added to operands (digits) in position *i*+1
- A chain of full adders, connected in sequence, can perform this operation
- Such a configuration is called a ***ripple-carry adder*** because of the way the carry signal 'ripple' through from stage to stage

Ripple-carry adder



Ripple-carry adder

- Each full adder introduces a certain delay before its s_i and c_{i+1} outputs are valid
  - The propagation delay through the full adder
- Let this delay be Δt
- The carry out of the first stage c_1 arrives at the second stage Δt after the application of the x_0 and y_0 inputs
- The carry out of the second stage c_2 arrives at the third stage with a delay of $2\Delta t$, and so on
- The signal c_{n-1} is valid after $(n-1)\Delta t$, and the complete sum is available after a delay of $(n)\Delta t$
- The delay obviously depends on the size of the numbers (*i.e.* the number of bits)