# TSTE18 Digital Arithmetic
# Seminar 2

Oscar Gustafsson

## Addition and subtraction

- **Ripple-carry addition**
- **Solutions to slow ripple-carry addition**
  - **Redundant number systems**
  - Carry-acceleration
- Multi-operand addition

## Binary addition

- Add the two binary numbers

$$X = \sum_{i=-L}^{M-1} x_i 2^i \text{ and } Y = \sum_{i=-L}^{M-1} y_i 2^i \tag{1}$$

- The result can be written as

$$S = c_M + \sum_{i=-L}^{M-1} s_i 2^i \tag{2}$$

where

$$s_i = x_i \oplus y_i \oplus c_i$$

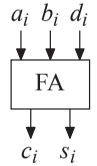
and

$$c_{i+1} = \text{majority}(x_i, y_i, c_i) = x_i y_i + y_i c_i + x_i c_i = x_i y_i + (x_i \oplus y_i) c_i$$

with $c_{-L} = c_{\text{in}} = 0$

## Binary addition

- These equations are often realized with a full adder cell employing three inputs, $x_i$, $y_i$, and $c_i$, and two outputs, $s_i$ and $c_{i+1}$



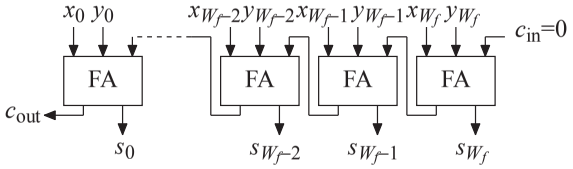

- Note that the signals of a full adder cell computes the expression

$$2c_{i+1} + s_i = x_i + y_i + c_i \tag{3}$$

## Binary addition

- Gate level realization

- Several full adders are connected to form a ripple carry adder




## Redundant adders

- Redundant example:

| Value | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|---|
| 75 | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | $x_i$ |
| 94 | + | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | $y_i$ |
| 169 | | 0 | 2 | 0 | 1 | 2 | 1 | 2 | 1 | $s_i$ |

- Constant gate delay!

## Addition

- Ripple carry adders are the base line adders
- Gate delay linear in the word length
- Example

| Value | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|---|
| 75 | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | $x_i$ |
| 94 | + | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | $y_i$ |
| | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | | $c_i$ |
| 169 | | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | $s_i$ |

- Becomes problematic for high-speed implementations
- Can be improved in two different ways
  - Compute the carries in parallel (carry-acceleration)
  - Do not compute the carries (redundant number systems)

## Carry-save representation

- A practical redundant number system is the carry-save representation
- Consider a ripple carry adder where the carries are not connected to the next stage
- Three words can be added and the result is represented using two outputs (carry and sum)
- Example

| Value | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|---|
| 75 | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | $x_i$ |
| −98 | | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | $y_i$ |
| 7 | + | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | $z_i$ |
| 30 | ~~0~~ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | | $c_i$ |
| −46 | + | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | $s_i$ |
| −16 | | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 0 | $s_i$ |

## Carry-save representation

- Carry-save representation can be seen as a representation using a digit-set $\{0, 1, 2\}$
- Two's complement representation can be straightforwardly used (the MSD has negative sign)

## Carry-save representation

- Consider a three digit CS representation:

| Value | Representation | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 6 | 022 | | | | | | | | |
| 5 | 021 | | | | | | | | |
| 4 | 020 | 012 | | | | | | | |
| 3 | | 011 | | | | | | | |
| 2 | | 010 | 002 | 122 | | | | | |
| 1 | | | 001 | 121 | | | | | |
| 0 | | | 000 | 120 | 112 | | | | |
| −1 | | | | | 111 | | | | |
| −2 | | | | | 110 | 102 | 222 | | |
| −3 | | | | | | 101 | 221 | | |
| −4 | | | | | | 100 | 220 | 212 | |
| −5 | | | | | | | | 211 | |
| −6 | | | | | | | | 210 | 202 |
| −7 | | | | | | | | | 201 |
| −8 | | | | | | | | | 200 |

## Carry-save representation

- Assume that the result should be converted to a number in the corresponding three-bit two's complement range $[-4, 3]$

| Value | Represenation | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 011 | | | | | | |
| 2 | 010 | 002 | 122 | | | | |
| 1 | | 001 | 121 | | | | |
| 0 | | 000 | 120 | 112 | | | |
| −1 | | | | 111 | | | |
| −2 | | | | 110 | 102 | 222 | |
| −3 | | | | | 101 | 221 | |
| −4 | | | | | 100 | 220 | 212 |

## Carry-save representation

- The result of a carry-save addition will never give two in the least significant digit (despite being a valid CS representation)

| Value | Representation | | | | | |
|---|---|---|---|---|---|---|
| 3 | 011 | | | | | |
| 2 | 010 | | | | | |
| 1 | | 001 | 121 | | | |
| 0 | | 000 | 120 | | | |
| −1 | | | | 111 | | |
| −2 | | | | 110 | | |
| −3 | | | | | 101 | 221 |
| −4 | | | | | 100 | 220 |

- If both MSBs are either one or zero, the sign of the result is well defined
- If exactly one of the MSBs is one, it is not possible to determine the sign directly
- Note that $-1$ has a unique representation (although there are many bit-patterns corresponding to $-1$)
- Hence, a quick way to detect zero is to add $-1$ and check for all ones among the digits

## Carry-save representation

- However, the carry-save representation is possibly shift unsafe
- Consider the following example

| Value | | Bits | | |
|---|---|---|---|---|
| 2 | | 0 | 1 | 0 |
| −2 | | 1 | 1 | 0 |
| 0 | + | 0 | 0 | 0 |
| −4 | ~~0~~ | 1 | 0 | |
| −4 | | 1 | 0 | 0 |
| -8 | ~~1~~ | 0 | 0 | 0 |

- Why does it work?
  - All three-bit two's complement arithmetic is performed modulo 8 (the effect of ignoring the carry output)
  - Hence, $-8 \bmod 8 = 0$

## Carry-save representation

- Consider shifting the carry-save result one position to the right

| Value | | Bits | | | |
|---|---|---|---|---|---|
| −2 | | 1 | 1 | 0 | |
| −2 | + | 1 | 1 | 0 | 0 |
| −4 | ~~1~~ | 1 | 0 | 0 | 0 |

- Note that the other representations of 0 works OK
  - $000 \Rightarrow 000.0$
  - $120 \Rightarrow 112.0 = -4 + 2 + 2 \times 1$
  - $112 \Rightarrow 111.2 = -4 + 2 + 1 + 2 \times \frac{1}{2}$
- How can this be solved?

## Carry-save representation

- Focus on the two most significant positions

$$\begin{array}{cc} c_{M+1} & c_M \\ s_{M+1} & s_M \end{array} \Rightarrow \begin{array}{cc} c_{M+1} & c_M \\ & s_M \end{array} \quad \begin{array}{c} \\ s_M \end{array}$$

- In the example we neglected the carry output twice
- Consider extending the wordlength

| Value | | Bits | | | |
|---|---|---|---|---|---|
| 2 | | | 0 | 1 | 0 |
| −2 | | | 1 | 1 | 0 |
| 0 | | + | 0 | 0 | 0 |
| 4 | | 0 | 1 | 0 | |
| −4 | + | 1 | 1 | 0 | 0 |
| 0 | ~~1~~ | 0 | 0 | 0 | 0 |

## Carry-save representation

- Shifting this representation is OK!

| Value | | Bits | | | | |
|---|---|---|---|---|---|---|
| 2 | | 0 | 0 | 1 | 0 | |
| −2 | + | 1 | 1 | 1 | 0 | 0 |
| 0 | ~~1~~ | ~~0~~ | 0 | 0 | 0 | 0 |

- So extending the word length helps – But not really the solution!
- The sum sign-bit is always the same after sign extension/right shift
- Problem occurs when the neglected carry bit is different from the carry sign bit, as the neglected carry bit is the correct bit to shift in

## Carry-save representation

- Consider the troublesome cases (remember that the result should be in the range $[-4, 3]$)

| $c_{M+1}$ | $s_M$ | $c_M$ | Value | Correct | Comment |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | Overflow |
| 0 | 1 | 1 | 2 | 0 | |
| 1 | 0 | 0 | 0 | 2 | |
| 1 | 1 | 0 | 1 | 1 | Overflow |

- Remaining cases

| $c_{M+1}$ | $s_M$ | $c_M$ | Value | Correct |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 | 2 |

## Carry-save representation

- Truth table

| $c_{M+1}$ | $s_M$ | $c_M$ | Value |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 2 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 2 |

- Can replace the sign-bits as

$$c'_M = c_{M+1} \oplus s_M \oplus c_M$$

$$s'_M = c_{M+1}$$

## Carry-save representation

- Redo the example

| Value | | Bits | | |
|---|---|---|---|---|
| 2 | | 0 | 1 | 0 |
| $-2$ | | 1 | 1 | 0 |
| 0 | $+$ | 0 | 0 | 0 |
| 0 | ~~0~~ | 0 | 0 | |
| 0 | $+$ | 0 | 0 | 0 |
| 0 | ~~0~~ | 0 | 0 | 0 |

## Redundant adders

- What about adding two redundant numbers?
- Let us consider a higher radix and come back to radix-2 later
- Adding two (non-redundant) radix-4 numbers

| $i$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 3 | 0 | 1 | 2 | 3 | 1 | 2 | $x_i$ |
| $+$ | 1 | 1 | 3 | 2 | 0 | 1 | 3 | 3 | $y_i$ |
| | 1 | 4 | 3 | 3 | 2 | 4 | 4 | 5 | $s_i$ |

- In general it will be enough to use digits up to $2(r-1)$ to represent the sum of two non-redundant numbers

## Redundant adders

- Add two redundant numbers

| $i$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 3 | 4 | 1 | 2 | 4 | 1 | 5 | $x_i$ |
| + | 1 | 2 | 3 | 2 | 0 | 0 | 5 | 3 | $y_i$ |
| | 1 | 5 | 7 | 3 | 2 | 4 | 6 | 8 | $s_i$ |

- The required range increases for each term
- Would like to move the results to within a suitable range ($2(r-1)$, in this case 6)
- Worst case result is 12, which is $12 = 2 \cdot 4 + 4$, leading to that we propagate a "carry", called transfer digit, of two to the next stage and keep an interim digit 4

## Redundant adders

- The largest incoming transfer digit is, thus, 2
- Hence, if we can make sure that the interim digit is at most 4, there will never be any further carries generated
- Example

| $i$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 3 | 4 | 1 | 2 | 4 | 1 | 5 | $x_i$ |
| + | 1 | 2 | 3 | 2 | 0 | 0 | 5 | 3 | $y_i$ |
| | 1 | 5 | 7 | 3 | 2 | 4 | 6 | 8 | Partial result |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | - | transfer digit |
| | 1 | 1 | 3 | 3 | 2 | 4 | 2 | 4 | interim digit |
| 0 | 2 | 2 | 3 | 3 | 2 | 5 | 3 | 4 | result |

## Signed-digit representations

- Last time we saw that a radix-$r$ representation should have digits in the range $0 \leq x_i \leq r-1$ to be non-redundant
- This can be generalized to $-\alpha \leq x_i \leq \beta$ leading to a signed-digit representation, i.e., the digit may have a sign
- A signed-digit representation can be either redundant or non-redundant
  - $\alpha + \beta = r - 1$: non-redundant
  - $\alpha + \beta \geq r$: redundant (compare with $x_i \geq r$)
- Define the *redundancy index* as $\rho = \alpha + \beta - r + 1$
- If $\rho \geq 1$, the representation is redundant
- For radix-2, the most common SD representation is the binary sined digit with $x_i \in \{-1, 0, 1\}$
- Here, $\alpha = 1$, $\beta = 1$, and $\rho = 1$

## Generalized carry-free addition

- The earlier described constant time addition can be described as follows
  1. Compute digit-wise sums for each position, $p_i = x_i + y_i$
  2. Divide the sum into a transfer digit, $t_{i+1}$, and an interim digit, $w_i = p_i - t_{i+1} r$
  3. Add the interim digit and the incoming transfer digit to obtain the sum $s_i = w_i + t_i$

## Generalized carry-free addition

- Assume the transfer digit is in the range $t_{\min} \leq t_i \leq t_{\max}$ and the interim digit is in the range $w_{\min} \leq w_i \leq w_{\max}$
- Clearly, we should have $t_{\max} + w_{\max} = \beta$ and $t_{\min} + w_{\min} = -\alpha$ (if not $\alpha$ and $\beta$ can be changed)
- The interim sum is bounded as $-2\alpha \leq p_i \leq 2\beta \Rightarrow -2\alpha \leq w_i + rt_{i+1} \leq 2\beta$
- For the largest digit-wise sum it must hold that

$$w_{\min} + rt_{\min} \leq -2\alpha$$

$$w_{\max} + rt_{\max} \geq 2\beta$$

- Combining these equations with those above leads to

$$t_{\min} \leq -\frac{\alpha}{r-1}$$

$$t_{\max} \geq \frac{\beta}{r-1}$$

- In addition, $w_{\max} - w_{\min} \geq r - 1$ to result in all intermediate values possible for the encoding

## Generalized carry-free addition

- For the previous scheme to operate properly one of the following two cases are required:
  - $r \geq 3$, $\rho \geq 3$
  - $r \geq 3$, $\rho = 2$, $\alpha \neq 1$, $\beta \neq 1$
- This means that binary signed-digit can not utilize the generalized carry-free addition
- However, a limited carry-propagation addition algorithm is available

## Binary signed-digit addition

- The main obstacle to avoid is that the incoming transfer digit and the interim digit are both 1 or both $-1$ as the result will not be within the correct range
- One way to solve this is to do the computation in a third step as
  1. Compute digit-wise sums for each position, $p_i = x_i + y_i$
  2. Divide the sum into a transfer digit, $t_{i+1}$, and an interim digit, $w_i$, based on the preliminary incoming transfer digit, $e_{i-1}$, where $e_i = 0$ if $x_i \geq 0$ and $y_i \geq 0$

| $p_i$ | $e_{i-1}$ | $t_{i+1}$ | $w_i$ |
|---|---|---|---|
| 2 | - | 1 | 0 |
| 1 | 0 | 1 | $-1$ |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| $-1$ | 0 | 0 | $-1$ |
| $-1$ | 1 | $-1$ | 1 |
| $-2$ | 0 | $-1$ | 0 |

  3. Add the interim digit and the incoming transfer digit to obtain the sum $s_i = w_i + t_i$

## Binary signed-digit addition

- Example

| $i$ | 4 | 3 | 2 | 1 | 0 | Signal |
|---|---|---|---|---|---|---|
| | 1 | 0 | $-1$ | 0 | $-1$ | $x_i$ |
| + | 0 | $-1$ | 0 | 1 | $-1$ | $y_i$ |
| | 1 | $-1$ | $-1$ | 1 | $-2$ | $p_i$ |
| | 0 | 1 | 1 | 0 | 1 | $e_i$ |
| 0 | $-1$ | 0 | 0 | $-1$ | - | transfer digit |
| | 1 | 1 | $-1$ | 1 | 0 | interim digit |
| 0 | 0 | 1 | $-1$ | 0 | 0 | result |