Progettazione e sintesi di circuiti digitali

Lecture 12
Fixed-point arithmetics
Binary representation of real numbers

- Problem: how to represent real numbers \( x \) in range \([a, b]\) using an \( n \)-bit binary string \( b \)
  - more real numbers then different \( n \)-bit strings
  - many real numbers map to the same string
  - all, except one at most, will be represented with an error \( e \)
Binary representation of real numbers

Definitions:

- $R(x)$: binary number representing real number $x$
- $V(b)$: value of binary number $b$

Absolute and relative error:

$$e_a = |V(R(x)) - x|, \quad e_r = \left| \frac{V(R(x)) - x}{x} \right|$$

Absolute and relative accuracy:

$$a_a = \max_{x \in [a,b]} |V(R(x)) - x|, \quad a_r = \max_{x \in [a,b]} \left| \frac{V(R(x)) - x}{x} \right|$$
Binary representation of real numbers

Example: represent any $X \in [0, 1000]$ as 10-bit binary integer, converting $X$ to nearest integer

• Compute absolute error and accuracy when $X=512.742$
  
  512.742 $\rightarrow$ nearest integer $513 = 10\ 0000\ 0001_2$
  
  $e_a(512.742) = |512.742 - 513| = 0.258$
  
  $a_a = \max e_a(X) = 0.5$

• Error and accuracy depend on function $R$
  
  — if we choose to truncate $X$ to nearest integer less than $X$, then: $512.742 \rightarrow 512$
  
  $e_a(512.742) = |512.742 - 512| = 0.742$
  
  $a_a = \max e_a(X) = 1$
Binary representation of real numbers

- Resolution: \( r = \min_{b_1, b_2 \in B, b_1 \neq b_2} |V(b_1) - V(b_2)| \)
  - equal to the value of the least significant bit (LSB) of the binary code

- Example: find the resolution needed to represent \( X \) from 54 500 000 km to 4 500 000 000 km with 3% relative accuracy
  
  max relative error halfway between \( X_{\min} \) and \( X_{\min} + r \)

  \[
  a_r = \left| \frac{54500000 - (54500000 + 0.5 \cdot r)}{54500000 + 0.5 \cdot r} \right| = 3\% \implies r = 3370000
  \]

  number of bits required for an unsigned representation:

  \[
  n = \left\lceil \log_2 \left( \frac{X_{\max}}{r} \right) \right\rceil = \left\lceil \log_2 \left( \frac{4500}{3.37} \right) \right\rceil = \left\lceil 10.38 \right\rceil = 11
  \]
Fixed-point representation

• A **p.f** fixed-point number is an n-bit binary string (with \( n = p + f \)) coding a rational number with \( p \) bits for the integer part and \( f \) bits for the fractional part.

<table>
<thead>
<tr>
<th>weight</th>
<th>( 2^{p-1} )</th>
<th>( 2^{p-2} )</th>
<th>( ... )</th>
<th>( 2^0 )</th>
<th>( 2^{-1} )</th>
<th>( ... )</th>
<th>( 2^{-f} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>( b_{n-1} )</td>
<td>( b_{n-2} )</td>
<td>( ... )</td>
<td>( b_{n-p} )</td>
<td>( b_{f-1} )</td>
<td>( ... )</td>
<td>( b_0 )</td>
</tr>
</tbody>
</table>

• Signed numbers require one more bit
  – format **sp.f** \( \rightarrow \) \( n = p + f + 1 \)

• Resolution \( r \) set by the number \( f \) of fractional bits
  – \( r = 2^{-f} \)

• Range (for a signed number system):
  – from \(-2^p\) to \(+2^p - r\)
Fixed-point representation

• Examples of fixed-point numbers:

<table>
<thead>
<tr>
<th>Format</th>
<th>Code</th>
<th>r</th>
<th>Integer</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>1.011</td>
<td>0.125</td>
<td>11</td>
<td>1.375</td>
<td>11/8</td>
</tr>
<tr>
<td>s1.3</td>
<td>01.011</td>
<td>0.125</td>
<td>11</td>
<td>1.375</td>
<td>11/8</td>
</tr>
<tr>
<td>s1.3</td>
<td>11.011</td>
<td>0.125</td>
<td>-5</td>
<td>-0.625</td>
<td>-5/8</td>
</tr>
<tr>
<td>2.4</td>
<td>10.0111</td>
<td>0.0625</td>
<td>39</td>
<td>2.4375</td>
<td>3916</td>
</tr>
</tbody>
</table>

• Conversion of an (s)p.f code to decimal:
  – kill the binary point, then convert to integer, then multiply by the resolution r

• Conversion from decimal to (s)p.f:
  – multiply by $2^f (=1/r)$, then round to nearest integer, then convert to binary
Fixed-point representation

• Conversion examples:
  convert x=1.389 to 1.3 format:
  → multiply by $2^f (=1/r)$: $x \cdot 2^3 = 11.112$
  → round to nearest integer: $11.112 \rightarrow 11$
  → convert to n-bit binary: $11 \rightarrow 1001 \rightarrow 1.001$
  convert 1.001 (assuming 1.3 format) to decimal:
  → kill the binary point: $1.001 \rightarrow 1001$
  → convert to integer: $1001 \rightarrow 11$
  → multiply by the resolution $r$: $11 \cdot 2^{-3} = 1.375$
  → conversion error $e_a = |1.389 - 1.375| = 0.014$
Fixed-point representation

• Fixed-point representation is often used in applications where range and accuracy are well known (e.g., signal processing)
  – binary point placed to exploit full range
  – overflow eliminated or kept under control

• **Scaling** applied to maximize range exploitation
  – typically, X values scaled to fall between -1 and +1
  – then, **s0.f** representation can be used (with s0.15 being the most popular)
Fixed-point representation

Fixed-point system design example:

• Represent a voltage between 0 and 10V with 10mV absolute accuracy
  → p=4 bits required to represent 10V
  → for \( a_a \leq 10\text{mV} \), then \( r \leq 20\text{mV} \) (rounding to nearest integer)
  → then, \( r=2^{-6}=0.015625 \leq 0.02 \) and \( a_a=r/2=0.0078125 \)
  → a 4.6 FP format satisfies the requirements

• We can improve range exploitation if we design the FP system to have a resolution \( r=20\text{mV} \)
  → if \( r=20\text{mV} \), then \( n=\lceil \log_2(X_{\text{max}}/r) \rceil = \lceil 8.97 \rceil = 9 \) bits are required to represent the full 0 to 10V range
  → the representable range is 0 to \( (2^n-1) \cdot r =511 \cdot 20\text{mV}=10.22\text{ V} \)
  → a scaling operation is implied in setting the resolution to a value different from \( 2^{-f} \), with scaling factor \( S=2^{-f}/r \)
Fixed-point representation

Exercise:

• Design a fixed-point system to represent temperatures ranging from -20 °C to 80 °C with at least 0.1 °C absolute accuracy
  – assume conversion with rounding to nearest integer
  – try first without scaling
  – then, scale the temperature to minimize the number of bits required by the system
Operations on FP numbers

- Operations on FP numbers are based on normal integer arithmetics
  - FP numbers inside a circuit are represented as signed or unsigned binary integer
  - Information on (s)p.f format adopted is used to convert to and from decimal representation outside of the circuit, and to properly design the arithmetic datapath
- Different strategies to choose the length $n$ of the FP representation
  - Fixed $n$: most common solution; datapath design easier; must design to prevent overflow and accuracy loss beyond specs
  - Variable $n$: prevents (or limits) overflow and loss of accuracy by selectively increasing the size of the datapath; circuit design more complicated
Operations on FP numbers

• Operations on FP numbers are based on normal integer arithmetics
  – FP numbers inside a circuit are represented as signed or unsigned binary integer
  – information on (s)p.f format adopted is used to convert to and from decimal representation outside of the circuit, and to properly design the arithmetic datapath

• Problems with operations on FP numbers
  – addition/subtraction: possible overflow
  – multiplication: possible overflow and loss of significant digits
Operations on FP numbers

• Addition/subtraction:
  – adding two FP numbers with the same (s)p.f format requires an (s)(p+1).f format for the result to prevent overflow for any possible value of inputs
  – when formats are different, binary point must be aligned to obtain correct result:

    $$(s)p_1.f_1 + (s)p_2.f_2 \rightarrow (s)\text{max}(p_1,p_2).\text{max}(f_1,f_2)$$

Example:  
A (s3.4) + B (s5.1) = S (s5.4)
Operations on FP numbers

• Multiplication:
  – No overflow and no loss of significant digits if unsigned
    \[ p_1.f_1 \times p_2.f_2 = (p_1+p_2).(f_1+f_2) \]
  – Signed
    \[ sp_1.f_1 \times sp_2.f_2 = s(p_1+p_2+1).(f_1+f_2) \]
  – Typical method to keep the datapath size constant is to discard a suitable number of the rightmost (i.e., less significant) bits \( \rightarrow \) no overflow but loss of significant digits
  – If the input factors format is \( s0.f \), then the product format is \( s1.(2f) \) \( \rightarrow \) discarding the less significant \( f \) bits and the MSB brings the format back to \( s0.f \)
    • possible overflow only in one case: \((-1) \times (-1) = +1\)
Operations on FP numbers

Practical example: MAC unit of a DSP

• many DSP scale numbers to a s0.15 format
• multiplying two s0.15 numbers gives a s1.30 number
• the MAC unit is designed to accumulate up to N products without overflow and loss of digits
• the result is then rounded to get back to s0.15 format
• what should be the size of the accumulator if N=256?
Operations on FP numbers

Practical example: MAC unit of a DSP (continued)

- Max value of accumulated sum is $N \times \text{max}(P)$
- $P$ is in s1.30 format, $S$ requires an additional $\lceil \log_2(N) \rceil = 8$ bits, the required format is s9.30
Operations on FP numbers

VHDL model of the MAC unit

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MAC is
  port(A,B: in signed(15 downto 0);
       CK,R: in std_logic;
       Y: out signed(15 downto 0));
end MAC;

architecture RTL of MAC is
  signal P: signed(31 downto 0);
  signal S: signed(39 downto 0);
begin  -- continues on next page
Operations on FP numbers

-- continued from previous page

P <= A * B;
ACC: process
begin
    wait until CK'event and CK = '1';
    if R = '0' then
        S <= (others => '0');
    else
        -- the decimal point of S and P are aligned w.r.t. the LSB
        -- so they can be added directly
        S <= S + P;
    end if;
end process;
Y <= resize(S(S'left-1 downto 15), Y'length);  
end RTL;

discard the f=15 less significant bits of S (i.e., S(14 downto 0)) and the MSB (S'left)

resize the remaining 24 bits to 16, discarding the 8 most significant bits
(but keeping the sign bit)
Operations on FP numbers

Truncating vs rounding to nearest integer

• Truncating: discarding the rightmost m bits
  – for an sp.f format, the absolute accuracy loss when truncating m (≥1) bits is \(2^{m-f-1}\)
  – hardware implementation comes for free

• Rounding to nearest integer
  – round up if the most significant bit discarded is 1
  – round down if the msb discarded is 0
  – rounding up requires to increment the result, so its hardware implementation includes an adder

– example: round .01111000 to 0.4 format \(\rightarrow .01111000\) round up by adding .00001 \(\rightarrow .1000\)