EE-334
Digital System Design

Custom Digital Circuits

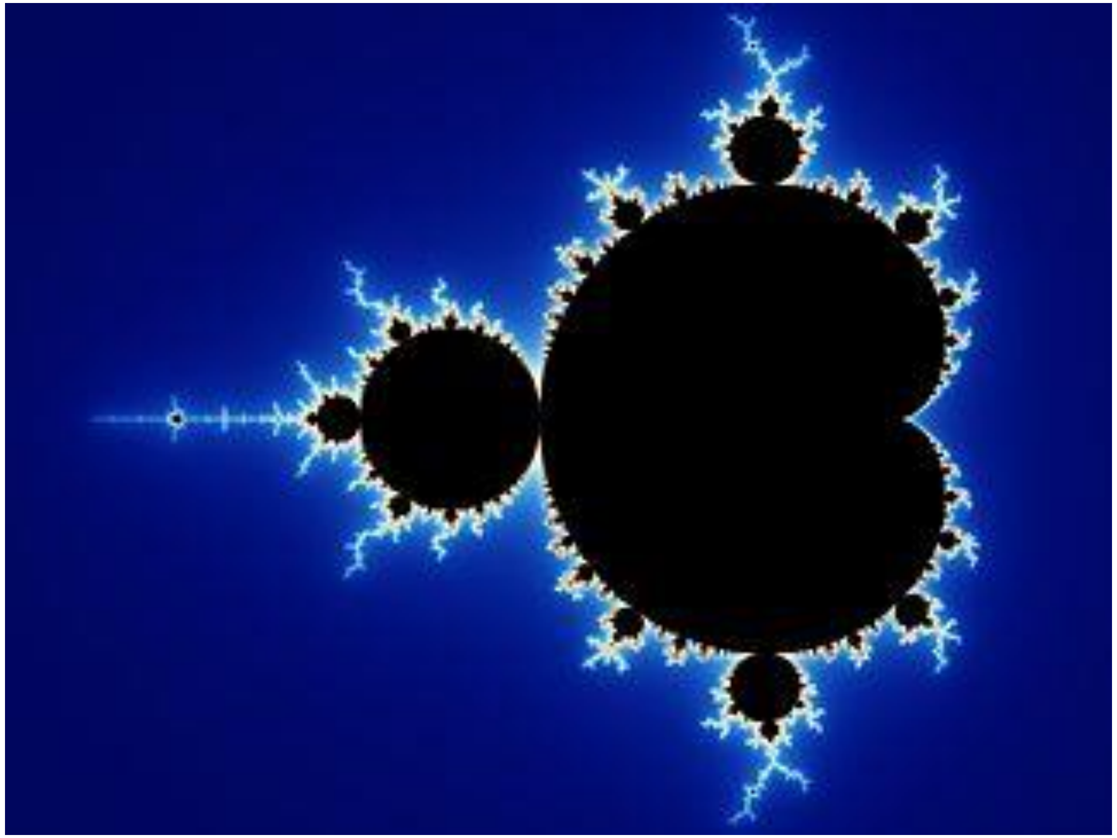
Lab 8: Datapath Design – Mandelbrot

Andreas Burg

The Mandelbrot Set

- The Mandelbrot set is defined as the complex numbers for which the following iteration does not diverge

$$z_{n+1} = z_n^2 + c \qquad\qquad z_0 = 0$$



- Divergence can not be predicted from c
- However, if $|z_{n+1}|^2 > 4$ we know that the iteration will diverge

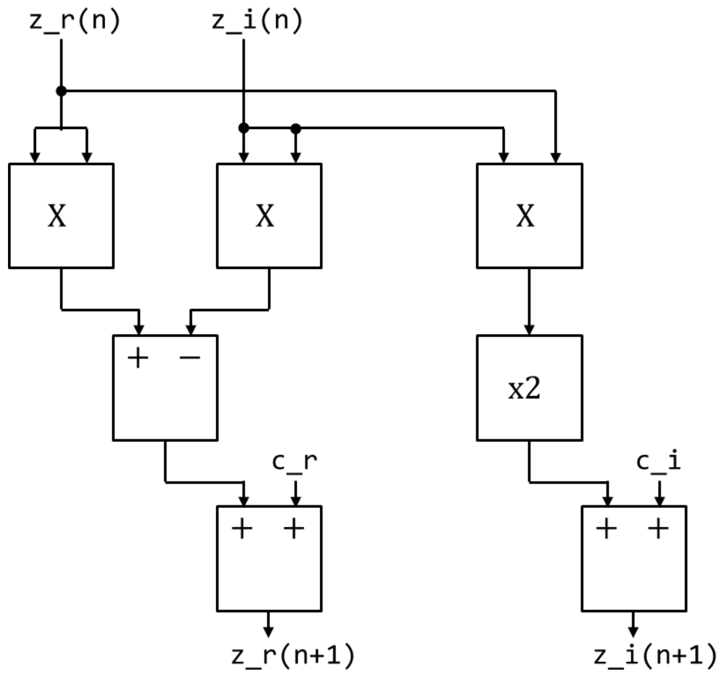
- Approach: run the iteration until $|z_{n+1}|^2 > 4$. The number of iterations is mapped to a colour

Mandelbrot Datapath

- **Mandelbrot algorithm specification as pseudo code:**
  - Rewrite complex numbers based on real- and imaginary-parts
  - Iterations limited to a maximum `MAX_ITER`

```
GIVEN AT DESIGN TIME : MAX_ITER
INPUTS : c_r , c_i ;
OUTPUT: n ;
z_r=c_r;
z_i=c_i;
n = 1;
While ((z_r * z_r + z_i * z_i) < 4 & n<MAX_ITER) {
   z_r' = z_r * z_r - z_i * z_i + c_r;
   z_i = 2 * z_r * z_i + c_i;
   z_r = z_r';
   n = n + 1;
}
```

- **Datapath core:**



EE-334
Digital System Design

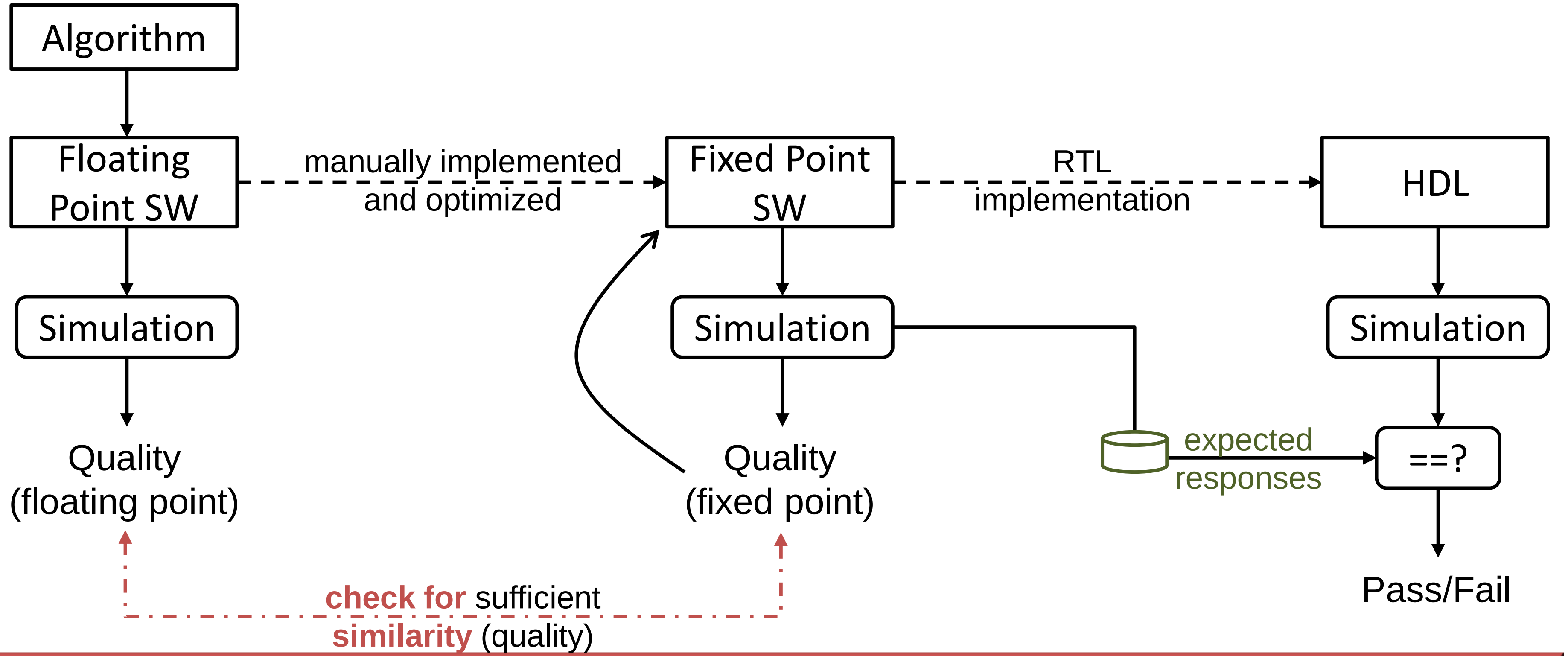
Custom Digital Circuits

VHDL Fixed-Point Arithmetic

Andreas Burg

Fixed Point Design Methodology

- **Algorithms are usually developed based on floating-point arithmetic**
- **Manual refinement to fixed point for fixed-point simulations**
  - **Check the quality of the fixed-point implementation against expectations**
  - **Produce expected responses for RTL verification**
- **RTL code: expected to match the fixed-point model**



Fixed-Point Number Representation

- Integers often do not provide sufficient accuracy and floating-point arithmetic is highly complex to implement in hardware

- **Fixed-Point binary numbers represent fractional numbers** in a similar way as integers (with a fixed dynamic range and precision)

Unsigned Integer

B integer bits

$$\overset{2^{B-1}}{x_{B-1}}\overset{2^{B-2}}{x_{B-2}}\cdots\overset{2^{0}}{x_0}$$

$$x = \sum_{n=0}^{B-1} x_n \cdot 2^n$$

Unsigned Fixed-Point

M integer, N fractional bits, $\mathrm{B} = \mathrm{M} + N$

$$\overset{2^{M-1}}{y_{B-1}}\overset{2^{M-2}}{y_{B-2}}\cdots\overset{2^{0}}{y_{B-M}}.\overset{2^{-1}}{y_{N-1}}\overset{2^{-2}}{y_{N-2}}\cdots\overset{2^{-N}}{y_0}$$

decimal point

$$y = \sum_{n=0}^{B-1} y_n \cdot 2^{n-N}$$

- Fixed-Point signed numbers: add a sign bit and follow the same scheme as integers

The Q-Notation

- Fixed-Point numbers are specified by
  - The type + number of integer and fractional bits
  - The type + total number of bits and the number of fractional bits

- The Q-notation denotes
  - Signed fixed-point numbers as : `Q<N_int>.<N_frac>` where
    `N_int` is the number of integer bits (excluding the sign bit) and
    `N_frac` is the number of fractional bits

$$B = N_{int} + N_{frac} + 1$$

  - Unsigned fixed-point numbers as : `UQ<N_int>.<N_frac>` where
    `N_int` is the number of integer bits and
    `N_frac` is the number of fractional bits

$$B = N_{int} + N_{frac}$$

Fixed-Point Arithmetic with Integers

- Unfortunately, **computers and many languages** (including VHDL) can natively **only handle integers**

- Luckily, the binary representation of fixed-point numbers is the same as integers

- We therefore **represent fixed-point numbers as integers by scaling them with 2^{N_frac}**

$$y_{B-1}y_{B-2}\cdots y_{B-M}\,.\,y_{N-1}y_{N-2}\cdots y_0 \qquad\qquad y_{B-1}y_{B-2}\cdots y_{B-M}y_{N-1}y_{N-2}\cdots y_0\,.$$

$$y = \sum_{n=0}^{B-1} y_n \cdot 2^{n-N} \xrightarrow[\boxed{Y = 2^{N\_frac} \cdot y}]{\cdot\, 2^{N\_frac}} Y = \sum_{n=0}^{B-1} y_n \cdot 2^{n-N\_frac} \cdot 2^{N\_frac} = \sum_{n=0}^{B-1} y_n \cdot 2^n$$

- **<u>NOTE</u>: numerical value of Y is a scaled version of the fixed-point number y !!**

Adding Int Representations of FixP Numbers

- We want to add two FixP numbers x $[\mathtt{Q}N_{int}^{x}.\ N_{frac}^{x}]$ and y $[\mathtt{Q}N_{int}^{y}.\ N_{frac}^{y}]$ with integer representations $X = x \cdot 2^{N_{frac}^{x}}$ and $Y = y \cdot 2^{N_{frac}^{y}}$ with $N_{frac}^{y} < N_{frac}^{x}$
  - First "harmonize" the scaling of X and Y to the same scaling factor $\max\left\{N_{frac}^{x}, N_{frac}^{y}\right\}$ by scaling the integer number with the smaller scale factor with $2^{\max\left\{N_{frac}^{x}, N_{frac}^{y}\right\} - \min\left\{N_{frac}^{x}, N_{frac}^{y}\right\}}$
  - Add the scaled integer to the other integer to get Z
  - The integer result Z has $N_{frac}^{z} = \max\left\{N_{frac}^{x}, N_{frac}^{y}\right\}$

$$z = x + y = \frac{X}{2^{N_{frac}^{x}}} + \frac{Y}{2^{N_{frac}^{y}}} = \frac{X}{2^{N_{frac}^{x}}} + \frac{Y}{2^{N_{frac}^{y}}} \cdot \frac{2^{\left(N_{frac}^{x} - N_{frac}^{y}\right)}}{2^{\left(N_{frac}^{x} - N_{frac}^{y}\right)}}$$

$$= \frac{X + Y \cdot 2^{\left(N_{frac}^{x} - N_{frac}^{y}\right)}}{2^{N_{frac}^{x}}}$$

Adding Int Representations of FixP Numbers

- Example:
  - Add x [U2.2]=1.25='01.01' and y [U5.3]=2.125='00010.001'
  - Integer representations:
    $X = x \cdot 2^2 = 5 =' 0101.'$ and $Y = y \cdot 2^3 = 17 =' 00010001.'$
  - $Z = X \cdot 2^1 + Y = 27$
  - $z = 1.25 + 2.125 = 3.375 = {}^{Z}/_{2^3} = {}^{27}/_{8}$

```
x      01.01
y  00010.001
   00011.011
```

```
X       0101
Y   00010001
    00010110
```

not aligned

```
X*2    01010
Y   00010001
    00011011
```

re aligned integers

- After scaling decimal points of the operands are aligned

Before scaling

$x_1 \ x_0 . x_{-1} x_{-2}$

$y_3 y_2 y_1 y_0 . y_{-1} y_{-2} y_{-3}$

After scaling

$x_1 x_0 . x_{-1} x_{-2} \ 0$

$y_3 y_2 y_1 y_0 . y_{-1} y_{-2} y_{-3}$

Addition: Accuracy and Overflows

- To represent any sum of two *N* bit numbers we require *N+1* bits
  - Example: 4-bit unsigned integers: *1111 (15) + 1111 (15) = 10000 (30: 5 bit)*
  - Example: 4-bit signed integers: *01111 (+15) + 01111 (+15) = 010000 (+16: 5+1 bit)*
    10000 (-16) + 10000 (-16) = 100000 (-32: 5+1 bit)

- For fixed point numbers, only the number of MSBs grows by one, but the number of fractional bits of the result remains the same as for the operands

```
 SIIIII.FFFFFF
 SIIIII.FFFFFF
SIIIIII.FFFFFF
```

  - Same as for decimal numbers: 3.756 + 8.111 = 11.867

Arithmetic Operations in VHDL

- **VHDL supports basic arithmetic operations on** `signed` and `unsigned` **integers** with the `numeric_std` package
  - **Operators define the width of result** (as function of the operands)
  - **Some operators put constraints on the** word-length of the **operands**

- **Addition**: both operands must be of same length and type

```
signed(N-1 downto 0) <= signed(N-1 downto 0) + signed(N-1 downto 0)
```

  - If overflows should be avoided, *N* bit operands must be resized first to *L=N+1* bits, adding the missing most-significant-bit (MSB) to "catch" potential overflows
  - Resizing (with sign extension for `signed`) adds MSBs:

```
signed(L-1 downto 0) <= resize(signed(N-1 downto 0),L)
```

SSIIIII.FFFFFF

SSIIIII.FFFFFF

SIIIIII.FFFFFF

Multiplying Int Representations of FixP Numbers

- We want to multiply two FixP numbers x $[\mathtt{Q}N_{int}^{x}.\ N_{frac}^{x}]$ and y $[\mathtt{Q}N_{int}^{y}.\ N_{frac}^{y}]$ with integer representations $X = x \cdot 2^{N_{frac}^{x}}$ and $Y = y \cdot 2^{N_{frac}^{y}}$
  - Simply multiply the integer representations $Z = X * Y$
  - The integer representation of the result is scaled by $2^{N_{frac}^{x}+N_{frac}^{y}}$ (i.e., has $N_{frac}^{z} = N_{frac}^{x} + N_{frac}^{y}$)

$$z = x * y = \frac{X}{2^{N_{frac}^{x}}} * \frac{Y}{2^{N_{frac}^{y}}} = \frac{X * Y}{2^{N_{frac}^{x}+N_{frac}^{y}}}$$

Multiplication: Accuracy and Overflows

- When multiplying an ***N*** bit and an ***M*** bit integer numbers (signed or unsigned), the result requires ***N+M*** bits:
  - Example: 4-bit unsigned integers: `1111 (15) + 1111 (15) = 1110 0001 (225)`
  - Example: 4-bit signed integers:
    ```
    01111 (+15) + 01111 (+15) = 00 1110 0001 (+225)
    10000 (-16) + 01111 (+15) = 11 0001 0000 (-240)
    10000 (-16) + 10000 (-16) = 01 0000 0000 (+256)
    ```

- For fixed point numbers, both the number of MSBs and the number of LSBs grows:
- Assume x [QN_{int}^{x}. N_{frac}^{x}] and y [QN_{int}^{y}. N_{frac}^{y}]: for $z = y * x$ we need z [Q$N_{int}^{x} + N_{int}^{y} + 1$.$N_{frac}^{x} + N_{frac}^{y}$]

`SII.FFF * SI.FF = SIIII.FFFFF`

Truncation and Rounding

- Full-precision fixed-point multiplications increase the number of fractional bits
  - For $z = y * x$: $N_{frac}^{z} = N_{frac}^{x} + N_{frac}^{y}$
  - Repeated multiplications of fixed-point numbers leads to a growing number of bits

- Often, we do not require an ever increasing accuracy: **least-significant-bits (LSBs) can sometimes be removed**, accepting a potentially small error
- Two methods to remove LSBs and avoid growing word-length
  - Truncation: simply remove LSBs

    SSIIIII.FFFFFF ➔ *SSIIIII.FF* *SSIIIII.FF~~FFFF~~*

  - Rounding: add ½ LSB (referred to the result) and truncate then

    SSIIIII.FFFFFF
    0000000.001000
    SSIIIII.FF

    LSB of result

Arithmetic Operations in VHDL

- **VHDL supports basic arithmetic operations on** `signed` and `unsigned` **integers** with the `numeric_std` package
  - **Operators define the width of result** (as function of the operands)
  - **Some operators put constraints on the** word-length of the **operands**

- **Multiplication**: both operands must be of same type (sgn/uns), but can have different length (number of bits)

```
signed(N+M-1 downto 0) <= signed(N-1 downto 0) * signed(M-1 downto 0)
```

  - Rounding and truncation of LSBs can be realized by simply removing ***P*** LSBs

```
signed(N+M-P-1 downto 0) <= signed(N+M-1 downto P)
```