RTR532 2018 Spring Lecture 04

- **Fixed point arithmetic**
- **Digital filters with VHDL I**
- **Freeform signal generation using VHDL**

Fixed point arithmetic

The BIG problem

- Fixed bit width
- Interpretation
- When can we truncate the result of addition / multiplication

- Why use fixed point at all ???

Binary number ***representation***

	No fraction	**Fraction**
Unsigned	Min – 0 Max – 2^(N) - 1 Each n-th bit is 2^n decimal value	Min – 0, Max – 2^N - 1 (depends on radix point) Each n-th bit to the left of radix is 2^n decimal value Each n-th bit to the right of radix is 2^(-n) decimal value
Signed	Min – -2^(N-1) Max – 2^(N-1) - 1 Each n-th bit is 2^n decimal value MSB bit – sign (0 plus, 1 minus)	Min – -2^(N-1), Max – 2^(N-1) – 1 (depends on radix point) Each n-th bit to the left of radix is 2^n decimal value Each n-th bit to the right of radix is 2^(-n) decimal value MSB bit – sign (0 plus, 1 minus)

Binary number ***representation***

No Fraction, unsigned

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	1

$2^7+2^6+2^4+2^1+2^0=211$

Fraction, unsigned

2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
1	1	0	1	0	0	1	1

$2^5+2^4+2^2+2^{-1}+2^{-2}=52.75$

No Fraction, signed

S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	1

$-2^7+2^6+2^4+2^1+2^0=$
$= -128+64+16+2+1 = -45$

Fraction, signed

S	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
1	1	0	1	0	0	1	1

$-2^5+2^4+2^2+2^{-1}+2^{-2}=$
$= -32+16+4+0.5+0.25 = -11.25$

Whiteboard example:

110101 (no fraction, unsigned)
110101 (no fraction, signed)
11.0101 (fraction, unsigned)
11.0101 (fraction, signed)

Binary addition

No fraction

0+0 = 0, no carry
1+0 = 1, no carry
0+1 = 1, no carry
1+1 = 0, carry 1

Whiteboard example:
01.01
+ 110.1

With fraction

- Align the radix point
- Fill blank spaces to the left with 0 or 1 (signed or unsigned)
- Fill blank space to the right with 0
- Continue as with no fraction
- Radix point stays at the location
- https://courses.cs.vt.edu/csonline/NumberSystems/Lessons/AddingBinaryNumbersWithFractions/index.html

Binary addition – signal width considerations

Signal A,B,C : unsigned(3 downto 0);

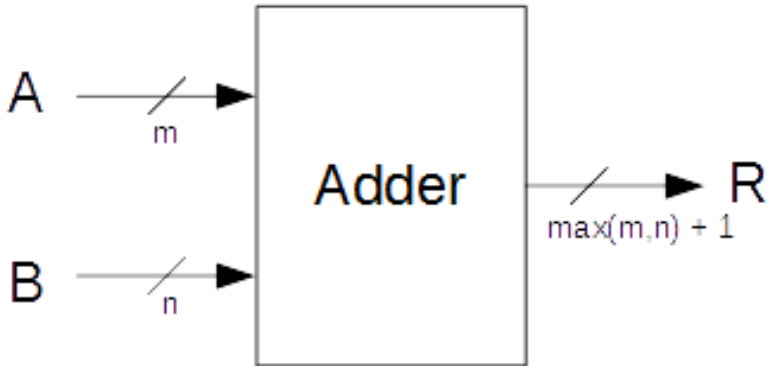
C <= A + B; --VHDL will allow this! **Overflow can happen**

Signal A,B : unsigned(3 downto 0);
Signal C : unsigned(4 downto 0);

C <= A(A'high) & A + B(B'high) & B;

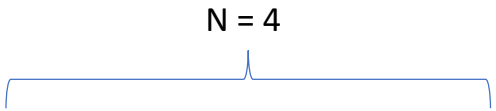
No overflow can happen

Better to always assign addition to wider vector! You can always extract required bits later.



Useful things to know with addition

- 1 + 1 = 0, carry 1
- 1 + 1 + 1 = 1, carry 1
- Consider sum of N ones:



  - Even number - result is zero, number of carry bits is N/2
  - Odd number – result is one, number of carry bits is (N-1)/2

Binary subtraction

No fraction

0-0 = 0

1-0 = 1

0-1 = 1, borrow from left

1-1 = 0

Whiteboard example:
01.01
- 110.1

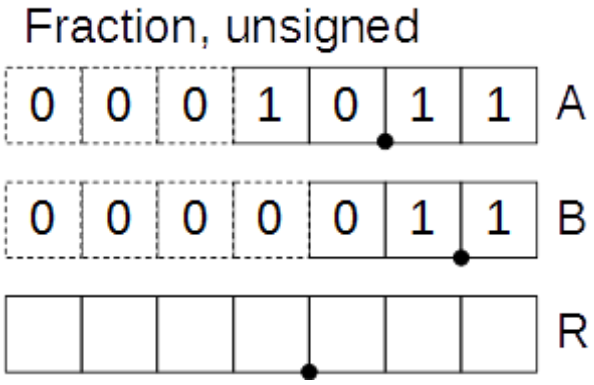
With fraction

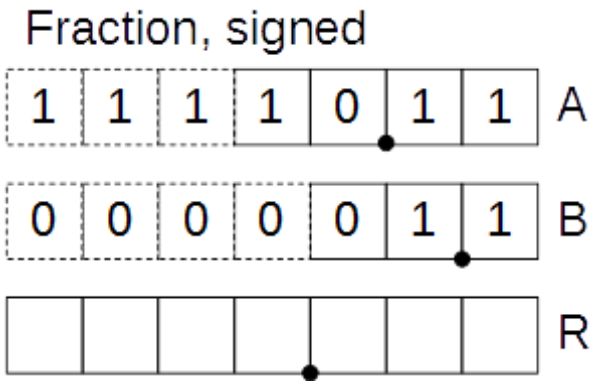
- Align the radix point
- Fill blank spaces to the left with 0 or 1 (signed or unsigned)
- Fill blank spaces to the right with 0
- Do as No fraction
- Radix point stays at the location

Binary multiplication

```
  1011.01
x   110.1
```

1. Add leading zeros (unsigned or plus) or leading ones (minus) – so that both inputs (m, n) match the length of output (m+n)
2. Multiply one by one
3. Sum all together

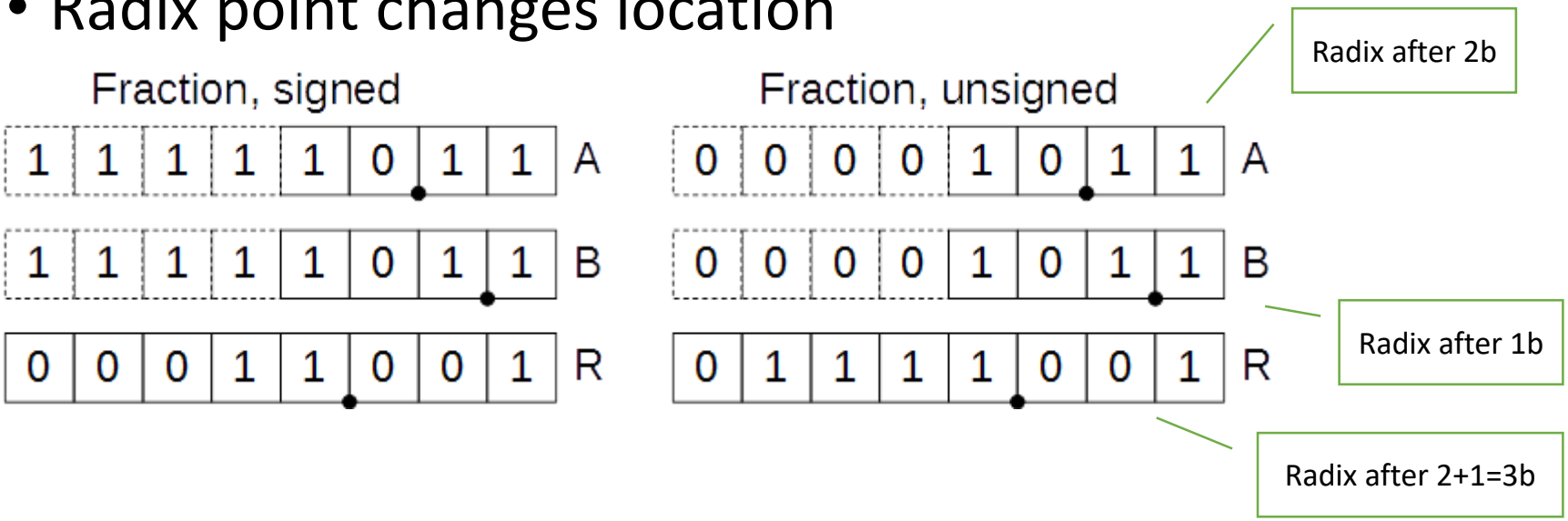




Binary multiplication - fractions

```
  1011.01
x   110.1
```

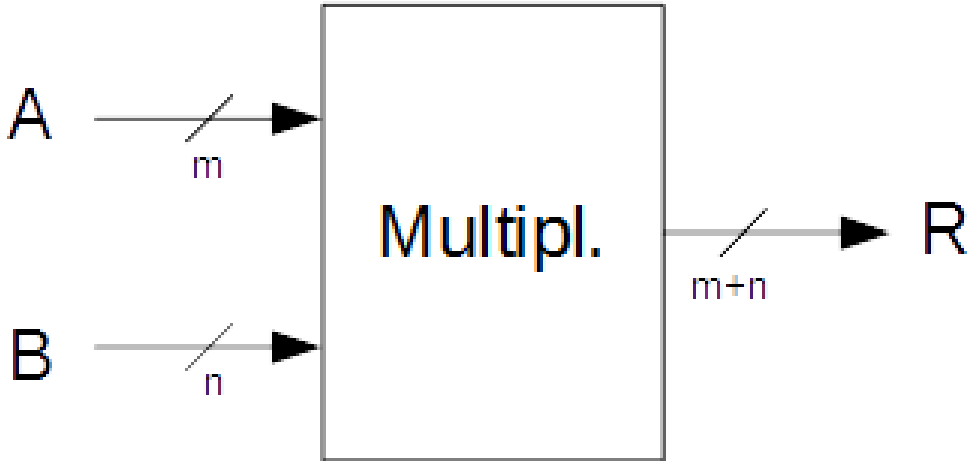
- Radix points does not require alignment
- Radix point changes location



Binary multiplication – signal width considerations

```
Signal A,B,C : unsigned(3 downto 0);
---
C <= A * B; --VHDL will NOT allow this!
```

```
Signal A,B : unsigned(3 downto 0);
Signal C : unsigned(7 downto 0);
---
C <= A * B; --OK now
```

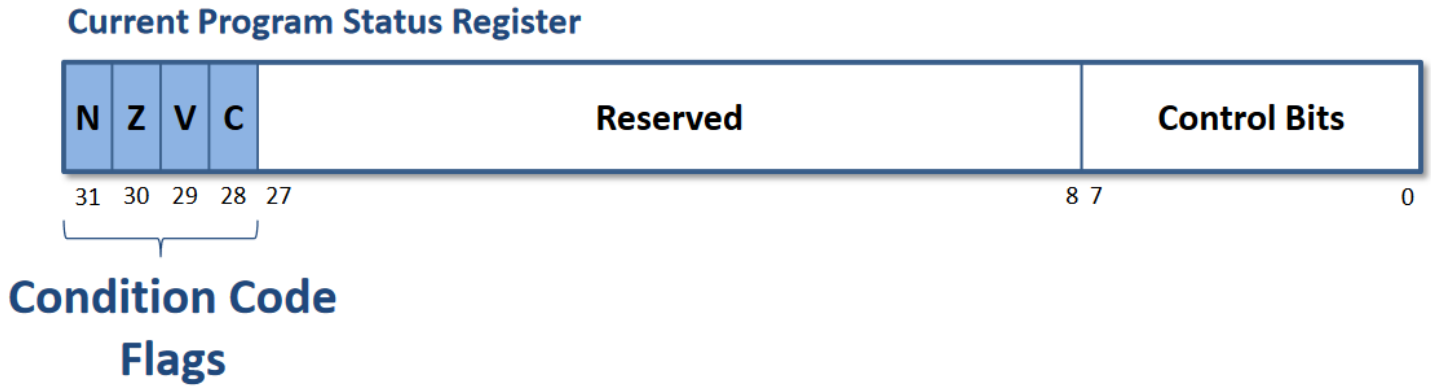


Good practice

- Use no fractions
- Use fractions to represent 0.zzzz numbers (whole number except sign is fraction) – useful in FIR filter design
- Use fractions in between – good luck !

Condition bits (flags) in microprocessor systems – Negative, Zero, Overflow, Carry ???

- In microprocessor systems there are fixed width signals, e.g. status register
- N – **N**egative, Z- **Z**ero, V – o**V**erflow, C – **C**arry
- Image from https://www.scss.tcd.ie/~waldroj/3d1/04-Arithmetic.pdf



Additional references

- https://www.swarthmore.edu/NatSci/echeeve1/Ref/BinaryMath/BinaryMath.html
- https://www.scss.tcd.ie/~waldroj/3d1/04-Arithmetic.pdf
- http://cs.furman.edu/digitaldomain/more/ch6/dec_frac_to_bin.htm
- http://ba.mirror.garr.it/1/groundup/a36.html#almltitle60

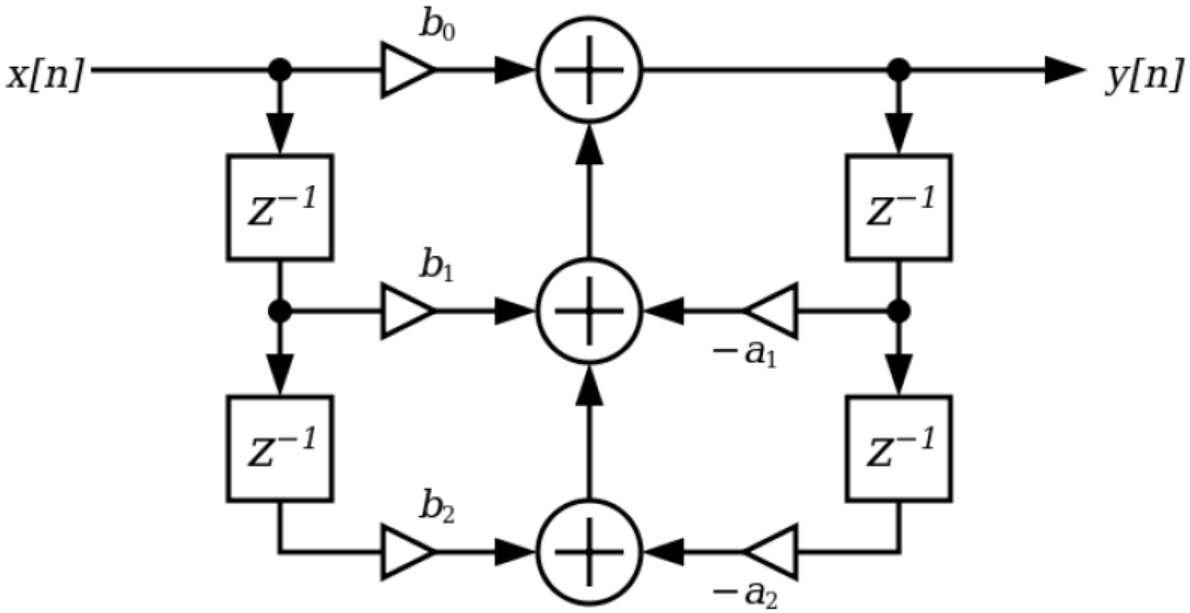
Digital filters with VHDL I

Filter defined by transfer function

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + A_M z^{-M}}$$

- But how to implement it in hardware ?

Direct Form I – FIR and IIR parts



Direct Form I - FIR Quartus example

- Input 8b signed (min -128, max 127) [0…1]
- Delay – one clock register line (we know how to make it, right?)
- Coefficients – 8b
- Output (untruncated) – 18b

- How can we get coefficients ? MATLAB filter design (fir1, fir2, …) !
  - Assume 3rd order FIR filter: b0 = 0.5, b1 = 0.75, b2 = -0.3
- Use binary fraction to represent values

Freeform signal generation using VHDL

Also called …

DDS or Direct Digital Synthesis

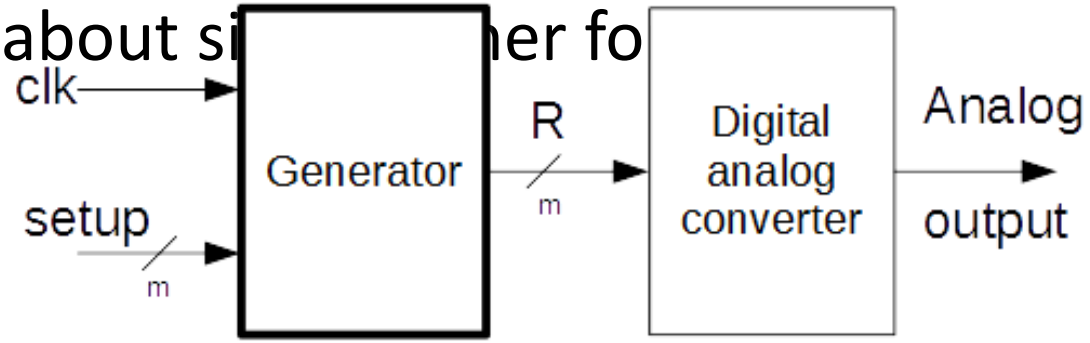
DDS (direct digital synthesis) ? I think I have heard that before…

- 33210A function generator from Keysight (ex Agilent, ex HP)
- DG1000 waveform generator from Rigol
- Arduino + AD9850 30MHz DDS generator for 12$

- Used extensively to generate arbitrary (free form) digital signal!

Basic principle

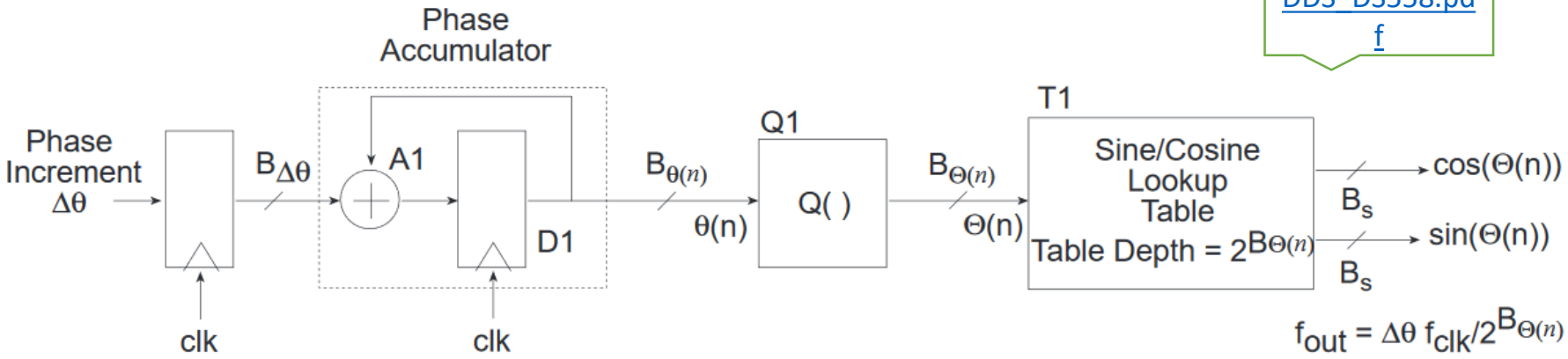
- Generator outputs (passes) digital signal values to digital analog converter
- We could already make generators for: rectangle, triangle, sawtooth.
- What about si ner fo



DDS – generalized method of generation

- Phase increment – determines output frequency
- Phase accumulator – the «counter»
- Look-up table – memory where waveform is stored

Fig from Xilinx DDS_DS558.pdf



DDS parameters and calculations

- Input:
  - Clock frequency f_{clk}
  - Phase increment $\Delta\theta$
  - Phase accumulator $B_{\theta(n)}$
  - Memory size and contents
- Calculable parameters
  - Signal output frequency $f_{out} = \frac{f_{clk} * \Delta\theta}{2^{B_{\theta(n)}}}$
  - Frequency resolution $\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}}$
  - Phase increment $\Delta\theta = \frac{f_{out}\ 2^{B_{\theta(n)}}}{f_{clk}}$

1. Output frequency

$f_{clk} = 120MHz$
$B_{\theta(n)} = 10$
$\Delta\theta = 12_{10}$

2. Frequency resolution

$f_{clk} = 120\ MHz$
$B_{\theta(n)} = 32$

3. Phase increment

$f_{clk} = 100\ MHz$
$B_{\theta(n)} = 18$
$f_{out} = 19\ MHz$

Lets do some Quartus project!

- Use files in DDS_VHDL
  - Dds_vhdl.vhd
  - Dds_vhdl_tb.vhd

- Fill in required code