FPGA tutorial

Lecture 7

09.12.2020

Jochen Steinmann

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY


Solution

BCD + Counter

Solution

```
ARCHITECTURE tb OF bcd_tb IS

    COMPONENT bcd is
    --    GENERIC (
    --    );
        port(
            i_slv_binary   :   in std_logic_vector( 7     downto 0);
            o_slv_decimal  :  out std_logic_vector( 3*4-1 downto 0)
        );
    end component bcd;

    component counter is
        generic(
            DATA_WIDTH    :  integer := 8
        );
        port(
            i_sl_CLK      :  in  std_logic;                                  -- clock
            i_sl_en       :  in  std_logic;                                  -- enable
            i_sl_dir      :  in  std_logic;                                  -- direction
            i_sl_rst      :  in  std_logic;                                  -- reset
            o_slv_counter :  out std_logic_vector(DATA_WIDTH-1  downto 0)    -- counter
        );
    end component counter;

    -- "local" signals

    -- signals for UUT
    -- input
    signal binary   : std_logic_vector( 7 downto 0) := (others => '0') ;

    signal clk : std_logic := '0';

    -- output
    signal decimal  : std_logic_vector( 3*4-1 downto 0);

    signal ones : std_logic_vector(3 downto 0);
    signal tens : std_logic_vector(3 downto 0);
    signal hundrets : std_logic_vector(3 downto 0);
```

```
BEGIN
    -- Instance of unit under test.
    bcd1: bcd
        port map(
            i_slv_binary  => binary,
            o_slv_decimal => decimal
        );

    counter1: counter
        port map(
            i_sl_CLK      => clk,
            i_sl_en       => '1',
            i_sl_dir      => '0',
            i_sl_rst      => '0',
            o_slv_counter => binary
    );


    ones     <= decimal(3 downto 0);
    tens     <= decimal(7 downto 4);
    hundrets <= decimal(11 downto 8);

-------------------------------------------------
    -- test bench definition.
    tb_test :process
    begin
        clk <= not clk;
        wait for 1 ns;

    end process;

END;
```

MultiBitShift

Part 1

```
library ieee;
use ieee.std_logic_1164.all;

package pkg is
    type slv8_array_t is array (natural range <>) of std_logic_vector(7 downto 0);
end package;

package body pkg is
end package body;

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.pkg.all;

----------------------------------------

entity MultiBitShift is
    port(
        i_sl_CLK     :  in  std_logic;                     -- clock
        i_sl_en      :  in  std_logic;                     -- enable
        i_slv_data   :  in  std_logic_vector(7 downto 0);  -- data input
        o_slv_shift  :  out slv8_array_t(7 downto 0)       -- last X samples
    );
end MultiBitShift;
```

MultiBitShift

Part 2

```
architecture behavior of MultiBitShift is
    signal shift_reg : slv8_array_t(7 downto 0) := (others=>(others=>'0'));
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                for I in 7 downto 1 loop
                    shift_reg(I) <= shift_reg( I - 1);
                end loop;
                shift_reg(0) <= i_slv_data;
            end if;
        end if;
    end process;
    o_slv_shift <= shift_reg;

end behavior;
```

MultiBitShift - testbench

Part 3

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.pkg.all;

ENTITY MultiBitShift_tb IS
END MultiBitShift_tb;

ARCHITECTURE tb OF MultiBitShift_tb IS

    COMPONENT MultiBitShift is
        port(
        i_sl_CLK     :  in  std_logic;                      -- clock
        i_sl_en      :  in  std_logic;                      -- enable
        i_slv_data   :  in  std_logic_vector(7 downto 0);    -- data input
        o_slv_shift  :  out slv8_array_t(7 downto 0)        -- last X samples
        );
    end COMPONENT;

    -- "local" signals

    -- signals for UUT
    -- input
    signal CLK    : std_logic := '0';
    signal en     : std_logic := '0';

    -- ouput
    signal slv_data  : std_logic_vector(7 downto 0) := (others => '0');
    signal arr_shift : slv8_array_t(7 downto 0);    -- last X samples
```

MultiBitShift

Part 4

```
BEGIN
    -- Instance of unit under test.
    uut: MultiBitShift
    PORT MAP (
        i_sl_CLK    => CLK,
        i_sl_en     => en,
        i_slv_data  => slv_data,
        o_slv_shift => arr_shift
    );

------------------------------------------------------------------
    -- test bench definition.
    tb_CLK :process
    begin
        CLK <= '0';
        wait for 10 ns;
        CLK <= '1';
        wait for 10 ns;
    end process;

    tb_en : process
    begin
        en <= '1';
        wait for 2000 ns;
    end process;

    tb_dir : process
    begin
        for data in 0 to 255 loop
            slv_data <= std_logic_vector(to_unsigned( data, 8));
            wait for 20 ns;
        end loop;
    end process;

END;
```


NEW

VHDL

III. Physikalisches Institut B
RWTH AACHEN UNIVERSITY

How to initialise an array

- When we first used the arrays, we could init them to zero
  - signal arr : slv8_array_t(3 downto 0) := (others => (others => ‘0’));
- What if we would like to set the values for each entry in a different way?
  - signal arr : slv8_array_t(3 downto 0)
    := (0 => x"01", 1 => x"02", 2 => x"03", 3 => x"04");

Up to now, we used binary initialization, x“01“ is hexadecimal.

Digital Signal Processing

DSP

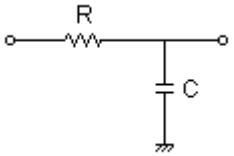
III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

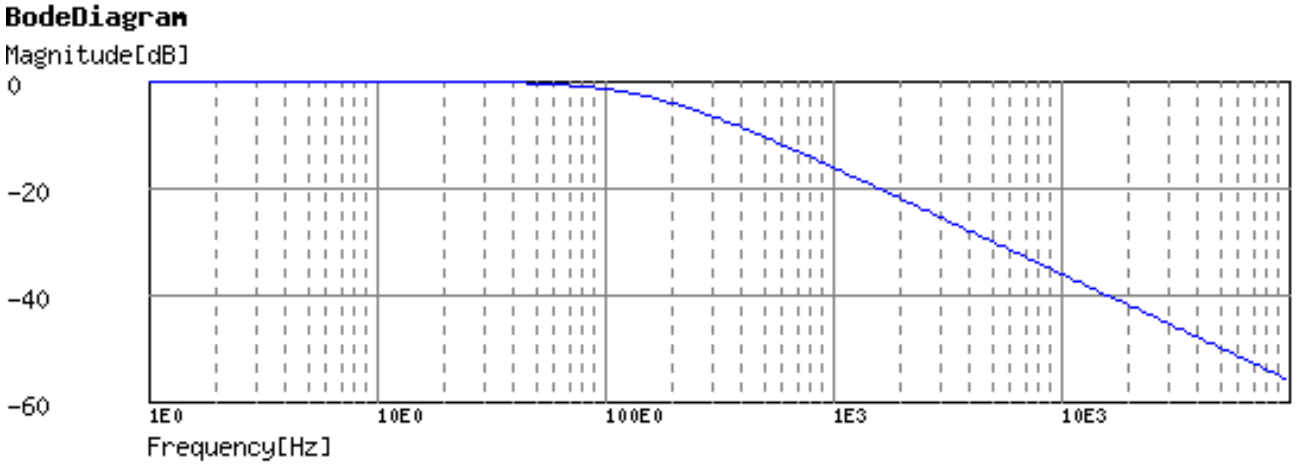
Analog Filter

Recap

- All of you know the passive analog filters from the first semesters
  - R/C
  - L/C
  - R/L/C
- Let‘s have a look at the R/C filter (low pass)



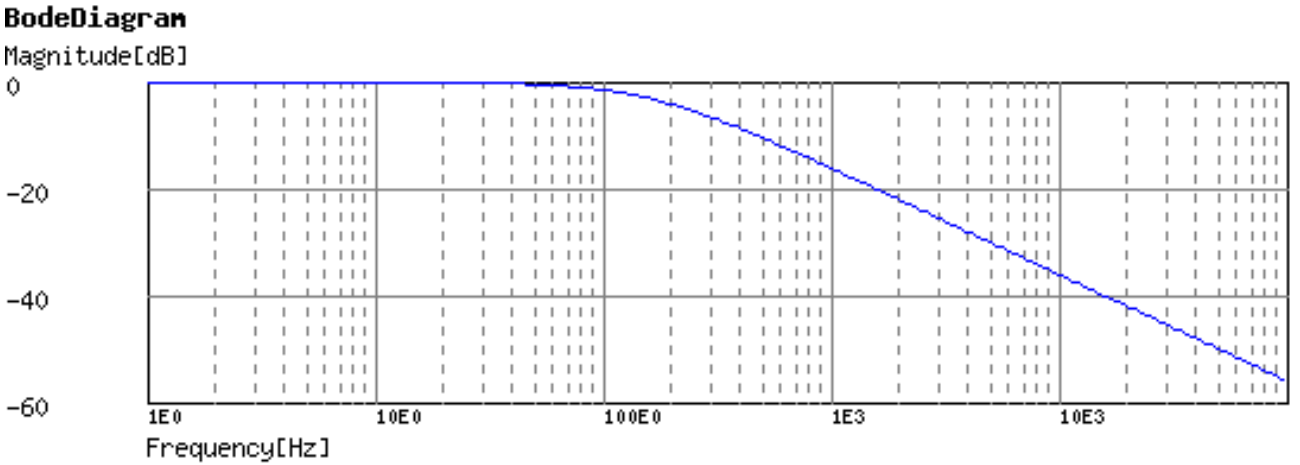
- R = 10 kΩ
- C = 100 nF



Characterization of a Filter

Gain / Attenuation

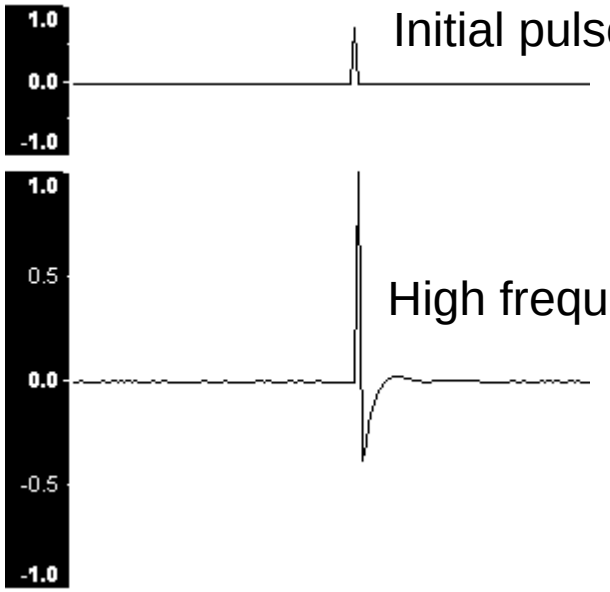
- Easy calculation
  - Ration of output to input voltage
  - Uout / Uin



Characterization of a Filter

Impulse Response

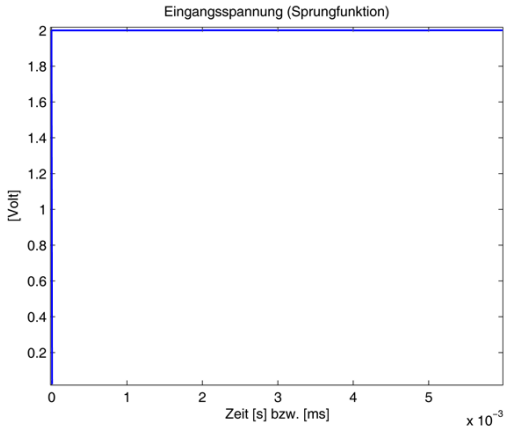
- Response to a tiny pulse (Dirac / Delta function)
- Difficult to get such a pulse in reality (different pulse width)

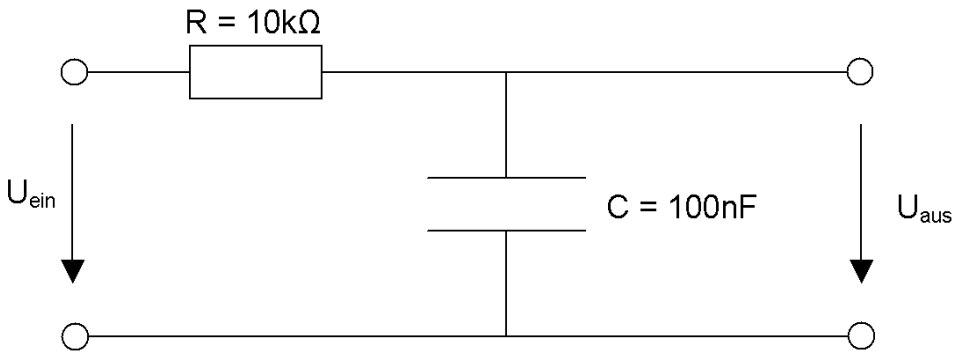


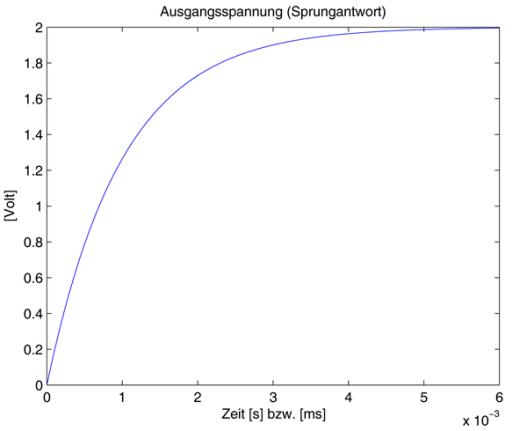
Characterization of a Filter

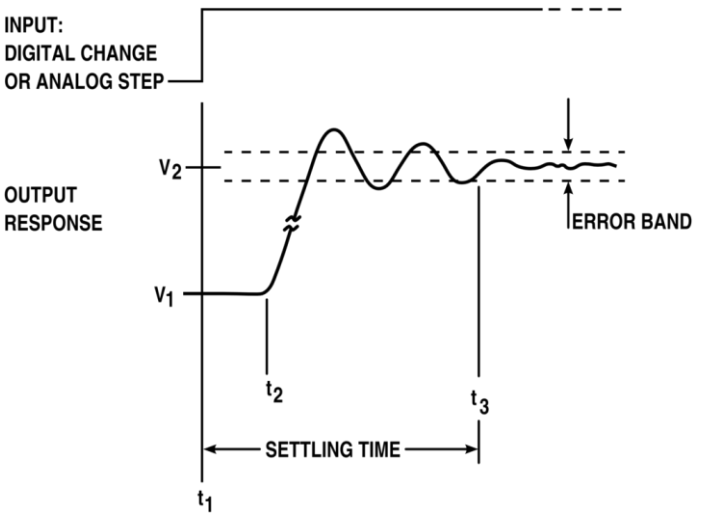
Step Response

- Response of the Filter to a step function at the input
- Much easier to test in the lab









Digital Filter

Two kind of filters

FIR

- Finite Impulse Response
- Stable
- (no) feedback from the output
- Needs more computation effort
- Some fancy features
  - Linear phase relation
- A bit more complex to implement
  - Means in this context more difficult to implement a proper frequency

IIR

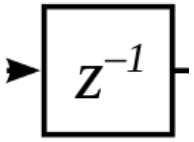
- Infinite Impulse Response
- Most efficient way of implementing filter
- Less computation needed
- Can be changed much easier on the fly
- Mostly implemented in DSP devices

Just some brief introduction / comparison!
Not very detailed.

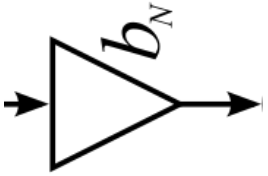
Contents of a Filter

Both are the same for FIR and IIR

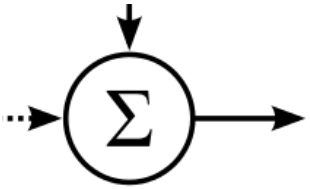
- We are not going to far into signal theory – we just use it
- What we will observe in all filters
- Delay by one clock cycle:



- Multiplication with a factor (here b_N)

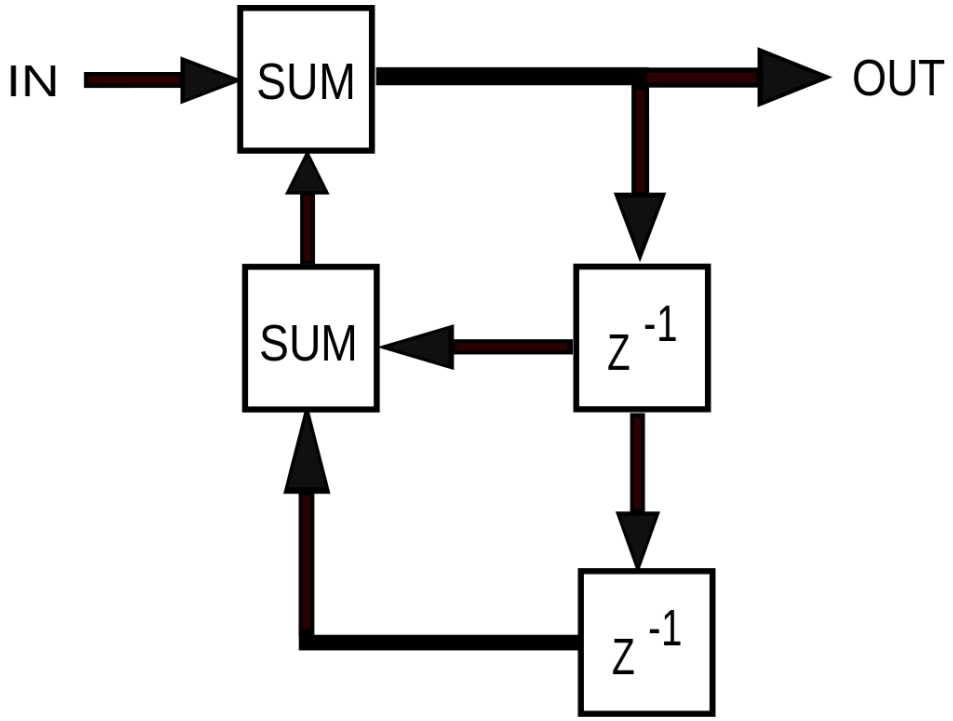


- Addition



Feedback from output

- This is also visible in the name (infinite impulse response)

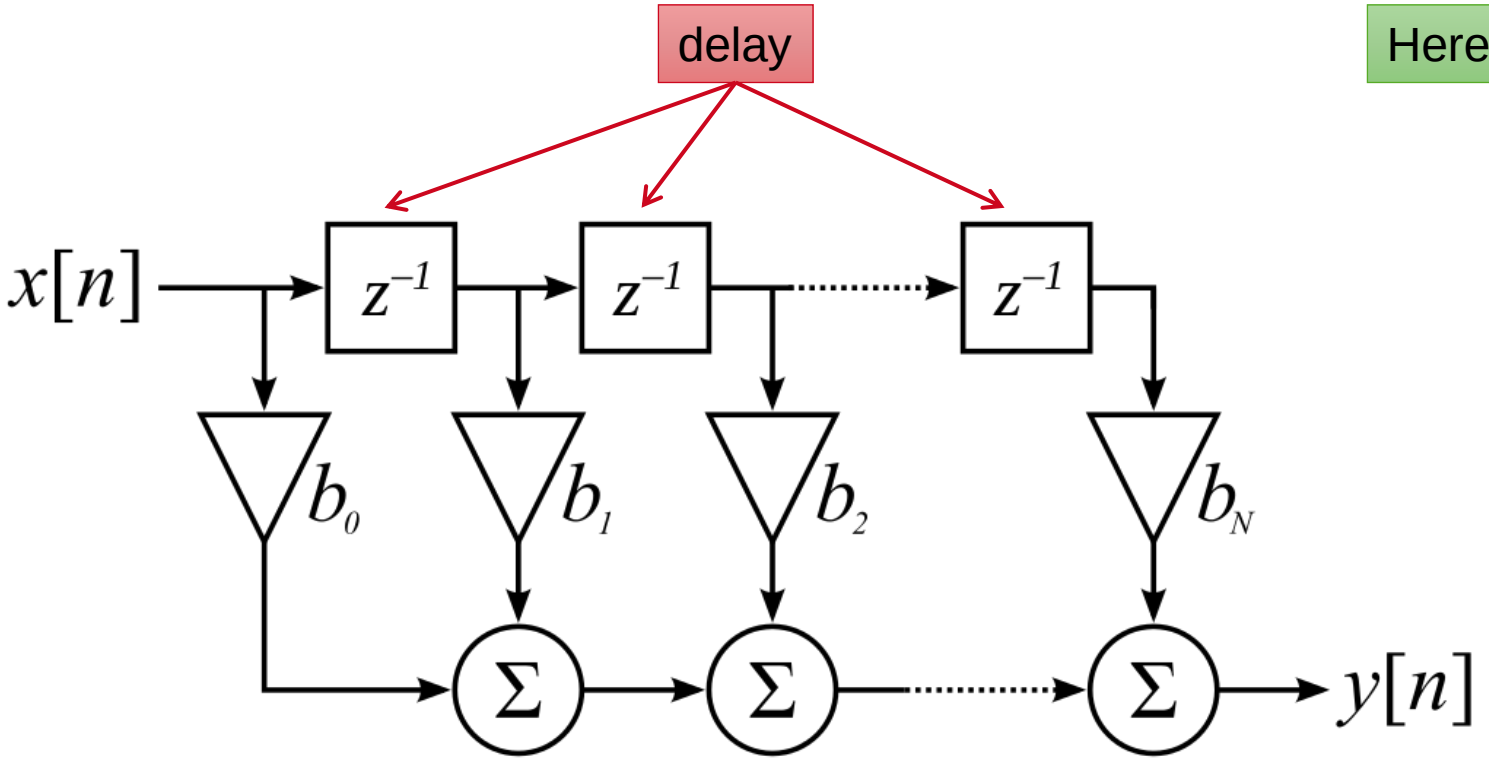


FIR Filter

Special kind of drawing

Order = Number of delays

Here 3rd order



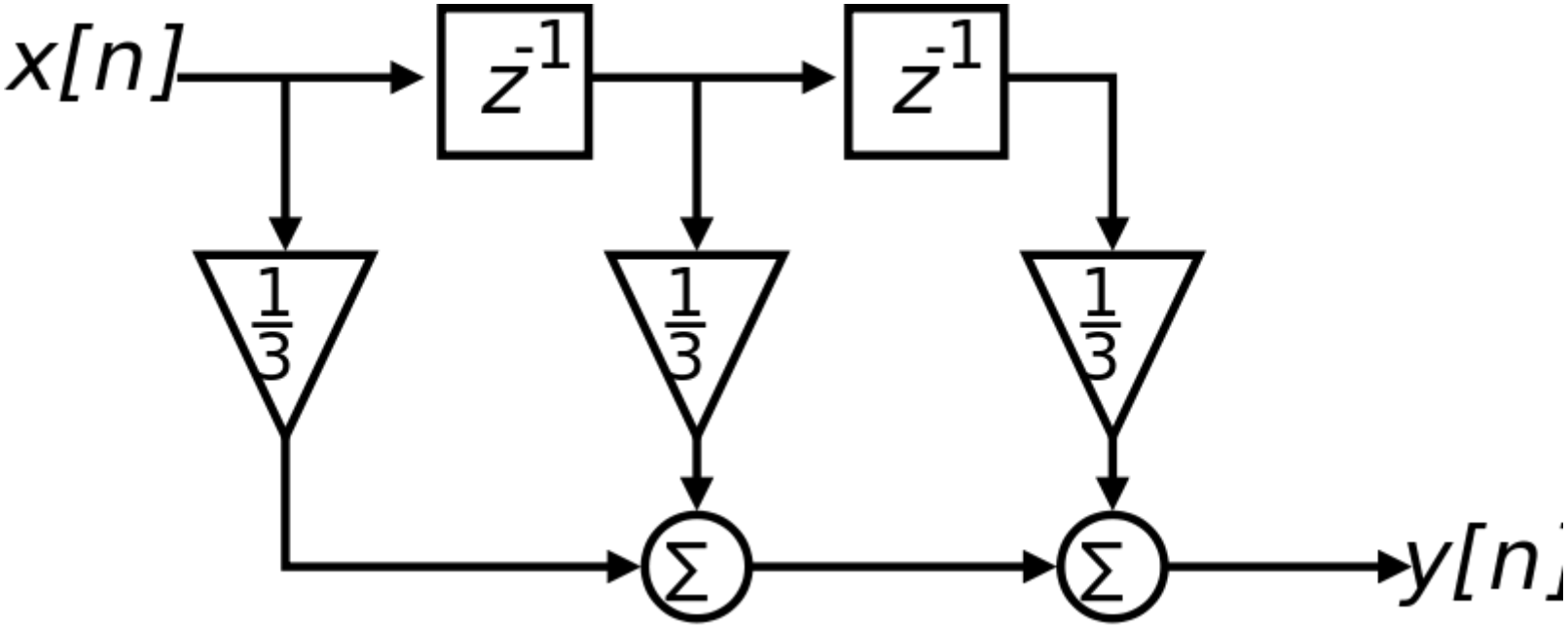
$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$
$$= \sum_{i=0}^{N} b_i \cdot x[n-i],$$

Jochen Steinmann | RWTH Aachen University

Having a look at a FIR filter

Very simple filter

- 2nd order Filter
- 3 taps



Moving Average: Y = 1/3 x[0] + 1/3 x[1] + 1/3 x[2]

$$H(z) = \frac{1}{3} + \frac{1}{3}z^{-1} + \frac{1}{3}z^{-2} = \frac{1}{3}\frac{z^2 + z + 1}{z^2}.$$

Problems when implementing filters

- The example on the previous slide uses float (1/3)
  - There is no float available in the FPGA
  - We have to implement it by using
    - unsigned
    - std_logic_vector

Which one would you choose?

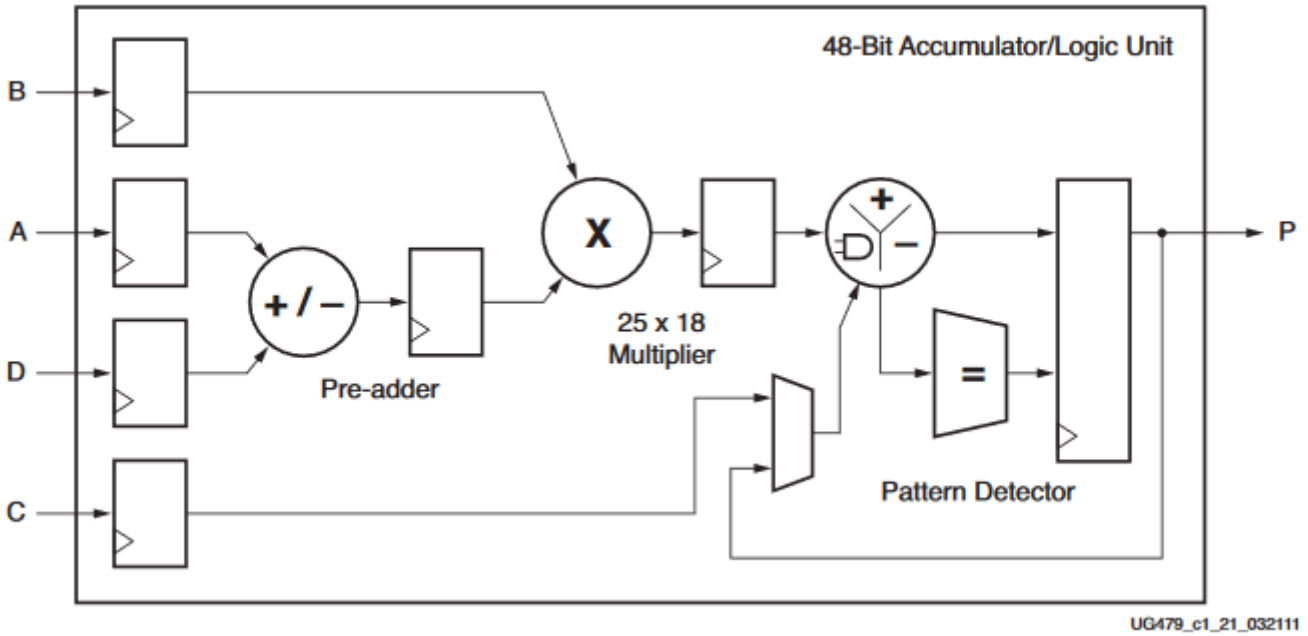
- There is also another limiting factor
  - The division by 3 is difficult to implement (so we change to to 4)
  - Division by 4 can easily been done!

- **Why is it a better choice to calculate the average and divide afterwards?**
  - ¼ * X[0] + ¼ * X[-1] + ¼ * X[-2] + ¼ * X[-3]
  - (X[0] + X[-1] + X[-2] + X[-3]) * ¼

Mathematically the same!

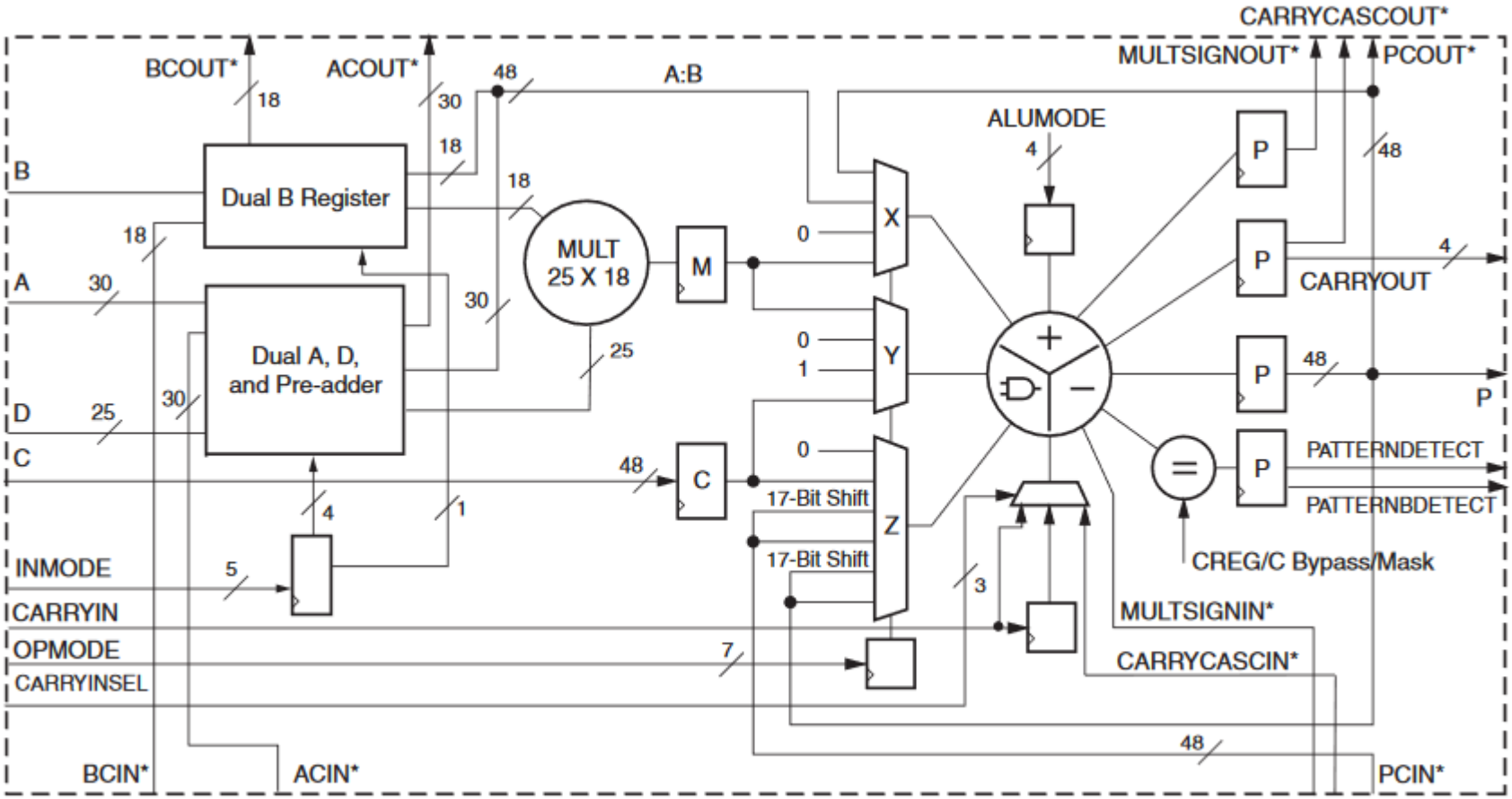
DSP Slice of the FPGA

A bit more complex



Xilinx 7 series

DSP48E1



*These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.

UG369_c1_01_052109

ALUmode?

ALUmode!

DSP

DSP Operation	OPMODE[6:0]	ALUMODE[3:0]			
		3	2	1	0
Z + X + Y + CIN	Any legal OPMODE	0	0	0	0
Z – (X + Y + CIN)	Any legal OPMODE	0	0	1	1
–Z + (X + Y + CIN) – 1 = not (Z) + X + Y + CIN	Any legal OPMODE	0	0	0	1
not (Z + X + Y + CIN) = –Z – X – Y – CIN - 1	Any legal OPMODE	0	0	1	0

Notes:

1. In two's complement: –Z = not (Z) + 1

Pattern detection

Table 2-13: **OPMODE and ALUMODE Control Bits Select Logic Unit Outputs**

Logic Unit Mode	OPMODE[3:2]		ALUMODE[3:0]			
	3	2	3	2	1	0
X XOR Z	0	0	0	1	0	0
X XNOR Z	0	0	0	1	0	1
X XNOR Z	0	0	0	1	1	0
X XOR Z	0	0	0	1	1	1
X AND Z	0	0	1	1	0	0
X AND (NOT Z)	0	0	1	1	0	1
X NAND Z	0	0	1	1	1	0
(NOT X) OR Z	0	0	1	1	1	1
X XNOR Z	1	0	0	1	0	0
X XOR Z	1	0	0	1	0	1
X XOR Z	1	0	0	1	1	0
X XNOR Z	1	0	0	1	1	1
X OR Z	1	0	1	1	0	0
X OR (NOT Z)	1	0	1	1	0	1
X NOR Z	1	0	1	1	1	0
(NOT X) AND Z	1	0	1	1	1	1

https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

For this course

- We are not using the DSP48E1 slice in simulation (maybe later)
- First we stick to the VHDL multiplication and addition

- But we have to talk about data sizes!
  - When summing up 2 x 8 bit, how much bits is the result?
    - **255 (8bit) + 255 (8bit) = 510 (9 bit)**
  - When multiplying 2 x 8 bit, how much bits is the result?
    - N bit x M bit
      - Smallest size for the result: max(N, M)
      - Largest size **N + M**
    - **255 (8 bit) * 255 (8 bit) = 65025 (16 bit)**

The bitsize is increasing a lot….

Today

A. Implementing the moving average FIR filter (N = 3)

B. Implement different filter coefficients (b‘s)
   - B0 = 1
   - B1 = 2
   - B2 = 3
   - B3 = 4

- Plot the step response for both of the filters (gtkwave can do this)

Keep in mind: Write your code, that it can be reused (generic).