FPGA tutorial

Lecture 5

25.11.2020

Jochen Steinmann

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

Discussion

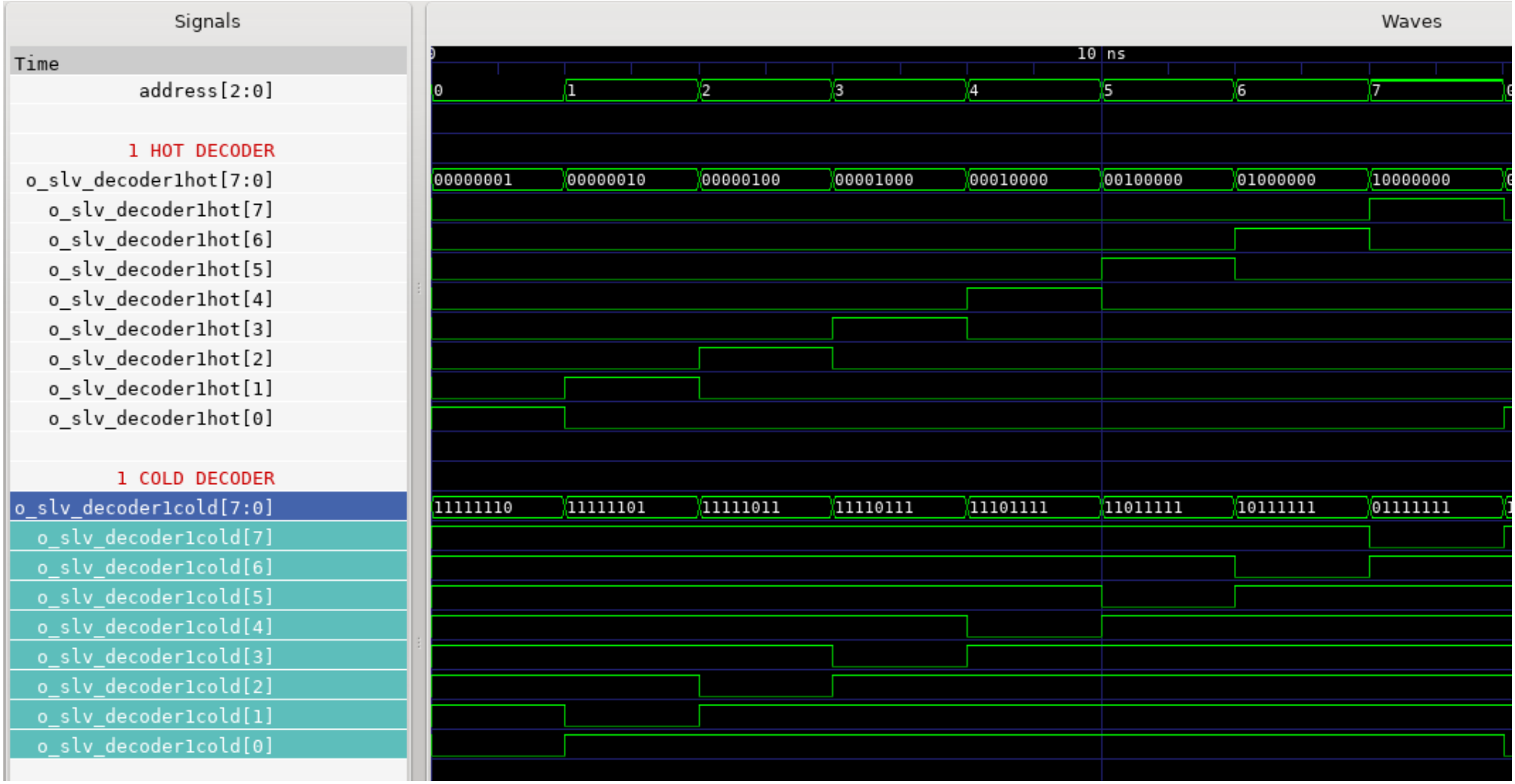
- If there are questions, please ask them
- There is enough time to discuss them.


Solution

Addressdecoder

```
entity addressdecoder is
    generic(
        DATA_BITS     :  integer := 3;
        ONE_HOT       :  std_logic := '1'
    );
    port(
        i_sl_en       :  in  std_logic;                                -- enable
        i_slv_address :  in  std_logic_vector(DATA_BITS-1 downto 0);     -- address input
        o_slv_decoder :  out std_logic_vector( (2**DATA_BITS) -1  downto 0)   -- decoded output
    );
end addressdecoder;

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

architecture behavior of addressdecoder is
    signal adecoded : unsigned( (2**DATA_BITS)-1 downto 0) := (others => '0');  -- define signal, because we cannot readback outputs
begin

    process(i_slv_address, i_sl_en)
    begin
        if (i_sl_en = '1') then
            if ONE_HOT then
                adecoded <= (others => '0');
                adecoded( to_integer( unsigned(i_slv_address))  ) <= '1';
            else
                adecoded <= (others => '1');
                adecoded( to_integer( unsigned(i_slv_address))  ) <= '0';
            end if;
        end if;
    end process;

    o_slv_decoder <= std_logic_vector(adecoded);

end behavior;
```

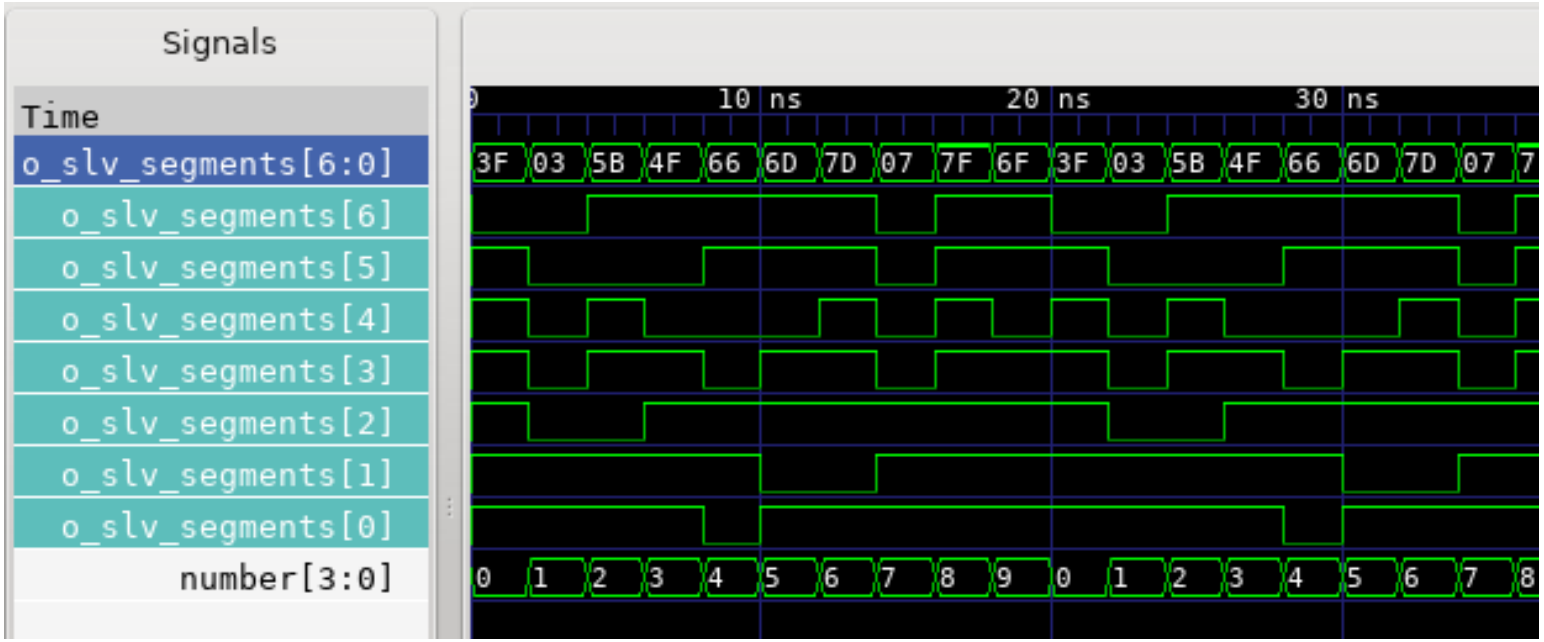
Addressdecoder



7 segment

```
entity segment7 is
    port(
        i_slv_number   :  in  std_logic_vector( 3 downto 0);  -- number input   0 - 9
        o_slv_segments :  out std_logic_vector( 6 downto 0)   -- segment output G - A
    );
end segment7;

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

architecture behavior of segment7 is
begin
    process(i_slv_number)
    begin
        segLUT: case i_slv_number is       --"GFEDCBA"
            when "0000" => o_slv_segments <= "0111111"; -- 0
            when "0001" => o_slv_segments <= "0000011"; -- 1
            when "0010" => o_slv_segments <= "1011011"; -- 2
            when "0011" => o_slv_segments <= "1001111"; -- 3
            when "0100" => o_slv_segments <= "1100110"; -- 4
            when "0101" => o_slv_segments <= "1101101"; -- 5
            when "0110" => o_slv_segments <= "1111101"; -- 6
            when "0111" => o_slv_segments <= "0000111"; -- 7
            when "1000" => o_slv_segments <= "1111111"; -- 8
            when "1001" => o_slv_segments <= "1101111"; -- 9
            when others =>
                         o_slv_segments <= "1000000";

        end case segLUT;
    end process;
end behavior;
```

7 segment


Signals
Time
o_slv_segments[6:0]
o_slv_segments[6]
o_slv_segments[5]
o_slv_segments[4]
o_slv_segments[3]
o_slv_segments[2]
o_slv_segments[1]
o_slv_segments[0]
number[3:0]
10 ns
20 ns
30 ns
3F 03 5B 4F 66 6D 7D 07 7F 6F 3F 03 5B 4F 66 6D 7D 07
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

How many bits do we need for a certain value

Some calculation

- Example for 999, we have to find the closest N for 2^N > 999

$$
\begin{aligned}
2^N &= 999 \\
\log_2 2^N &= \log_2 999 \\
&= \log_2 999 \\
&= \log(999)/\log(2) \\
&= 9.96 \\
N &= 10
\end{aligned}
$$

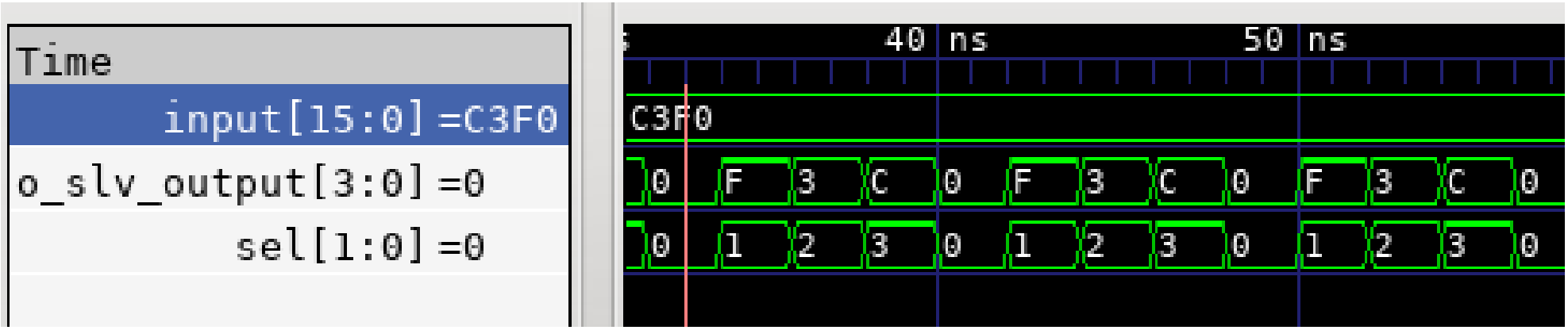
Multiplexer

```
entity mux is
    GENERIC (
        OUTPUT_BITS     : integer := 4;  -- how many bits does each output have
        SLOTS           : integer := 2   -- giving how many slots we would like to mux
    );
    port(
        i_slv_select  :   in std_logic_vector( integer(CEIL(LOG2(real(natural(SLOTS))))) - 1  downto 0);
        i_slv_input   :   in std_logic_vector( (SLOTS * OUTPUT_BITS)-1 downto 0);
        o_slv_output  :  out std_logic_vector( OUTPUT_BITS - 1 downto 0)
    );
end mux;

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

architecture behavior of mux is
begin
    process(i_slv_input, i_slv_select)
    begin
        if to_integer(unsigned(i_slv_select)) < SLOTS then
            o_slv_output <= i_slv_input( (to_integer(unsigned(i_slv_select)) + 1) * OUTPUT_BITS -1  downto to_integer(unsigned(i_slv_select)) * OUTPUT_BITS );
        else
            o_slv_output <= (others => 'X');
        end if;
    end process;
end behavior;
```

just some implementation from my side
We can also do more fancy calculations in VHDL

It is also fine to use:
std_logic_vector(7 downto 0)

Multiplexer



A bit more on the real calculation

Doing complex calculations in VHDL

- In simulation you might want to do some more complex calculations, where you need floating numbers
  - Simulate some analog effects etc.

- There is a solution in VHDL (which cannot been implemented on the FPGA)
  - New datatype REAL – it can store floating point numbers

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

use ieee.MATH_REAL.all;

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

entity mux is
    GENERIC (
        OUTPUT_BITS     : integer := 4;  -- how many bits does each output have
        SLOTS           : integer := 2   -- giving how many slots we would like to mux
    );
    port(
        i_slv_select  :   in std_logic_vector( integer(CEIL(LOG2(real(natural(SLOTS))))) - 1  downto 0);
        i_slv_input   :   in std_logic_vector( (SLOTS * OUTPUT_BITS)-1 downto 0);
        o_slv_output  :  out std_logic_vector( OUTPUT_BITS - 1 downto 0)
    );
end mux;
```

  - NATURAL is a subset of integer 0 – maximal integer

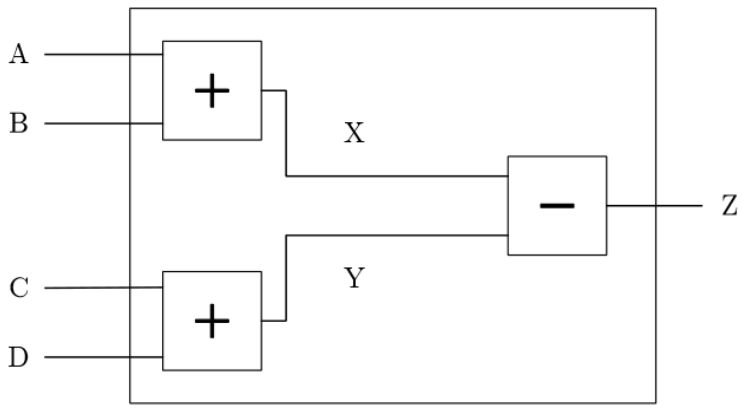

NEW

Difference FPGA / Microcontroller

Difference between Mikrocontroller and FPGA

Pros and Cons

FPGA	Microcontroller
Parallel „calculation“	**Sequential** calculation
Limited amount of logic blocks	Can handle more complex tasks (but slower)
Example: X = (A+B) , Y = (C+D), Z = (X – Y)	
2 Steps	12 Steps




```
load A
load B
add
store X
load C
load D
add
store Y
load X
load Y
sub
store Z
```

VHDL

III. Physikalisches Institut B
RWTH AACHEN UNIVERSITY

For loop

addition

- can also run downwards

```
for I in 0 to 3 loop
    if (A = I) then
        Z(I) <= '1';
    end if;
end loop;
```

```
for I in 3 downto 0 loop
    if (A = I) then
        Z(I) <= '1';
    end if;
end loop;
```

Keep in mind, we can use for loops to **describe** our algorithms.

Variables

- Until now, we know
  - input / output
  - Signals

  - They all store information over more than one clock cycle

- Sometimes, you need to store information just in order to make your algorithm be more understandable, or look more simple

  - **variable**

- variables are defined between *process(sens_list)* and *begin*
- Assignments are done by using := (not <= as for signals and in / out)

report

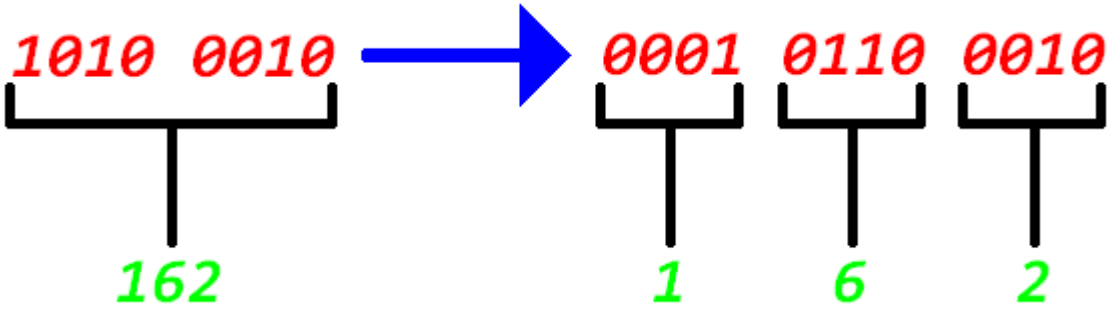
- Sometimes, e.g. if you simulate you want to output some information to the console
- Useable to detect cases, which are quite rare or when you have to get some information, which you cannot show in the simulation output.
  - calculations of constants etc.
- This is not synthesable and hence does not work on the FPGA.

```
report "ones " & integer'image( to_integer(ones) );
```

Exercises

BCD

- Example:



We would like to have this for 4 digits → Up to 9999

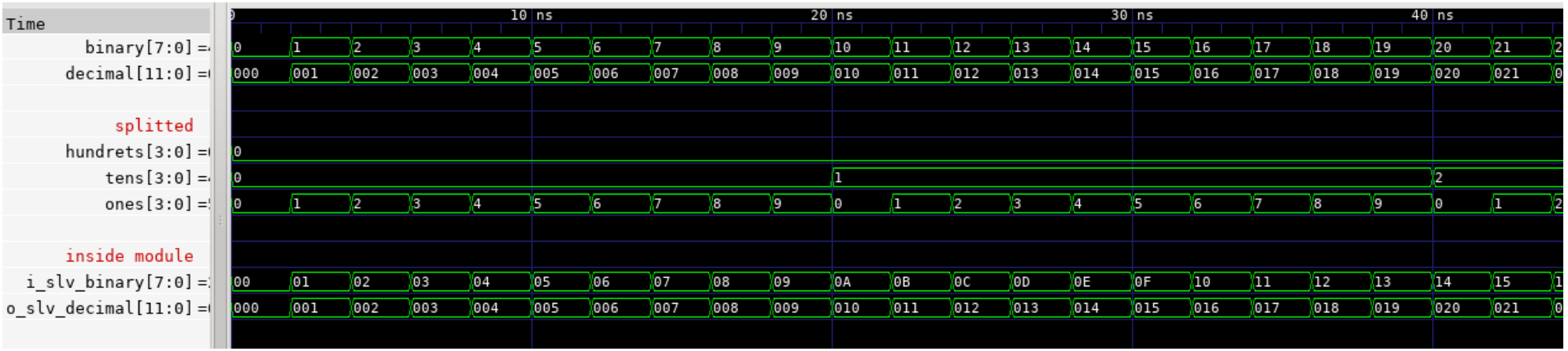
BCD Algorithm

1. If any column (100's, 10's, 1's) is 5 or greater add 3 to that column
2. Shift all #'s to the left 1 position
3. If **8** shifts are done, it's finished.
   Evaluate each column for the BCD values
4. Go to step 1.

BCD - Example

100's	10's	1's	Binary	Operation	
			1010 0010		162
		1	010 0010	<< #1	
		10	10 0010	<< #2	
		101	0 0010	<< #3	
		1000	0 0010	add 3	1 cycle
	1	0000	0010	<< #4	
	10	0000	010	<< #5	
	100	0000	10	<< #6	
	1000	0001	0	<< #7	
	1011	0001	0	add 3	1 cycle
1	0110	0010		<< #8	
1	6	2			

How does the BCD look behave



III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY