FPGA tutorial

Lecture 13

03.02.2021

Jochen Steinmann

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

Organisatorical

Next week …

… repetition

- Giving us some time to discuss some topics again
- If there is any topic, which you would like to be included in the repetition, please let me know.


Solution

ToDo Lecture 12

A. Implement the intersections FSM

B. Continue implementation of last weeks Fibonacci FSM

Fibonacci FSM

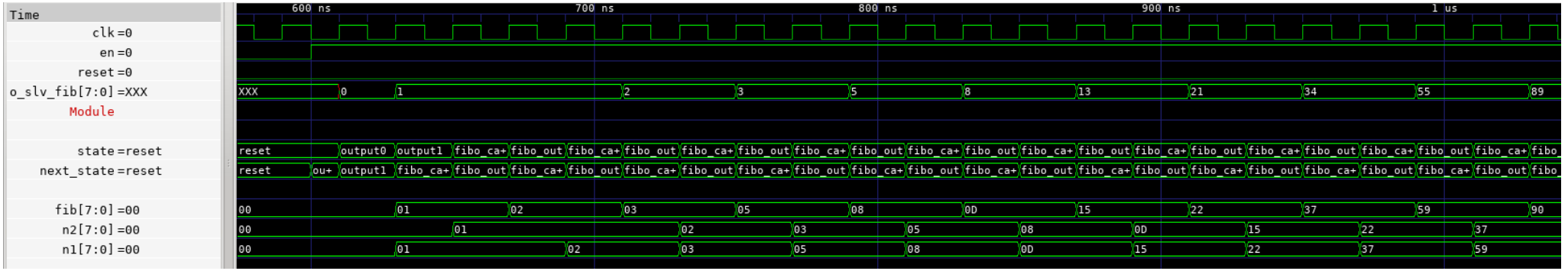
Module

```
architecture behavior of FSM_03 is

    type state_type is (
                        RESET,      -- reset state
                        OUTPUT0,    -- output 0
                        OUTPUT1,    -- output 1
                        FIBO_CALC,  -- calculate fibo
                        FIBO_OUT    -- output fibo
                        );

    signal state, next_state : state_type;

    signal n1  : unsigned(7 downto 0) := (others => '0');    -- n-1
    signal n2  : unsigned(7 downto 0) := (others => '0');    -- n-2
    signal fib : unsigned(7 downto 0) := (others => '0');   -- fibonacci number

    begin

    SYNC_PROC : process (i_sl_clk) begin
        if rising_edge(i_sl_clk) then
            if (i_sl_reset = '1') then
                state <= RESET;
            else
                state <= next_state;
            end if;
        end if;
    end process;

    OUTPUT_DECODE : process (state) begin
        --  o_slv_fibo <= (others => '0');
        case (state) is
            when OUTPUT0 =>
                o_slv_fibo <= (others => '0');
            when OUTPUT1 =>
                o_slv_fibo <= x"01";
            when FIBO_OUT =>
                o_slv_fibo <= std_logic_vector(fib);
            when others =>
                -- o_slv_fibo <= (others => '0');
        end case;
    end process;
```

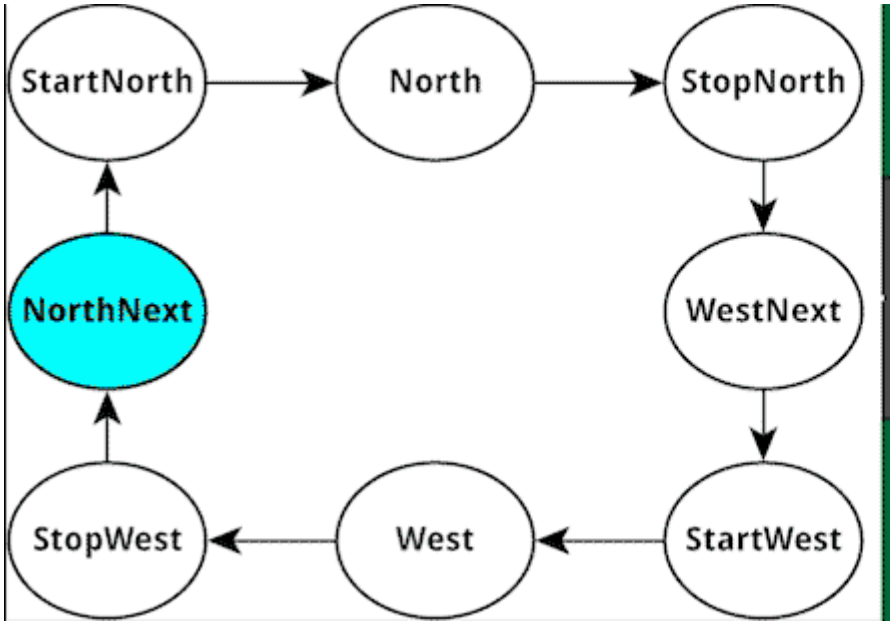
```
NEXT_STATE_DECODE : process (state, i_sl_en) begin
    next_state <= RESET;
    case (state) is
        when RESET =>
            if (i_sl_en = '1') then
                next_state <= OUTPUT0;
            end if;
        when OUTPUT0 =>
            if (i_sl_en = '1') then
                next_state <= OUTPUT1;
            end if;
        when OUTPUT1 =>
            if (i_sl_en = '1') then
                next_state <= FIBO_CALC;
            end if;
            n2 <= x"00";
            n1 <= x"01";
            fib <= x"01";
        when FIBO_CALC =>
            if (i_sl_en = '1') then
                next_state <= FIBO_OUT;
            end if;
            n1  <= fib;
            n2  <= n1;
        when FIBO_OUT =>
            fib <= n1+n2;
            if (i_sl_en = '1') then
                next_state <= FIBO_CALC;
            end if;
        when others =>
            next_state <= RESET;
    end case;
end process;
```

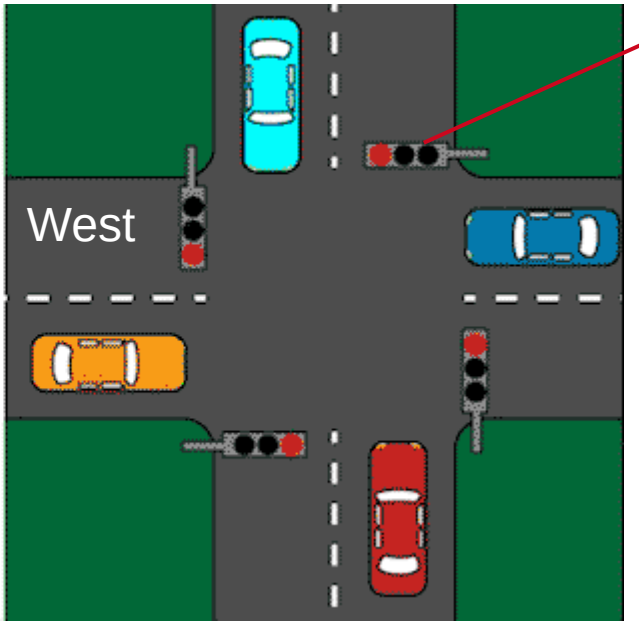
Fibonacci FSM

Module



A Intersection FSM





Traffic lights:
- red
- red – orange
- green
- orange
- red

Intersection:

All red – one directions gets green – All red – next direction gets green.

Intersection FSM

Module

```
NEXT_STATE_DECODE : process (Clk) begin

    if rising_edge(Clk) then
        if nRst = '0' then
            -- Reset values
            Counter <= 0;
        else
            Counter <= Counter + 1;

            case State is
                when NorthNext =>
                    -- If 5 seconds have passed
                    if Counter = ClockFrequencyHz * 5 -1 then
                        Counter <= 0;
                        NextState   <= StartNorth;
                    end if;

                -- Red and yellow in north/south direction
                when StartNorth =>
                    -- If 5 seconds have passed
                    if Counter = ClockFrequencyHz * 5 -1 then
                        Counter <= 0;
                        NextState   <= North;
                    end if;

                -- Green in north/south direction
                when North =>
                    -- If 1 minute has passed
                    if Counter = ClockFrequencyHz * 60 -1 the
                        Counter <= 0;
                        NextState   <= StopNorth;
                    end if;

                -- Yellow in north/south direction
                when StopNorth =>
                    -- If 5 seconds have passed
                    if Counter = ClockFrequencyHz * 5 -1 then
                        Counter <= 0;
                        NextState   <= WestNext;
                    end if;

                -- Red in all directions
                when WestNext =>
                    -- If 5 seconds have passed
                    if Counter = ClockFrequencyHz * 5 -1 then
                        Counter <= 0;
                        NextState   <= StartWest;
                    end if;
```

```
OUTPUT_DECODE : process (state) begin
    -- switch all off
    NorthRed    <= '0';
    NorthYellow <= '0';
    NorthGreen  <= '0';
    WestRed     <= '0';
    WestYellow  <= '0';
    WestGreen   <= '0';

    case (state) is
        -- Red in all directions
        when NorthNext =>
            NorthRed <= '1';
            WestRed  <= '1';
        -- Red and yellow in north/south direction
        when StartNorth =>
            NorthRed    <= '1';
            NorthYellow <= '1';
            WestRed     <= '1';
        -- Green in north/south direction
        when North =>
            NorthGreen <= '1';
            WestRed    <= '1';
        -- Yellow in north/south direction
        when StopNorth =>
            NorthYellow <= '1';
            WestRed     <= '1';

        -- Red in all directions
        when WestNext =>
            NorthRed <= '1';
            WestRed  <= '1';
        -- Red and yellow in west/east direction
        when StartWest =>
            NorthRed    <= '1';
            WestRed     <= '1';
            WestYellow <= '1';
        -- Green in west/east direction
        when West =>
            NorthRed  <= '1';
            WestGreen <= '1';
        -- Yellow in west/east direction
        when StopWest =>
            NorthRed    <= '1';
            WestYellow <= '1';
    end case;
end process;
```

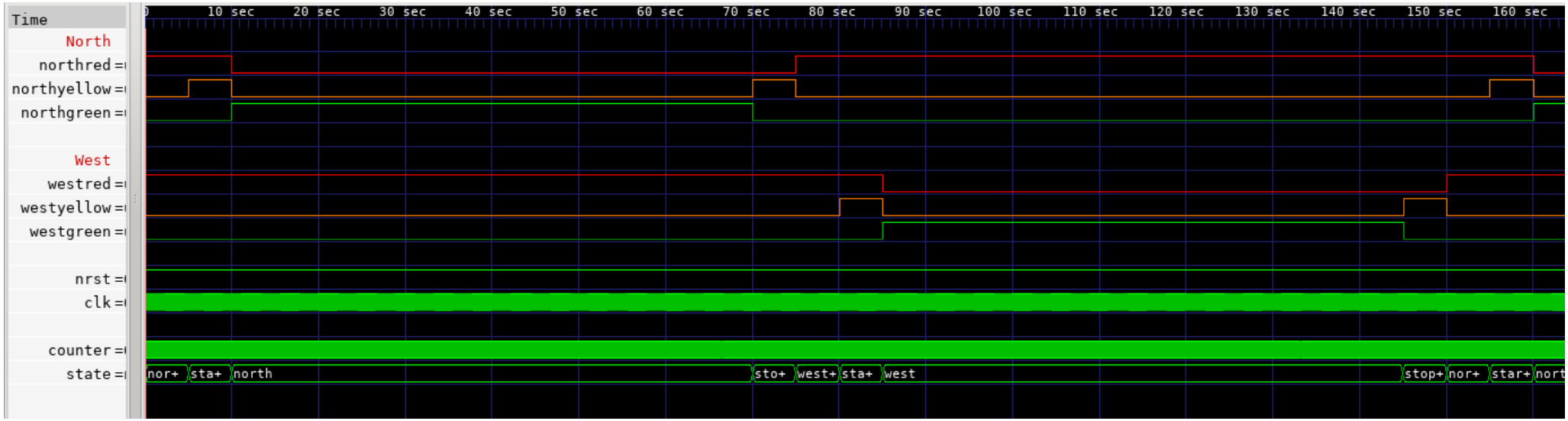
Intersection FSM

Testbench

```
architecture simulation of Traffic_Lights_Tb is

    -- We are using a low clock frequency to speed up the simulation
    constant ClockPeriod : time := 1000 ms / 100;

    signal Clk         : std_logic := '1';
    signal nRst        : std_logic := '0';

    -- traffic lights
    -- north south
    signal NorthRed    : std_logic;
    signal NorthYellow : std_logic;
    signal NorthGreen  : std_logic;

    -- west east
    signal WestRed     : std_logic;
    signal WestYellow  : std_logic;
    signal WestGreen   : std_logic;

begin

    -- The Device Under Test (DUT)
    i_TrafficLights : entity work.Traffic_Lights(behavior)
    port map (
        Clk         => Clk,
        nRst        => nRst,
        NorthRed    => NorthRed,
        NorthYellow => NorthYellow,
        NorthGreen  => NorthGreen,
        WestRed     => WestRed,
        WestYellow  => WestYellow,
        WestGreen   => WestGreen);

    -- Process for generating clock
    process begin
        Clk <= not Clk;
        wait for ClockPeriod / 2;
    end process;

    -- Testbench sequence
    process is
    begin
        wait until rising_edge(Clk);    -- wait for rising edge
        wait until rising_edge(Clk);    --

        -- Take the DUT out of reset
        nRst <= '1';

        wait;   -- wait forever
    end process;

end architecture;
```

Intersection FSM

Simulation output



Jochen Steinmann | RWTH Aachen University

An additional comment

Using integer

- You might remember the lecture, when we calculated the size, we need to store a certain bit size.

- In case, we need to cover a certain range, we can also use integer:
  variable x: integer range 0 to 100;

- **IMPORTANT**: if the range of the integer is not given, all calculations are done using 32bit!

It is also possible to use unsigned!


NEW

VHDL

III. Physikalisches Institut B
RWTH AACHEN UNIVERSITY

Assignments in VHDL

A more detailed view

Process „runs“, when a,b,c changes!

```
process(a, b, c)
begin
        y <= '0';
        if a = '1' or b = '1' then y <= '1'; end if;
        if c = '1' then y <= '0'; end if;
end process;
```

After an event in a, b or c:

1. new value '0' is scheduled for y, but y still has its old value
2. if a or b are '1', the new scheduled value of y is changed to '1'
3. if c is '1', the scheduled value of y is changed to '0'
4. at the end of the process the scheduled value is assigned to y
5. as y is not on the sensitivity list, the process is suspended until the next event on a, b or c, and y preserves its new value

All this happens within the same simulation time!

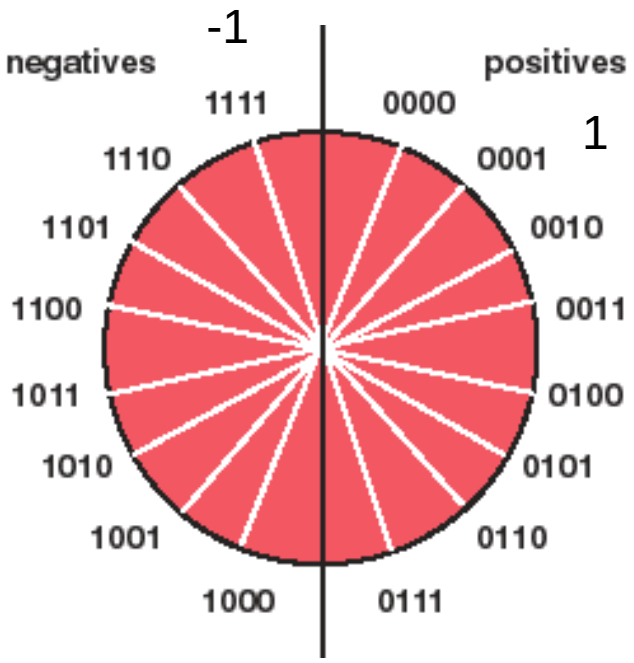
Positive / Negative

signed – unsigned

Going to negative values …

- Up to now, we are using unsigned or std_logic_vector, which is the same!

- How are negative binary numbers represented?



Binary	Hex	Decimal	
		US	S
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	-8
1001	9	9	-7
1010	A	10	-6
1011	B	11	-5
1100	C	12	-4
1101	D	13	-3
1110	E	14	-2
1111	F	15	-1

How to convert between signed / unsigned?

Two‘s compliment

- **How to make the two‘s compliment**
  - Invert all bits
  - Add +1

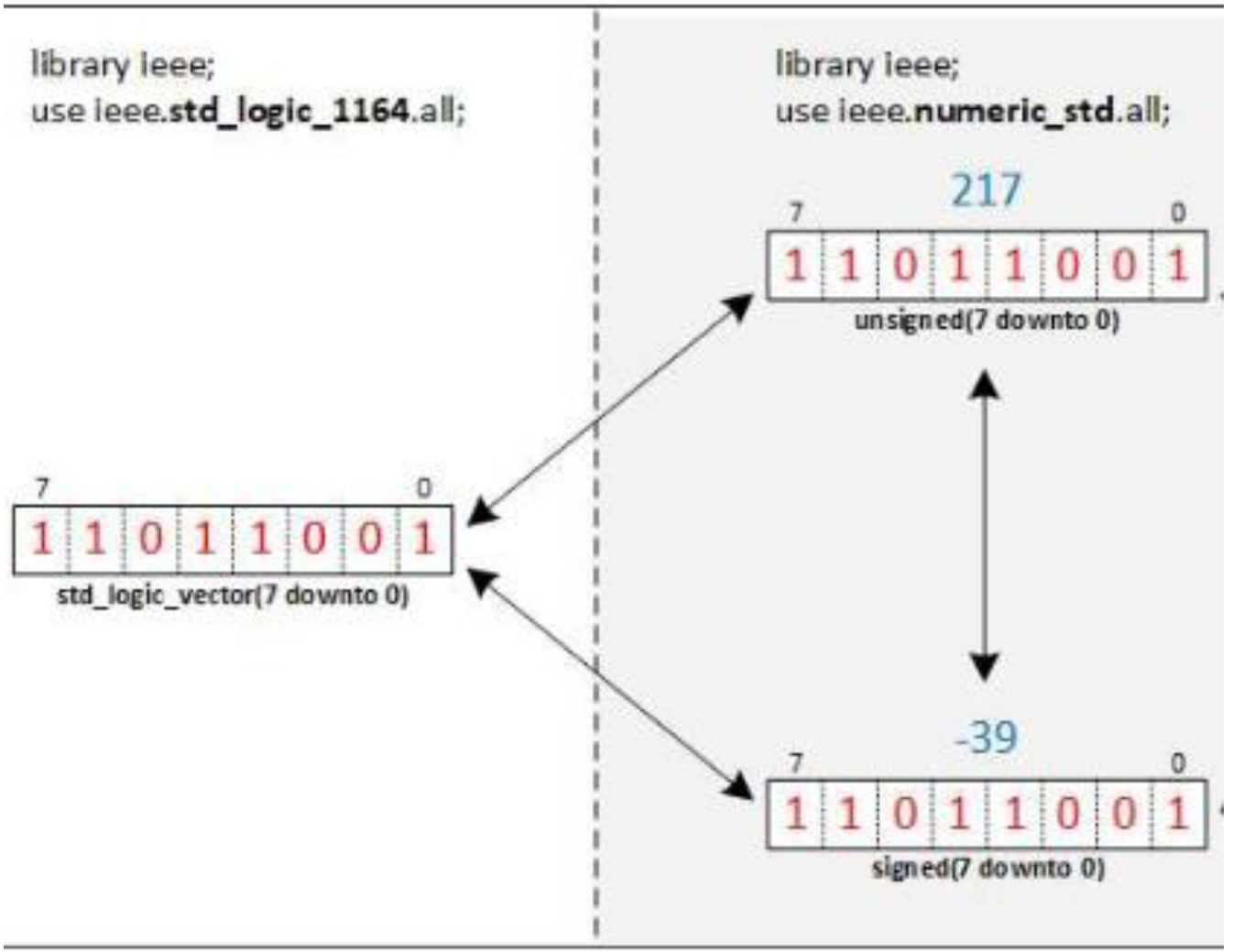
- Convert signed N-bit into decimal

  - **1** 1 1 1 1 1 0 1 twos complement => **0** 0 0 0 0 0 1 0 + 1 = 1 1 = 3
    result is **-**3

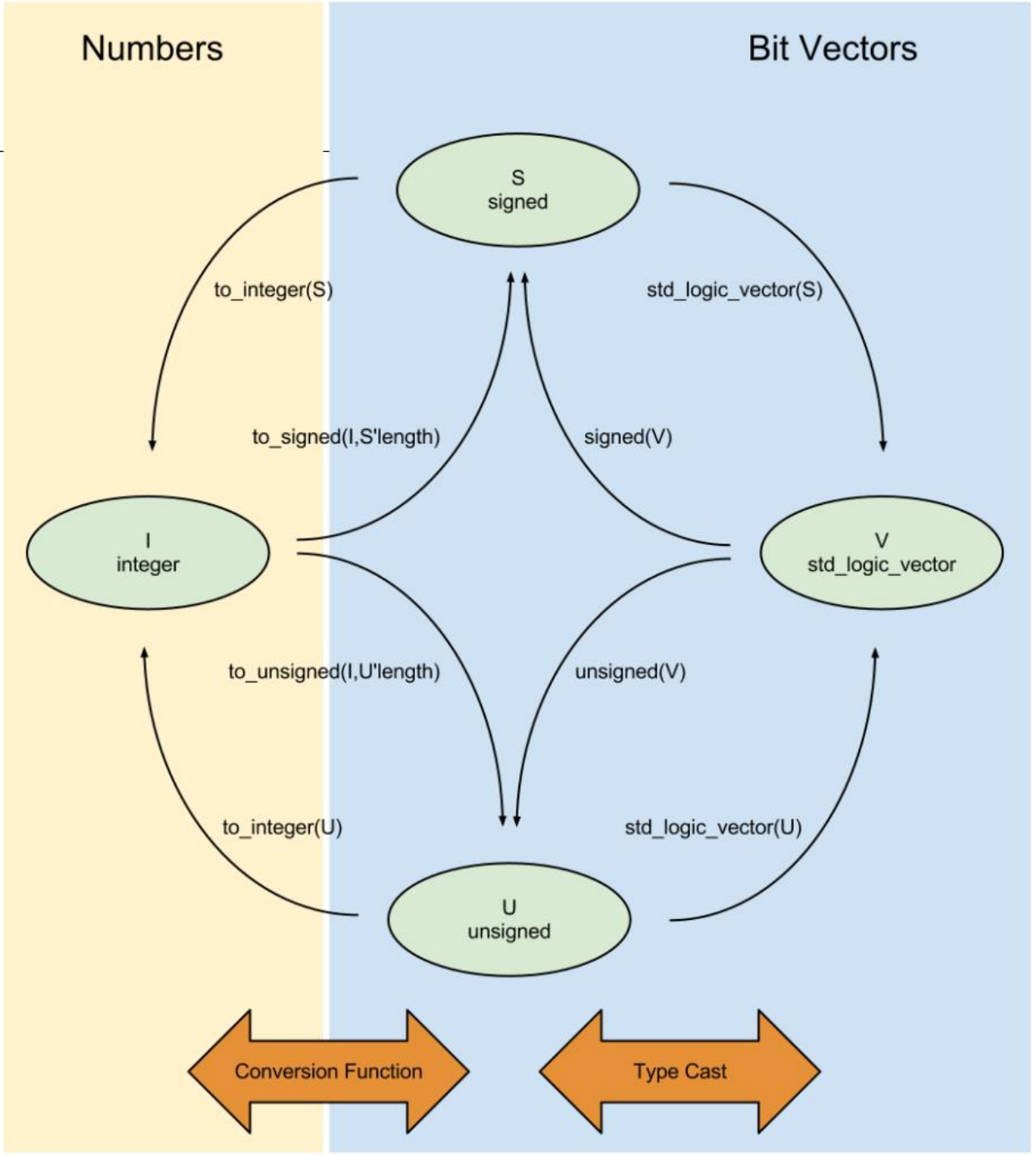
    first bit gives, whether number is positive or negative

Conversion

We can convert a std_logic_vector to both signed and unsigned



Conversion

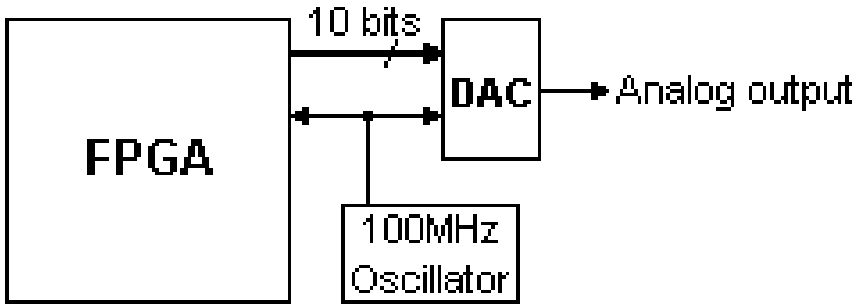

Numbers
Bit Vectors
S
signed
to_integer(S)
std_logic_vector(S)
to_signed(I,S'length)
signed(V)
I
integer
V
std_logic_vector
to_unsigned(I,U'length)
unsigned(V)
to_integer(U)
std_logic_vector(U)
U
unsigned
Conversion Function
Type Cast

DDS

Generating a sine wave

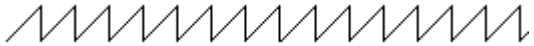
Direct Digital Synthesis

- Easy way of generating an arbitrary waveforms



- If we use a simple counter (e.g. an 8 bit counter)
  - connect bit 8 (counter(7)) to the DAC
  - of course set it to full scale, we can easily generate a square wave.

- If we use the full counter output (-> saw tooth)

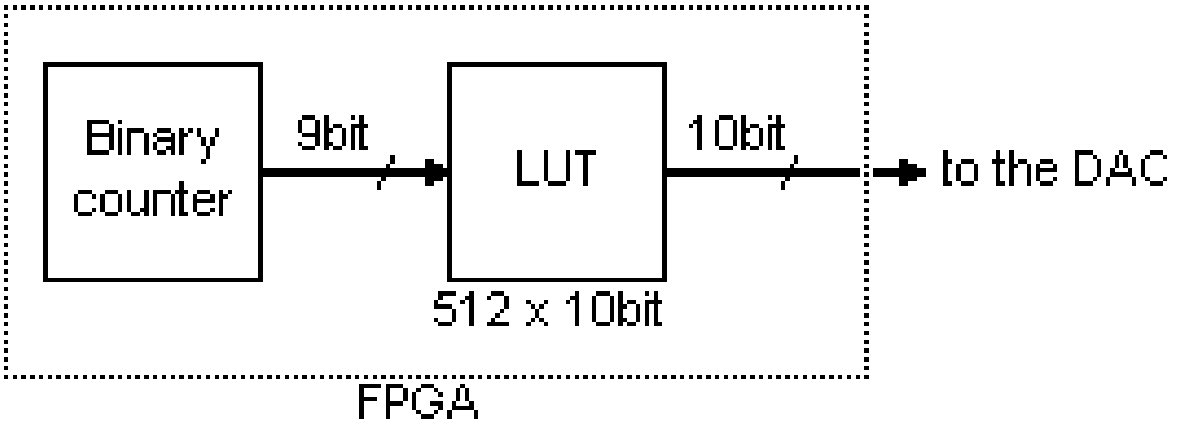


- If we use the counter output + a sign bit (triangle)

How to create a sine wave?

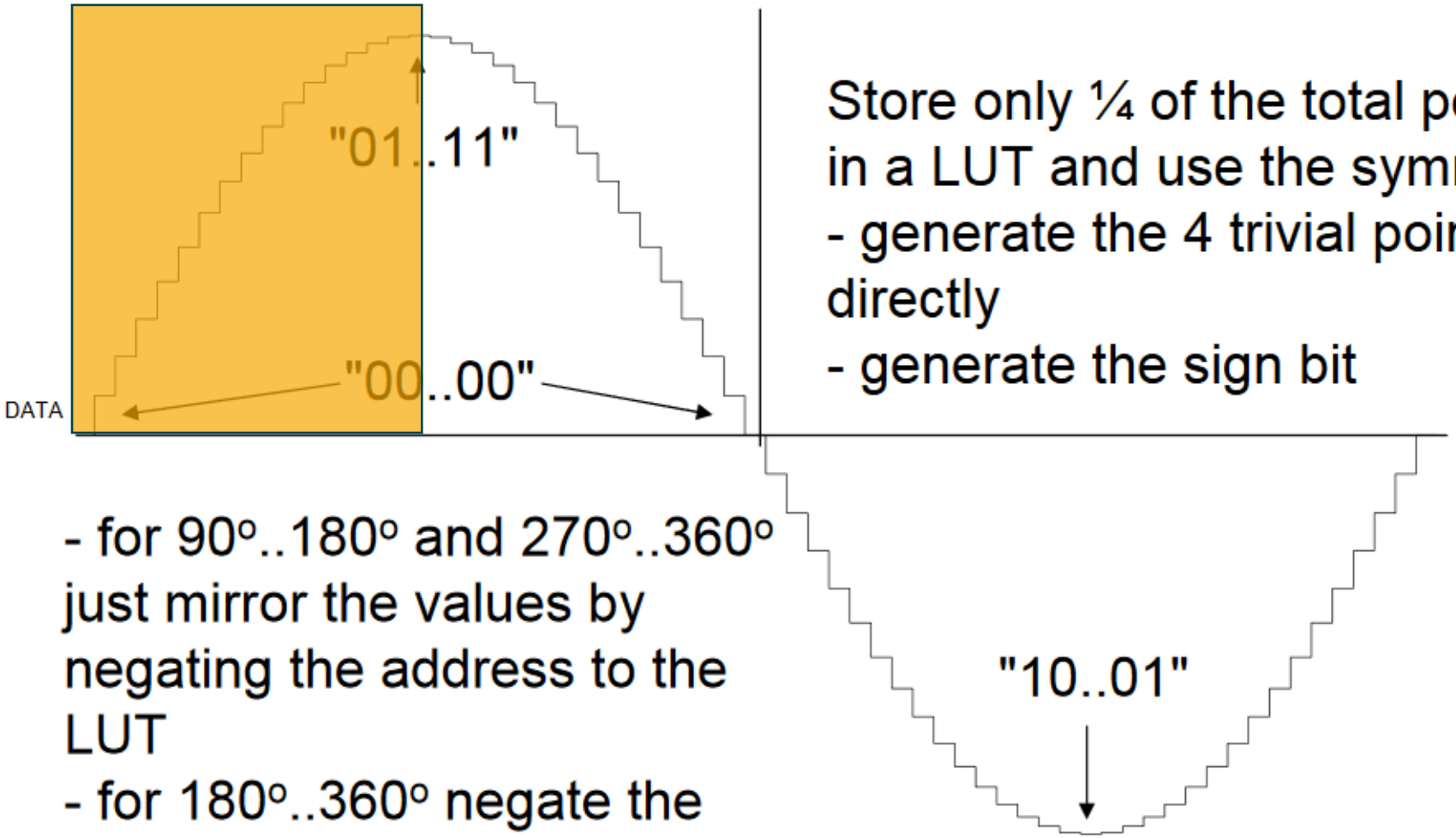
Look up table

- Any arbitrary waveform can be generated via this way



Sine wave generation

Using the symmetries



Store only ¼ of the total period in a LUT and use the symmetry:
- generate the 4 trivial points directly
- generate the sign bit

- for 90°..180° and 270°..360° just mirror the values by negating the address to the LUT
- for 180°..360° negate the output of the LUT

ToDo Lecture 13

General constrains
assume usage of a 9 bit DAC (using two‘s complement)

A. Create a tool for generating the sine wave lookup table (Python, C++ etc.)
   16 entries are sufficient (we need 8bit resolution)

B. Implement a DDS for generating a sine wave
   1. Counter for address of the LUT (you can reuse your modules – lecture 3 / lecture 6)
   2. Output from lookup table
   3. Encoding of positive / negative

C. Use your sine wave generator to provide
   1. Two different frequencies
   2. Two different phases per frequency (something different than n x 90°)
      use a phase difference of 120°

Additional Explanation

A

```
#!/bin/python

import numpy as np

lut_addr = np.arange(0,16)

print (lut_addr)

lut_content =  Insert your calculation here
lut_content = lut_content.astype(int)

print(lut_content)

for addr, content in zip(lut_addr, lut_content):
    print("when %d =>   y <= %d"%(addr, content))
```

Pitfalls

- Pay attention to the sensitivity lists

Simple 16 entry LUT

```
when 0 =>   y <= 0
when 1 =>   y <= 26
when 2 =>   y <= 53
when 3 =>   y <= 78
when 4 =>   y <= 103
when 5 =>   y <= 127
when 6 =>   y <= 149
when 7 =>   y <= 170
when 8 =>   y <= 189
when 9 =>   y <= 206
when 10 =>   y <= 220
when 11 =>   y <= 232
when 12 =>   y <= 242
when 13 =>   y <= 249
when 14 =>   y <= 253
when 15 =>   y <= 255
```

Thank you!