

In the previous lecture you learned how different flavors of latches and flip-flops can be described in VHDL. However, while these elements enable the construction of sequential circuits within the synchronous paradigm, you have not yet seen how such a circuit can actually be described. This is what we'll do in this lecture.

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

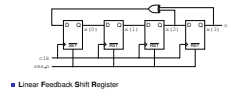
Hardware Modeling [VU] (191.011) – WS25 – Sequential Circuit Example: LFSR

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

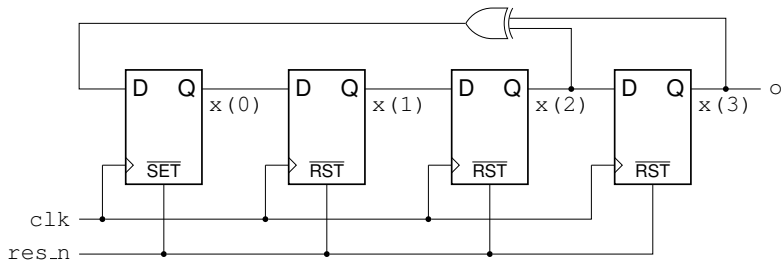
Sequential Circuit Example: LFSR

Example: 4-bit LFSR



On the slide you can find the schematic of our example circuit. In particular, this circuit is a so-called linear-feedback-shift-register, abbreviated as LFSR.

Example: 4-bit LFSR



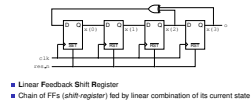
Linear Feedback Shift Register

HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench

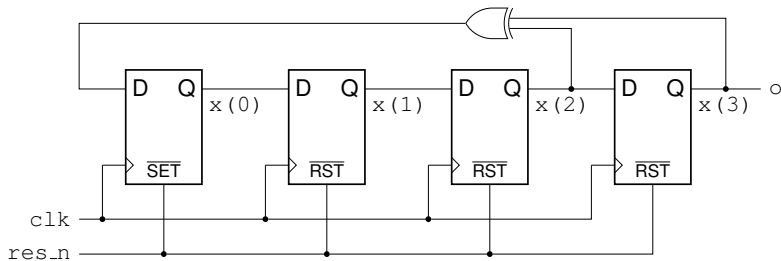
Sequential Circuit Example: LFSR

Example: 4-bit LFSR



As the name suggests, such circuits revolve around a so-called *shift register*, fed by a linear combination of its current state. A shift register itself is nothing more than a chain of flip-flops, where each flip-flop samples the output of its predecessor. This essentially shifts the currently stored bits by one flip-flop per clock period, hence the name.

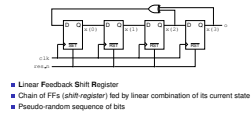
Example: 4-bit LFSR



- **Linear Feedback Shift Register**
- Chain of FFs (*shift-register*) fed by linear combination of its current state

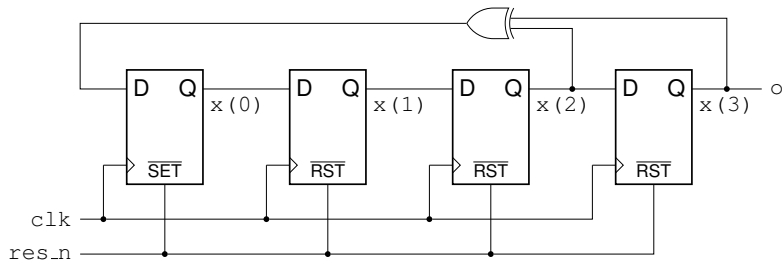
Sequential Circuit Example: LFSR

Example: 4-bit LFSR



Note that this type of circuit is not some artificial example, but actually something commonly encountered in circuits. In particular, LFSRs are often used as pseudo-random number generators. This means that they can generate a sequence of bits that looks like it is random while it actually is not. We will now discuss how the circuit operates.

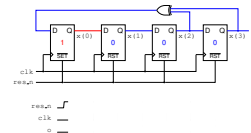
Example: 4-bit LFSR



- **Linear Feedback Shift Register**
- Chain of FFs (*shift-register*) fed by linear combination of its current state
- Pseudo-random sequence of bits

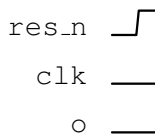
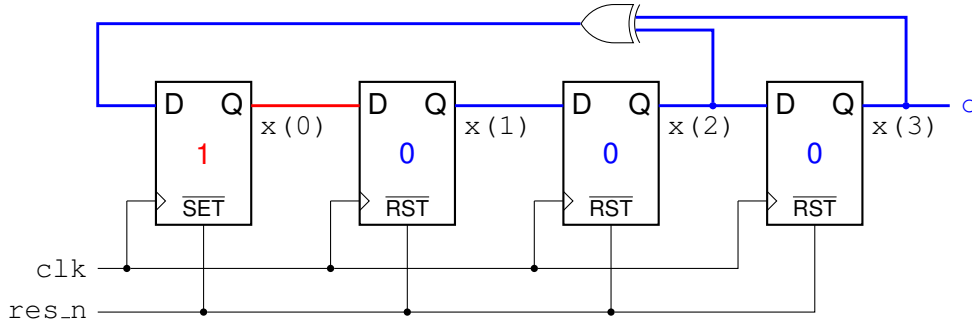
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



To demonstrate the circuit's operation, we highlight the data wires of the circuit in blue when propagating a logical 0, and in red for logical 1. We also show the logical value currently held by each flip-flop and a timing diagram of the inputs and outputs underneath the circuit.

LFSR - Circuit Operation Principle

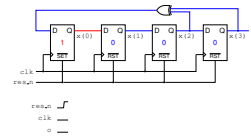


HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

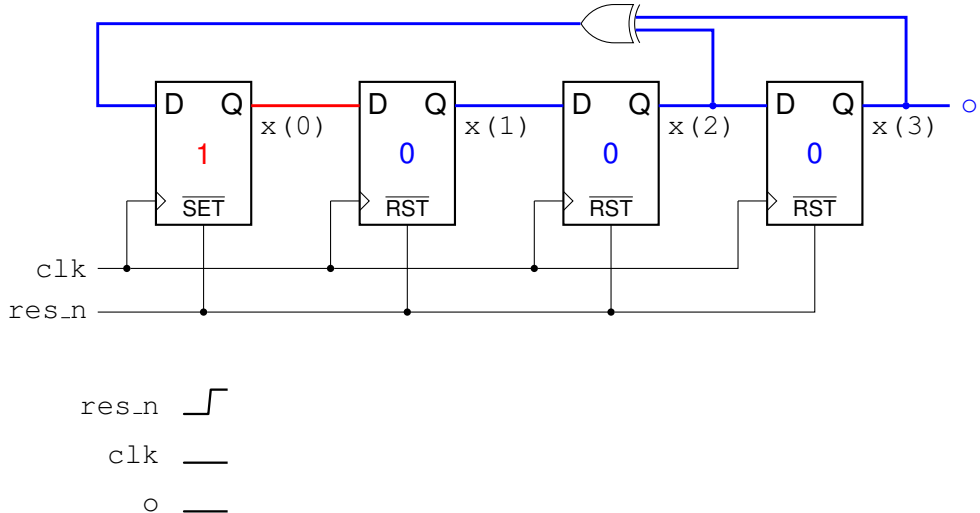
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



As you can observe in the wave diagram, the first thing that is done is to reset the circuit – just as we discussed in the previous lecture. As a result, all flip-flops, except the first one, hold a zero. This initial value is vital, as all flip-flops being zero would result in the output being constantly zero as well. And that's not very random.

LFSR - Circuit Operation Principle

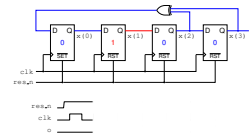


HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench

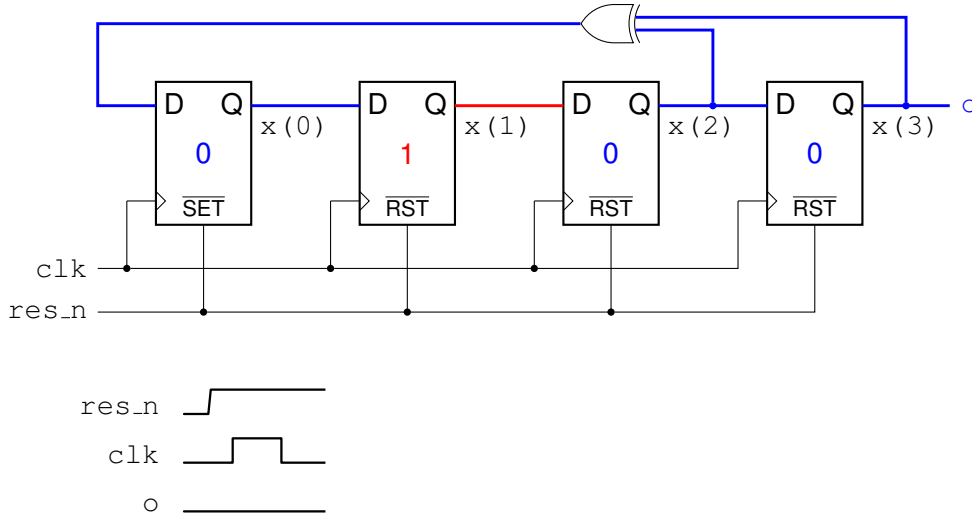
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



At the first rising clock edge, all flip-flops sample their inputs. This leads to the stored 1 being shifted to the right. The left-most flip-flop samples the output of the XOR gate, which is 0 since the two right-most flip-flops both hold a 0.

LFSR - Circuit Operation Principle

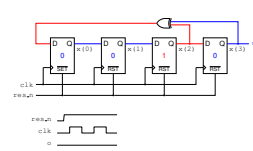


HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

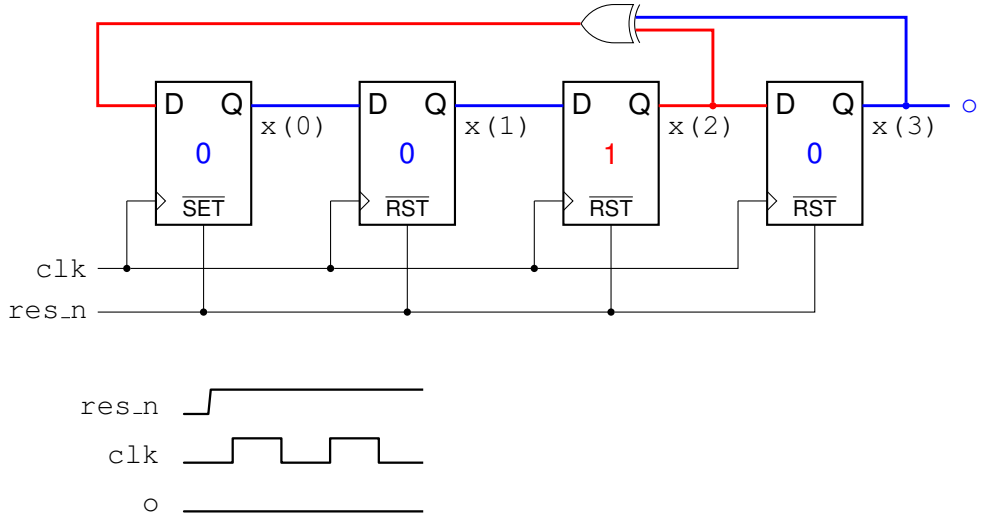
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



At the next clock edge, the current register bits are shifted to the right again. Note how this makes the feedback path high.

LFSR - Circuit Operation Principle

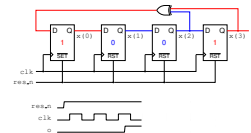


HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench

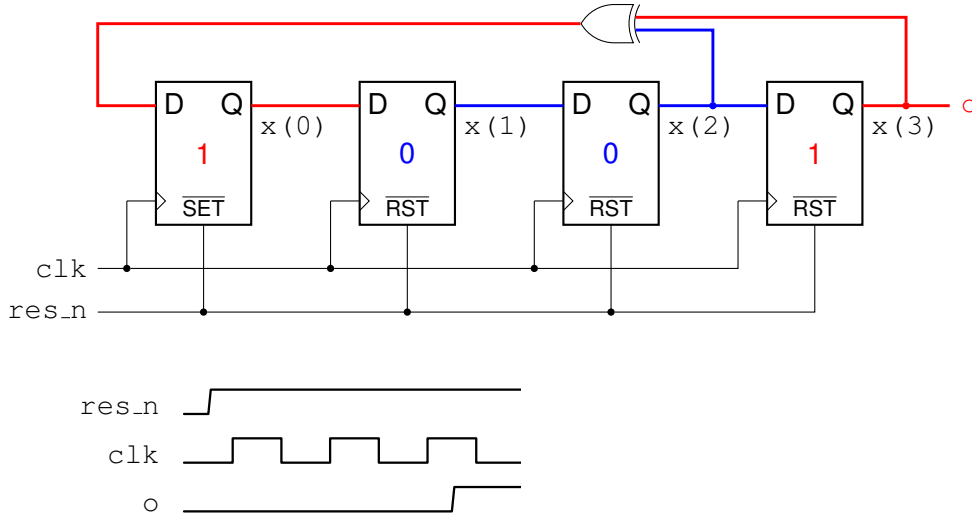
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



The first flip-flop now samples the applied 1 from the feedback path, while the already stored 1 is shifted into the last flip-flop. As a result, the output signal is asserted, as can also be observed in the wave diagram.

LFSR - Circuit Operation Principle

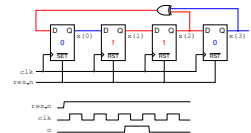


HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench

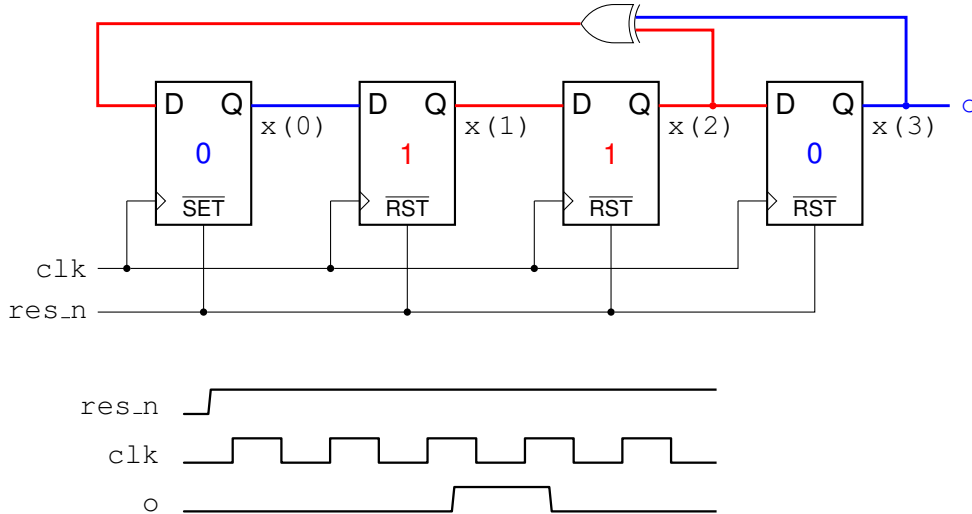
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



The LFSR continues in this fashion of sampling the feedback and shifting its internal state, producing a sequence of 0s and 1s at its output.

LFSR - Circuit Operation Principle

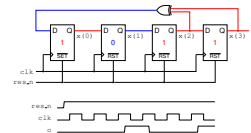


HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench

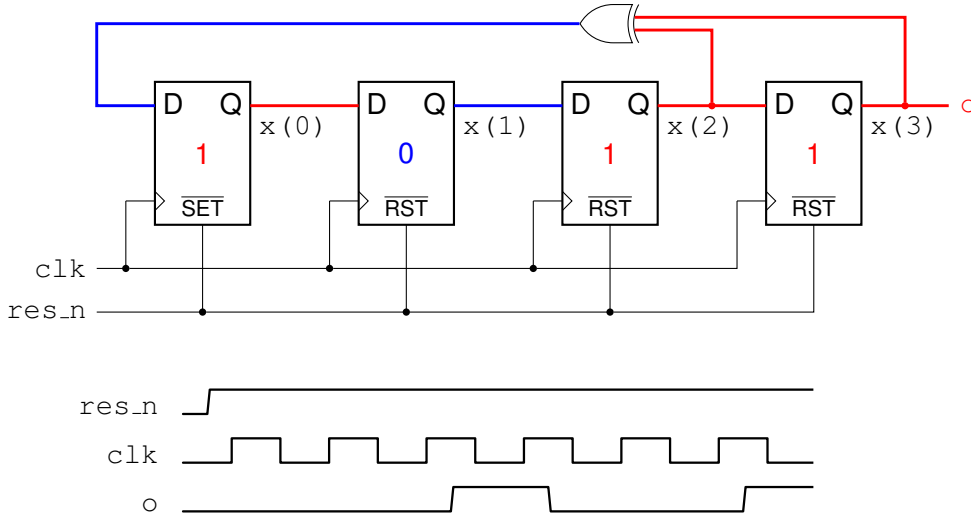
Sequential Circuit Example: LFSR

LFSR - Circuit Operation Principle



The shifting now leads to the output being set for two consecutive clock cycles. However, by now you have likely gotten the hang of it, and we can continue by discussing how such a circuit can be described in VHDL.

LFSR - Circuit Operation Principle



HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

LFSR - VHDL Design



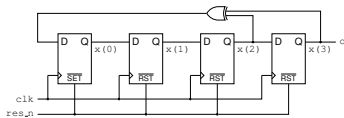
```
1 entity lfsr is
2   port (
3     clk : in  std_ulogic;
4     res_n : in  std_ulogic;
5     o : out std_ulogic
6   );
7 end entity;
```

On the slide you can find an entity declaration for the LFSR circuit. The ports should not be too surprising, containing a clock, a reset and output. Let us now look at the accompanying architecture.

LFSR - VHDL Design

HWMod
WS25

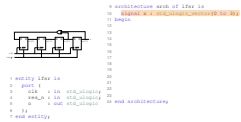
LFSR
Operation
VHDL Design
Testbench



```
1 entity lfsr is
2   port (
3     clk : in  std_ulogic;
4     res_n : in  std_ulogic;
5     o : out std_ulogic
6   );
7 end entity;
```

Sequential Circuit Example: LFSR

LFSR - VHDL Design

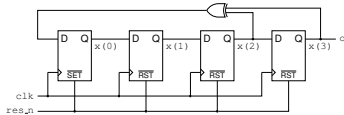


In its declarative part, the architecture declares a single vector signal for the register x .

LFSR - VHDL Design

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

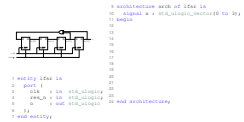


```
1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;
```

```
9 architecture arch of lfsr is
10    signal x : std_ulogic_vector(0 to 3);
11 begin
12
13
14
15
16
17
18
19
20
21
22
23
24 end architecture;
```

Sequential Circuit Example: LFSR

LFSR - VHDL Design

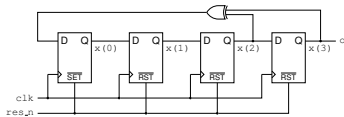


As shown in the previous lecture, to model this register, we can use a process that asynchronously resets this signal and updates it at rising clock edges.

LFSR - VHDL Design

HWMoD
WS25

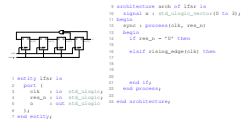
LFSR
Operation
VHDL Design
Testbench



```
1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;
```

```
9 architecture arch of lfsr is
10   signal x : std_ulogic_vector(0 to 3);
11 begin
12
13
14
15
16
17
18
19
20
21
22
23
24 end architecture;
```

LFSR - VHDL Design

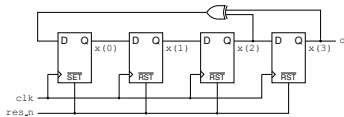


Hence, we add a respective process with the respective `if`-structure added inside the process. Note that we assume that the flip-flops are reset asynchronously.

LFSR - VHDL Design

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench



```

1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;

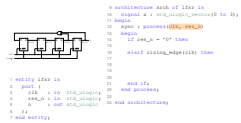
```

```

9 architecture arch of lfsr is
10  signal x : std_ulogic_vector(0 to 3);
11 begin
12  sync : process(clk, res_n)
13  begin
14    if res_n = '0' then
15
16      elsif rising_edge(clk) then
17
18
19
20
21      end if;
22  end process;
23 end architecture;

```

LFSR - VHDL Design

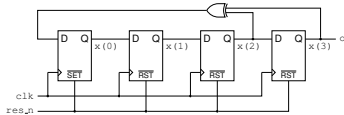


Recall that this also means that the process must be sensitive to both the reset, and the clock.

LFSR - VHDL Design

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench



```

1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;

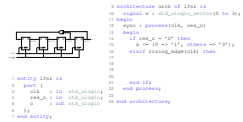
```

```

9 architecture arch of lfsr is
10  signal x : std_ulogic_vector(0 to 3);
11 begin
12  sync : process (clk, res_n)
13  begin
14    if res_n = '0' then
15
16      elsif rising_edge(clk) then
17
18
19
20
21      end if;
22  end process;
23 end architecture;

```

LFSR - VHDL Design

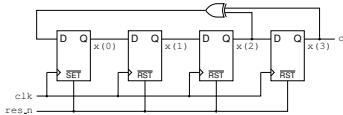


During the asynchronous reset, all bits of x , except for the one at index 0, are reset to '0' via an aggregate expression. The first bit is set to '1' to model the asynchronous set of the left-most flip-flop we mentioned before.

LFSR - VHDL Design

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench



```

1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;

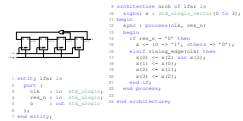
```

```

9 architecture arch of lfsr is
10  signal x : std_ulogic_vector(0 to 3);
11 begin
12  sync : process(clk, res_n)
13  begin
14    if res_n = '0' then
15      x <= (0 => '1', others => '0');
16    elsif rising_edge(clk) then
17
18
19
20
21      end if;
22    end process;
23  end architecture;

```

LFSR - VHDL Design

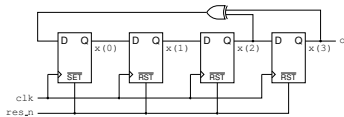


At rising clock edges, the register is updated such that it implements the introduced LFSR. In particular, this means that the distinct bits of the register are connected to form a chain of flip-flops, where each flip-flop samples the output of its predecessor. The left-most flip-flop samples the XOR of the bits 2 and 3.

LFSR - VHDL Design

HWMMod
WS25

LFSR
Operation
VHDL Design
Testbench



```

1 entity lfsr is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     o        : out std_ulogic
6   );
7 end entity;

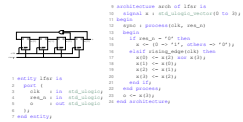
```

```

9 architecture arch of lfsr is
10  signal x : std_ulogic_vector(0 to 3);
11 begin
12  sync : process(clk, res_n)
13  begin
14    if res_n = '0' then
15      x <= (0 => '1', others => '0');
16    elsif rising_edge(clk) then
17      x(0) <= x(2) xor x(3);
18      x(1) <= x(0);
19      x(2) <= x(1);
20      x(3) <= x(2);
21    end if;
22  end process;
23 end architecture;

```

LFSR - VHDL Design

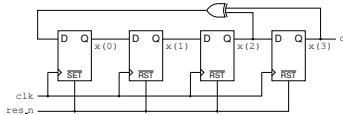


Finally, we use a concurrent signal assignment to connect the output of the right-most flip-flop to the output port o.

LFSR - VHDL Design

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench



```

1 entity lfsr is
2   port (
3     clk : in std_ulogic;
4     res_n : in std_ulogic;
5     o : out std_ulogic
6   );
7 end entity;

```

```

9 architecture arch of lfsr is
10  signal x : std_ulogic_vector(0 to 3);
11  begin
12    sync : process(clk, res_n)
13    begin
14      if res_n = '0' then
15        x <= (0 => '1', others => '0');
16      elsif rising_edge(clk) then
17        x(0) <= x(2) xor x(3);
18        x(1) <= x(0);
19        x(2) <= x(1);
20        x(3) <= x(2);
21      end if;
22    end process;
23    o <= x(3);
24 end architecture;

```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6
7
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15    );
end;
```

To conclude this example, let us discuss how a testbench can be written for the LFSR. On the slide you can already find a part of the architecture for such a testbench. It declares the signals required by the unit under test, and instantiates it.

LFSR - Testbench

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6
7
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15    );
end;
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6
7
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15    );
16 end;
```

Obviously, a synchronous design like this one requires a clock to simulate it. But where does this clock signal come from? As for all inputs of the unit-under-test, the testbench has to generate and apply it. This is what we'll address now, as you previously only had to test combinational designs.

LFSR - Testbench

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6
7
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15    );
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o,
15  );
16 end;
```

First, we declare an additional constant and a signal.

LFSR - Testbench

HWMod
WS25

LFSR
Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

The constant holds the desired clock period, in this case 10 ns. Often, the exact value of the clock period is not too important. However, in general, a design is specified for a range or a particular clock frequency. The testbench should then also adhere to this. An exception where setting the right clock period is paramount are *postlayout* simulations, but more on that at another time.

LFSR - Testbench

HWMoD
WS25

LFSR

Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

LFSR - Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o,
15    stop_clk => stop_clk
16  );
17 end;
```

The purpose of the `stop_clk` signal will become clear in a moment.

LFSR - Testbench

HWMod
WS25

LFSR
Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 clkgen : process
2 begin
3   while not stop_clk loop
4     clk <= '0';
5     wait for 0.5*CLK_PERIOD;
6     clk <= '1';
7     wait for 0.5*CLK_PERIOD;
8   end loop;
9   wait;
10 end process;
11
12 uut : entity work.lfsr
13 port map (
14   clk => clk,
15   res_n => res_n,
16   o => o
17 );
```

TO deal with the clock, we start by creating a dedicated clock generation process. The purpose of this **non-synthesizable** code is to periodically toggle the clock signal.

LFSR - Testbench

HWMoD
WS25

LFSR

Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

```
1  clkgen : process
2  begin
3    while not stop_clk loop
4      clk <= '0';
5      wait for 0.5*CLK_PERIOD;
6      clk <= '1';
7      wait for 0.5*CLK_PERIOD;
8    end loop;
9    wait;
10 end process;
```

```

1 clkgen < process
2 begin
3   while not stop_clk loop
4     clk <= '0';
5     wait for 0.5*CLK_PERIOD;
6     clk <= '1';
7     wait for 0.5*CLK_PERIOD;
8   end loop;
9 end process;
10
11 uut < work.lfsr;
12 clk <= clk;
13 res_n <= res_n;
14 o <= o;
15
16

```

The desired clock period is achieved by waiting for half the respective constant after each of the two assignments.

LFSR - Testbench

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

```

1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );

```

```

1  clkgen : process
2  begin
3    while not stop_clk loop
4      clk <= '0';
5      wait for 0.5*CLK_PERIOD;
6      clk <= '1';
7      wait for 0.5*CLK_PERIOD;
8    end loop;
9    wait;
10 end process;

```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 clkgen : process
2 begin
3     wait for 0.5*CLK_PERIOD;
4     clk <= '0';
5     wait for 0.5*CLK_PERIOD;
6     clk <= '1';
7     wait for 0.5*CLK_PERIOD;
8     clk <= '0';
9     wait for 0.5*CLK_PERIOD;
10    clk <= '1';
11 end process;
```

However, if we want the simulation to terminate automatically, we must ensure that all signals become stable eventually. Naturally, this includes the clock signal. Therefore, we wrap the assignments to the clock signal in a while-loop that runs until the `stop_clk` signal becomes `true`.

LFSR - Testbench

HWMoD
WS25

LFSR
Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2     signal clk : std_ulogic;
3     signal res_n : std_ulogic;
4     signal o : std_ulogic;
5
6     constant CLK_PERIOD : time := 10 ns;
7     signal stop_clk : boolean := false;
8 begin
9
10    uut : entity work.lfsr
11    port map (
12        clk => clk,
13        res_n => res_n,
14        o => o
15    );
```

```
1     clkgen : process
2     begin
3         while not stop_clk loop
4             clk <= '0';
5             wait for 0.5*CLK_PERIOD;
6             clk <= '1';
7             wait for 0.5*CLK_PERIOD;
8         end loop;
9         wait;
10    end process;
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 clkgen : process
2 begin
3   while not stop_clk loop
4     clk <= '0';
5     wait for 0.5*CLK_PERIOD;
6     clk <= '1';
7     wait for 0.5*CLK_PERIOD;
8   end loop;
9   wait;
10 end process;
```

After that, the clock remains stable, and the clock generation process will `wait` indefinitely. Note that the stop signal will be set by the stimulus process once it is done with testing the unit-under-test. We will now look at this process.

LFSR - Testbench

HWMoD
WS25

LFSR

Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11  port map (
12    clk => clk,
13    res_n => res_n,
14    o => o
15  );
```

```
1  clkgen : process
2  begin
3    while not stop_clk loop
4      clk <= '0';
5      wait for 0.5*CLK_PERIOD;
6      clk <= '1';
7      wait for 0.5*CLK_PERIOD;
8    end loop;
9    wait;
10 end process;
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 clkgen : process
2 begin
3   architecture bench of lfsr_tb is
4     signal clk : std_ulogic;
5     signal res_n : std_ulogic;
6     signal o : std_ulogic;
7     constant CLK_PERIOD : time := 10 ns;
8     signal stop_clk : boolean := false;
9   begin
10    uut : entity work.lfsr
11      port map (
12        clk => clk,
13        res_n => res_n,
14        o => o
15      );
16
17   end architecture bench;
18
19   stimulate : process
20   begin
21     res_n <= '0';
22     wait until rising_edge(clk);
23     wait until rising_edge(clk);
24     res_n <= '1';
25     wait for 6*CLK_PERIOD;
26     stop_clk <= true;
27     wait;
28   end process;
29
30   stop_clk : process
31   begin
32     wait for 0.5*CLK_PERIOD;
33     clk <= '1';
34     wait for 0.5*CLK_PERIOD;
35     clk <= '0';
36     wait;
37   end process;
38
39 end clkgen;
```

The first task of the stimulus process is to reset the UUT. As already mentioned, otherwise the design might be in an arbitrary state which prohibits proper testing. Note that the reset should be applied for some time, ideally more than a clock cycle, to ensure that everything is properly reset. In this case we activate the reset until two rising clock edges were observed.

LFSR - Testbench

HWMoD
WS25

LFSR

Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15    );
```

```
1   clkgen : process
2   begin
3     while not stop_clk loop
4       clk <= '0';
5       wait for 0.5*CLK_PERIOD;
6       clk <= '1';
7       wait for 0.5*CLK_PERIOD;
8     end loop;
9     wait;
10  end process;
11
12  stimulate : process
13  begin
14    res_n <= '0';
15    wait until rising_edge(clk);
16    wait until rising_edge(clk);
17    res_n <= '1';
18    wait for 6*CLK_PERIOD;
19    stop_clk <= true;
20    wait;
21  end process;
```

Sequential Circuit Example: LFSR

LFSR - Testbench

```
1 clkgen : process
2 begin
3   architecture bench of lfsr_tb is
4     signal clk : std_ulogic;
5     signal res_n : std_ulogic;
6     signal o : std_ulogic;
7     constant CLK_PERIOD : time := 10 ns;
8     signal stop_clk : boolean := false;
9   begin
10    uut : entity work.lfsr
11      port map (
12        clk => clk,
13        res_n => res_n,
14        o => o
15      );
16
17    stim : process
18      res_n <= '0';
19      wait until rising_edge(clk);
20      res_n <= '1';
21      wait for 6*CLK_PERIOD;
22      stop_clk <= true;
23      wait;
24    end process;
25
26    stim2 : process
27      res_n <= '0';
28      wait until rising_edge(clk);
29      res_n <= '1';
30      wait for 6*CLK_PERIOD;
31      stop_clk <= true;
32      wait;
33    end process;
34
35    stim3 : process
36      res_n <= '0';
37      wait until rising_edge(clk);
38      res_n <= '1';
39      wait for 6*CLK_PERIOD;
40      stop_clk <= true;
41      wait;
42    end process;
43
44    stim4 : process
45      res_n <= '0';
46      wait until rising_edge(clk);
47      res_n <= '1';
48      wait for 6*CLK_PERIOD;
49      stop_clk <= true;
50      wait;
51    end process;
52
53    stim5 : process
54      res_n <= '0';
55      wait until rising_edge(clk);
56      res_n <= '1';
57      wait for 6*CLK_PERIOD;
58      stop_clk <= true;
59      wait;
60    end process;
61
62    stim6 : process
63      res_n <= '0';
64      wait until rising_edge(clk);
65      res_n <= '1';
66      wait for 6*CLK_PERIOD;
67      stop_clk <= true;
68      wait;
69    end process;
70
71    stim7 : process
72      res_n <= '0';
73      wait until rising_edge(clk);
74      res_n <= '1';
75      wait for 6*CLK_PERIOD;
76      stop_clk <= true;
77      wait;
78    end process;
79
80    stim8 : process
81      res_n <= '0';
82      wait until rising_edge(clk);
83      res_n <= '1';
84      wait for 6*CLK_PERIOD;
85      stop_clk <= true;
86      wait;
87    end process;
88
89    stim9 : process
90      res_n <= '0';
91      wait until rising_edge(clk);
92      res_n <= '1';
93      wait for 6*CLK_PERIOD;
94      stop_clk <= true;
95      wait;
96    end process;
97
98    stim10 : process
99      res_n <= '0';
100     wait until rising_edge(clk);
101     res_n <= '1';
102     wait for 6*CLK_PERIOD;
103     stop_clk <= true;
104     wait;
105   end process;
106 end architecture bench;
107 end lfsr_tb;
```

Afterwards, we let the LFSR run for six clock periods and then stop the clock. The simulation will then be able to automatically terminate because all signals are stable. Finally, we want to remark that the general structure of this testbench can be used for any of the synchronous designs you will encounter in this course.

LFSR - Testbench

HWMoD
WS25

LFSR

Operation
VHDL Design
Testbench

```
1 architecture bench of lfsr_tb is
2   signal clk : std_ulogic;
3   signal res_n : std_ulogic;
4   signal o : std_ulogic;
5
6   constant CLK_PERIOD : time := 10 ns;
7   signal stop_clk : boolean := false;
8 begin
9
10  uut : entity work.lfsr
11    port map (
12      clk => clk,
13      res_n => res_n,
14      o => o
15  );
```

```
1   clkgen : process
2   begin
3     while not stop_clk loop
4       clk <= '0';
5       wait for 0.5*CLK_PERIOD;
6       clk <= '1';
7       wait for 0.5*CLK_PERIOD;
8     end loop;
9     wait;
10  end process;
11
12  stimulus : process
13  begin
14    res_n <= '0';
15    wait until rising_edge(clk);
16    wait until rising_edge(clk);
17    res_n <= '1';
18    wait for 6*CLK_PERIOD;
19    stop_clk <= true;
20    wait;
21  end process;
```

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

Lecture Complete!