Technische
Hochschule
Augsburg

VLSI Design

Sequential Circuit Design

J Färber

SS2026

Contents

Sequential Circuit Design

Learning Objectives

After completing this unit, you will be able to:

- Describe the function of a D-type flip-flop and its role as the fundamental storage element in synchronous circuits
- State and apply the three mandatory design rules for synchronous digital circuits
- Model a D-type flip-flop and an n-bit register using `WHEN ... ELSE` in VHDL
- Design an edge detector and derive its output logic from a truth table of flip-flop states
- Decompose a synchronous counter into next-state logic and state register, and model both using `WHEN ... ELSE`
- Model a shift register and describe the shift-right operation as a concatenation of flip-flop outputs
- Describe the structure of a Linear Feedback Shift Register and its application as a pseudo-random sequence generator
- Write a synchronous testbench with clock generation, asynchronous reset, and self-checking `ASSERT` statements

D-Type Flip-Flop / D-Type Register

Schematic Symbol

Function

Timing Diagram

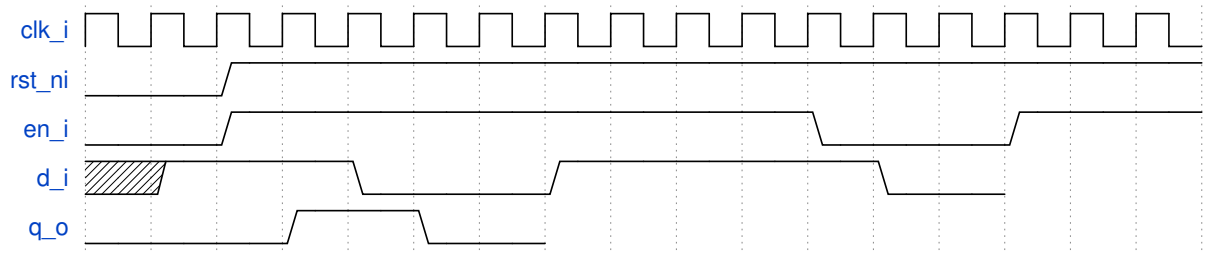


Figure 1: D-Type Flip-Flop - Timing Behaviour

VHDL Model

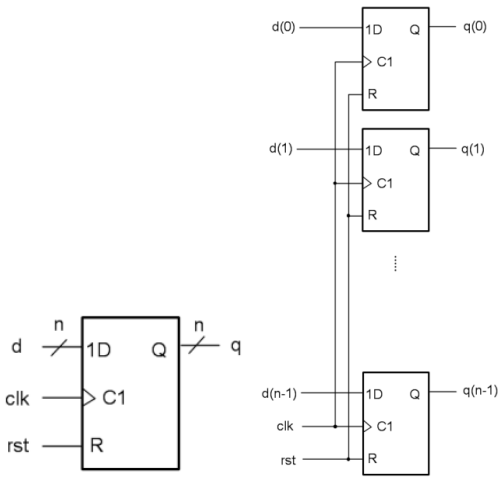
```vhdl
ENTITY d_ff IS
  PORT (clk_i  : IN  std_ulogic;
        rst_ni : IN  std_ulogic;


        d_i    : IN  std_ulogic;
        q_o    : OUT std_ulogic
       );
END d_ff;

ARCHITECTURE rtl OF d_ff IS

BEGIN

  dflipflop : q_o <=




END rtl;
```

```vhdl
ENTITY dregen IS
  PORT (clk_i  : IN  std_ulogic;
        rst_ni : IN  std_ulogic;


        d_i    : IN  std_ulogic
        q_o    : OUT std_ulogic
        );
END dregen;

ARCHITECTURE rtl OF dregen IS

BEGIN

dregister : q_o <=




END rtl;
```

Design Rules for Synchronous Digital Circuit Design

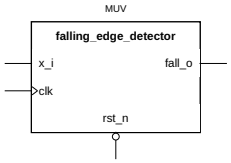
The general design rules for designing synchronous circuits are mandatory:

- There is only one common clock signal clk, which is connected to all clock inputs of n-bit storage elements
- A clock signal is never linked via logic elements.
- Each synchronous module has a reset signal, never connected via additional logic.



Falling Edge Detector

Schematic Symbol and RTL Diagram



Timing Diagram

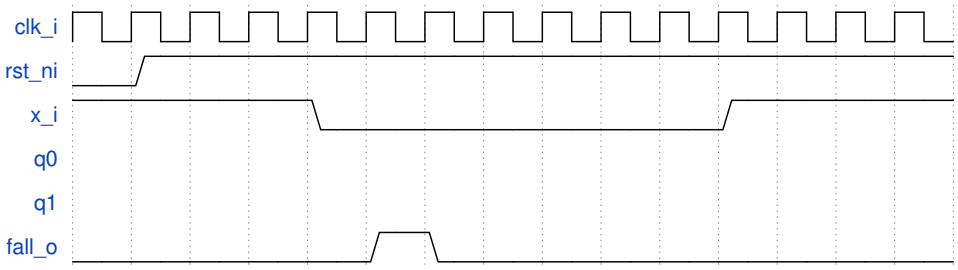


Figure 2: Falling Edge Detector - Waveform Diagram

Truth Table

```
q1 q0 | fall_o
------+-------
 0 0  |

 0 1  |

 1 0  |

 1 1  |

 fall_o = f(q1,q0)
```

VHDL Model

```
ENTITY falling_edge_detector IS
  PORT (
    clk_i  : IN  std_ulogic;
    rst_ni : IN  std_ulogic;
    x_i    : IN  std_ulogic;
    fall_o : OUT std_ulogic
    );
END falling_edge_detector;

ARCHITECTURE rtl OF falling_edge_detector IS

  SIGNAL q0, q1 : std_ulogic;           -- D-Type Flip-Flop outputs

BEGIN

  dflipflop_0 : q0 <=



  dflipflop_1 : q1 <=



  output_logic : fall_o <=

END rtl;
```

Simulation Environment

```
ENTITY t_falling_edge_detector IS
END t_falling_edge_detector;

ARCHITECTURE tbench OF t_falling_edge_detector IS

  COMPONENT falling_edge_detector
    PORT (
      clk_i  : IN  std_ulogic;
      rst_ni : IN  std_ulogic;
      x_i    : IN  std_ulogic;
      fall_o : OUT std_ulogic);
  END COMPONENT;

  -- definition of a clock period
  CONSTANT period : time    := 10 ns;
  -- switch for clock generator
  SIGNAL clken_p  : boolean := true;

  -- component ports
  SIGNAL clk_i  : std_ulogic;
  SIGNAL rst_ni : std_ulogic;
  SIGNAL x_i    : std_ulogic;
  SIGNAL fall_o : std_ulogic;
```

```
BEGIN  -- tbench

  -- Module Under Verification
  MUV : ENTITY work.falling_edge_detector(rtl)
    PORT MAP (
      clk_i  => clk_i,
      rst_ni => rst_ni,
      x_i    => x_i,
      fall_o => fall_o);

  -- clock generation
  clock_p : PROCESS
  BEGIN
    WHILE clken_p LOOP
      clk_i <= '0'; WAIT FOR period/2;
      clk_i <= '1'; WAIT FOR period/2;
    END LOOP;
    WAIT;
  END PROCESS;

  -- initial reset, always necessary at the beginning of a simulation
  reset : rst_ni <= '0', '1' AFTER period;

  stimuli_p : PROCESS
  BEGIN

    ------------------------------------------------------------------------
    -- wait until asynchronous reset is deactivated
    ------------------------------------------------------------------------


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


    ------------------------------------------------------------------------
    -- create a low-active pulse over a no. of clock periods
    ------------------------------------------------------------------------
                                                -- assign a '1' to x_i



                                                -- set input to '0' ...
                                                -- ... for a no. of periods


                                                -- assign a '1' to form a
                                                -- low active input pulse

    ------------------------------------------------------------------------
```

```
    ------------------------------------------------------------------------
    -- create a low-active pulse over a no. of clock periods
    ------------------------------------------------------------------------
    x_i <= '1';                         -- assign a '1' to x_i
    WAIT FOR period;

    x_i <= '0';                         -- set input to '0' ...
    WAIT UNTIL rising_edge(clk_i);
    WAIT UNTIL falling_edge(clk_i);
    ------------------------------------------------------------------------
    -- Observer: check, if fall_o is assigned to '1' for one clock period
    ------------------------------------------------------------------------
    ASSERT fall_o = '1' REPORT "Error: Expected fall_o = '1' !" SEVERITY failure;
    WAIT UNTIL falling_edge(clk_i);
    ASSERT fall_o = '0' REPORT "Error: Expected fall_o = '0' !" SEVERITY failure;
    WAIT FOR 6 * period;                -- ... for a no. of periods

    x_i <= '1';                         -- assign a '1' to form a
    WAIT FOR 3 * period;                -- low active input pulse
    ------------------------------------------------------------------------


    clken_p <= false;                   -- switch clock generator off

    WAIT;                               -- suspend proces
  END PROCESS;


END tbench;
```

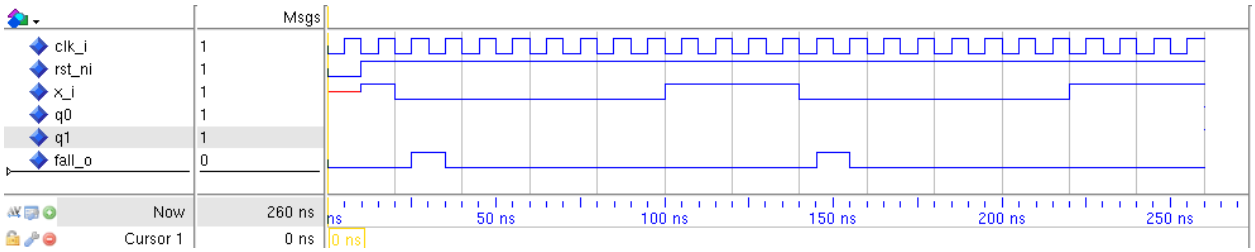


Figure 3: - Simulation Result

Counter

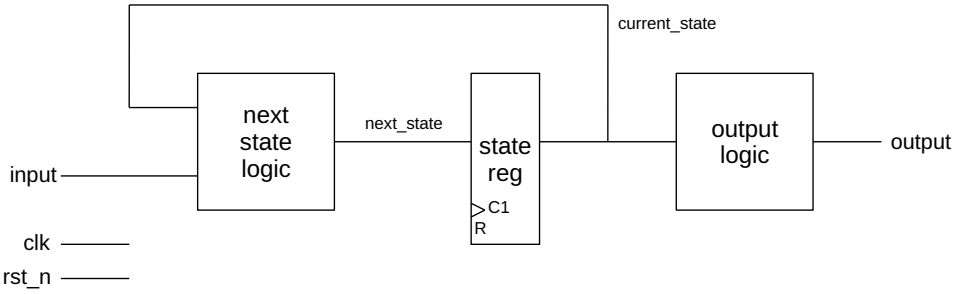
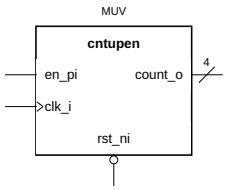


Figure 4: Syncronous Sequential Circuit - Conceptional Diagram

Schematic Symbol and RTL Diagram



Function

inputs			outputs	
rst_ni	clk_i	en_pi	q(t+1)	Remark
0	-	-	0	Asynchronous reset active
1	0	-	q(t)	Store count value
1	1	-	q(t)	Store count value
1	^	0	q(t)	Store count value
1	^	1	q(t) + 1	count = count + 1

VHDL Model

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY cntupen IS
  PORT (clk_i   : IN  std_ulogic;
        rst_ni  : IN  std_ulogic;
        en_pi   : IN  std_ulogic;
        count_o : OUT std_ulogic_vector(3 DOWNTO 0)
        );
END cntupen;

ARCHITECTURE rtl OF cntupen IS

  -- datatype unsigned is defined in package numeric_std
  SIGNAL next_state, current_state : unsigned(3 DOWNTO 0);

BEGIN

  -- package numeric_std overloads operator '+'
  -- for arguments of different types, here: unsigned and integer

  incrementer : next_state <=


  -- synthesisable construct of a d-type register with synchronrous enable

  state_register : current_state <=



  -- type conversion from unsignd to std_ulogic_vector necessary

  counter_output : count_o <=

END rtl;
```

Simulation Environment

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY t_cntupen IS
END t_cntupen;

ARCHITECTURE tbench OF t_cntupen IS

  COMPONENT cntupen
    PORT (
      clk_i   : IN  std_ulogic;
      rst_ni  : IN  std_ulogic;
      en_pi   : IN  std_ulogic;
      count_o : OUT std_ulogic_vector(3 DOWNTO 0));
  END COMPONENT;

  -- definition of a clock period
  CONSTANT period : time    := 10 ns;
  -- switch for clock generator
  SIGNAL clken_p  : boolean := true;

  -- component ports
  SIGNAL clk_i   : std_ulogic;
  SIGNAL rst_ni  : std_ulogic;
  SIGNAL en_pi   : std_ulogic;
  SIGNAL count_o : std_ulogic_vector(3 DOWNTO 0);

BEGIN  -- tbench

  -- component instantiation
  MUV : ENTITY work.cntupen(rtl)
    PORT MAP (
      clk_i   => clk_i,
      rst_ni  => rst_ni,
      en_pi   => en_pi,
      count_o => count_o);

  -- clock generation
  clock_p : PROCESS









  -- initial reset, always necessary at the beginning of a simulation
  ------------------------------------------------------------------------------
  -- Following a verification plan:
  -- ------------------------------
  -- 1. t = 0 ns: activate asynchronous reset
  -- 2. t = 10 ns: deactivate asynchronous reset
  ------------------------------------------------------------------------------
  reset :
```

```
  stimuli_p : PROCESS

  BEGIN

    --------------------------------------------------------------------------
    -- ... continuing with the verification plan:
    --------------------------------------------------------------------------
    WAIT UNTIL rst_ni = '1';            -- wait until asynchronous reset ...
                                        -- ... is deactivated
    --------------------------------------------------------------------------

    --------------------------------------------------------------------------
    -- 2. activate enable
    -- 3. Wait for a full counting cycle
    --------------------------------------------------------------------------
    en_pi <= '1';                       -- activate enable input en_pi
    WAIT FOR 15 * period;               -- wait for at least one count cycle
    ASSERT count_o = X"F" REPORT "Error: Expected count_o = F !" SEVERITY failure;
    --------------------------------------------------------------------------

    WAIT FOR 5 * period;                -- wait for a no. of count values
                                        -- then ...
    --------------------------------------------------------------------------

    --------------------------------------------------------------------------
    -- 4. After another five periods: Deactivate Enable
    --------------------------------------------------------------------------
    en_pi <= '0';                       -- .... deactivate enable input en_pi
    WAIT FOR 3 * period;
    --------------------------------------------------------------------------

    --------------------------------------------------------------------------
    -- 5. After another three periods: Activate Enable
    -- 6. Simulate another complete counting cycle
    --------------------------------------------------------------------------
    en_pi <= '1';                       -- activate enable input en_pi
    WAIT FOR 15 * period;               -- wait for at least one count cycle
    ASSERT count_o = X"3" REPORT "Error: Expected count_o = F !" SEVERITY failure;


    --------------------------------------------------------------------------
    -- 7. Switch off the clock generator
    --------------------------------------------------------------------------
    clken_p <= false;                   -- switch clock generator off

    WAIT;                               -- suspend proces
  END PROCESS;

END tbench;
```

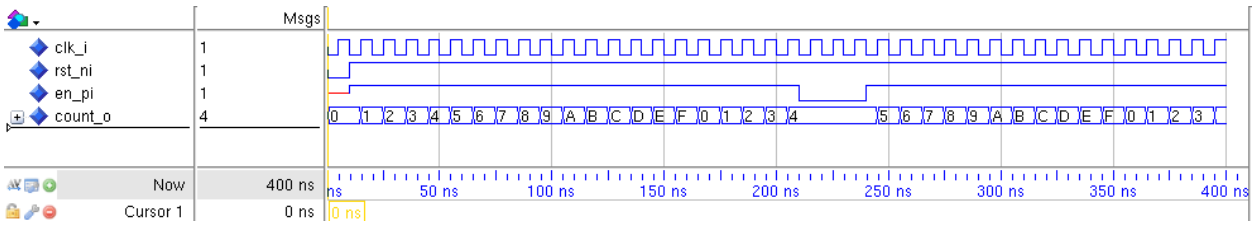


Figure 5: 4-Bit Binary Up Counter with Enable - Simuation Result

Prototype Test Environment

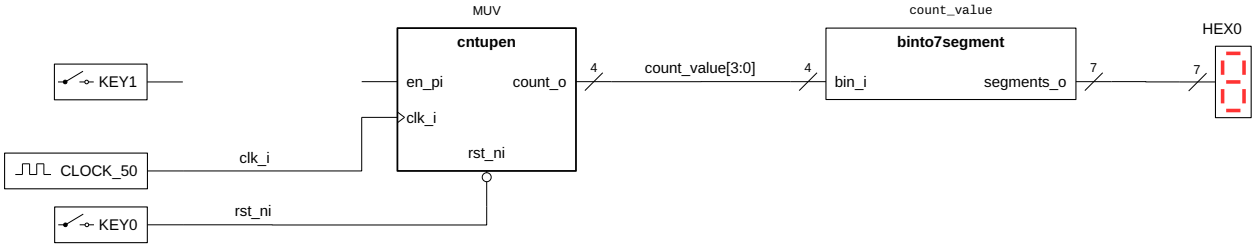


Figure 6: 4-Bit Binary Up Counter with Enable - DE1 Test Environemnt

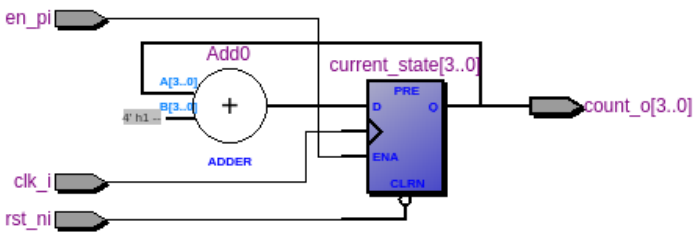


Figure 7: 4-Bit Binary Up Counter with Enable - Quartus RTL Viewer

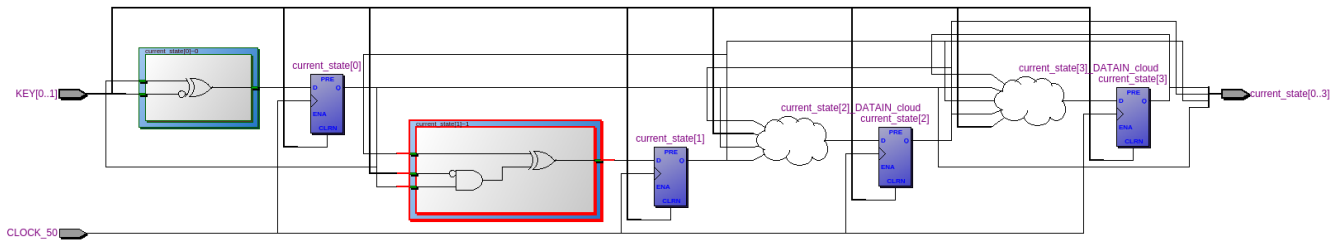


Figure 8: 4-Bit Binary Up Counter with Enable - Quartus Technology Map Viewer

Shift Register

Schematic Symbol

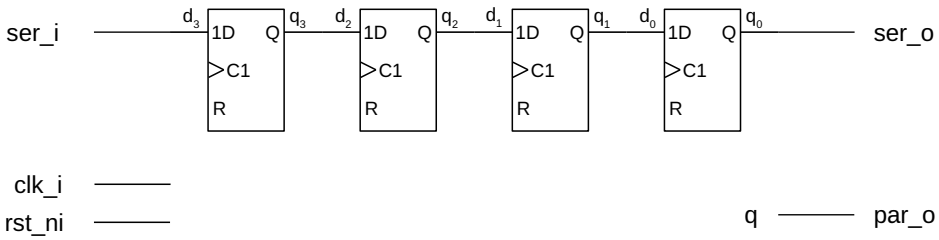


Figure 9: 4-Bit Shift Register - Schematic

Function

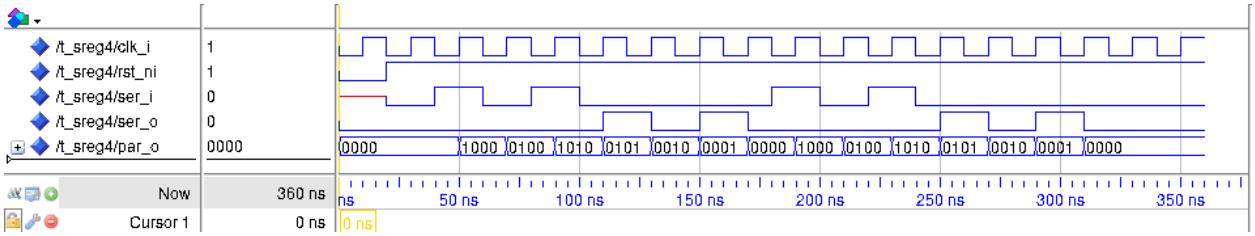


Figure 10: 4-Bit Shift Right Register - Simuation Result

VHDL Model

```
ENTITY sreg4 IS
  PORT (clk_i  : IN  std_ulogic;
        rst_ni : IN  std_ulogic;        -- asynchronous L-active reset
        ser_i  : IN  std_ulogic;        -- serial data stream input
        ser_o  : OUT std_ulogic;        -- serial data stream output
        par_o  : OUT std_ulogic_vector(3 DOWNTO 0)  -- parallel data output
        );
END sreg4;

ARCHITECTURE rtl OF sreg4 IS

  SIGNAL q : std_ulogic_vector(3 DOWNTO 0);
  SIGNAL d : std_ulogic_vector(3 DOWNTO 0);


  -- continuous on next page ...
```

```
BEGIN
```

Simulation Environment

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY t_sreg4 IS
END ENTITY t_sreg4;

ARCHITECTURE tbench OF t_sreg4 IS

  -- component ports
  SIGNAL clk_i  : std_ulogic;
  SIGNAL rst_ni : std_ulogic;
  SIGNAL ser_i  : std_ulogic;
  SIGNAL ser_o  : std_ulogic;
  SIGNAL par_o  : std_ulogic_vector(3 DOWNTO 0);

  -- definition of a clock period
  CONSTANT period : time    := 20 ns;
  -- switch for clock generator
  SIGNAL clken_p  : boolean := true;

BEGIN  -- ARCHITECTURE tbench

  -- component instantiation
  DUT : ENTITY work.sreg4
    PORT MAP (
      clk_i  => clk_i,
      rst_ni => rst_ni,
      ser_i  => ser_i,
      ser_o  => ser_o,
      par_o  => par_o);

  -- clock generation
  clock_p : PROCESS
  BEGIN
    WHILE clken_p LOOP
      clk_i <= '0'; WAIT FOR period/2;
      clk_i <= '1'; WAIT FOR period/2;
    END LOOP;
    WAIT;
  END PROCESS;

  -- initial reset, always necessary at the beginning OF a simulation
  reset : rst_ni <= '0', '1' AFTER period;

  -- process for stimuli generation
  stimuli_p : PROCESS

  BEGIN

    WAIT UNTIL rst_ni = '1';            -- wait until asynchronous reset ...
    -- ... is deactivated

    -- serial input data stream
    ser_i <= '0';
    WAIT FOR period;
    ser_i <= '1';
    WAIT FOR period;
    ser_i <= '0';
```

```
    WAIT FOR period;
    ser_i <= '1';
    WAIT FOR period;
    -- check parallel output


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

    ser_i <= '0';
    WAIT FOR 4 * period;
    -- check parallel output


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


    -- serial input data stream
    ser_i <= '1';
    WAIT FOR period;
    ser_i <= '0';
    WAIT FOR period;
    ser_i <= '1';
    WAIT FOR period;
    ser_i <= '0';
    WAIT FOR period;
    -- check parallel output


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

    WAIT FOR 4 * period;


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

    clken_p <= false;                   -- switch clock generator off

    WAIT;
  END PROCESS;
```

Pseudo-Random Binary Sequence Generator

Schematic

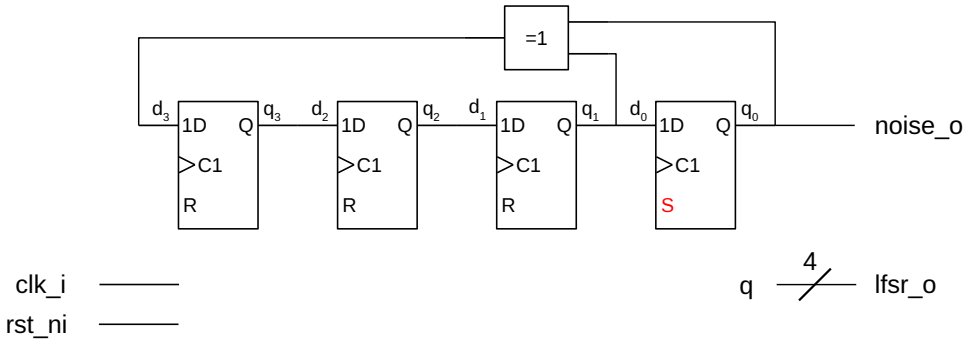


Figure 11: 4-bit Linear-Feedback Shift Register (Fibonacci) - Schematic

Function

VHDL Model

```
ENTITY lfsr4 IS
  PORT (clk_i   : IN  std_ulogic;
        rst_ni  : IN  std_ulogic;       -- asynchronous L-active reset
        en_pi   : IN  std_ulogic;       -- '1' -> enable data generation
        lfsr_o  : OUT std_ulogic_vector(3 DOWNTO 0);  -- lfsr data output
        noise_o : OUT std_ulogic;       -- lfsr serial data stream output
        eoc_po  : OUT std_ulogic);      -- end of cycle notification:
-- provides '1' WHEN lfsr_o = "00 .. 01"
END lfsr4;

ARCHITECTURE fibonacci OF lfsr4 IS

  SIGNAL q : std_ulogic_vector(lfsr_o'length-1 DOWNTO 0);  -- current state
  SIGNAL d : std_ulogic_vector(lfsr_o'length-1 DOWNTO 0);  -- next state

BEGIN

  -- linear feedback
  feedback :


  -- next state logic
  next_state_logic :


  -- state_register :
  state_register :



  -- parallel output
  parallel_out :


  -- serial output
  serial_out :


  -- end of cycle notification
  end_of_cycle :



END fibonacci;
```

Simulation Environment

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY t_lfsr4 IS
END ENTITY t_lfsr4;

ARCHITECTURE tbench OF t_lfsr4 IS

  -- component ports
  SIGNAL clk_i   : std_ulogic;
  SIGNAL rst_ni  : std_ulogic;
  SIGNAL en_pi   : std_ulogic;
  SIGNAL lfsr_o  : std_ulogic_vector(3 DOWNTO 0);
  SIGNAL noise_o : std_ulogic;
  SIGNAL eoc_po  : std_ulogic;


  -- definition of a clock period
  CONSTANT period : time    := 20 ns;
  -- switch for clock generator
  SIGNAL clken_p  : boolean := true;

BEGIN  -- ARCHITECTURE tbench

  -- component instantiation
  DUT : ENTITY work.lfsr4(fibonacci)
    PORT MAP (
      clk_i   => clk_i,
      rst_ni  => rst_ni,
      en_pi   => en_pi,
      lfsr_o  => lfsr_o,
      noise_o => noise_o,
      eoc_po  => eoc_po);

-- clock generation
  clock_p : PROCESS
  BEGIN
    WHILE clken_p LOOP
      clk_i <= '0'; WAIT FOR period/2;
      clk_i <= '1'; WAIT FOR period/2;
    END LOOP;
    WAIT;
  END PROCESS;

-- initial reset, always necessary at the beginning OF a simulation
  reset : rst_ni <= '0', '1' AFTER period;

-- process for stimuli generation
  stimuli_p : PROCESS


  BEGIN

    WAIT UNTIL rst_ni = '1';            -- wait until asynchronous reset ...
    -- ... is deactivated
```

```
    clken_p <= false;                   -- switch clock generator off

    WAIT;
  END PROCESS;

END ARCHITECTURE tbench;
```

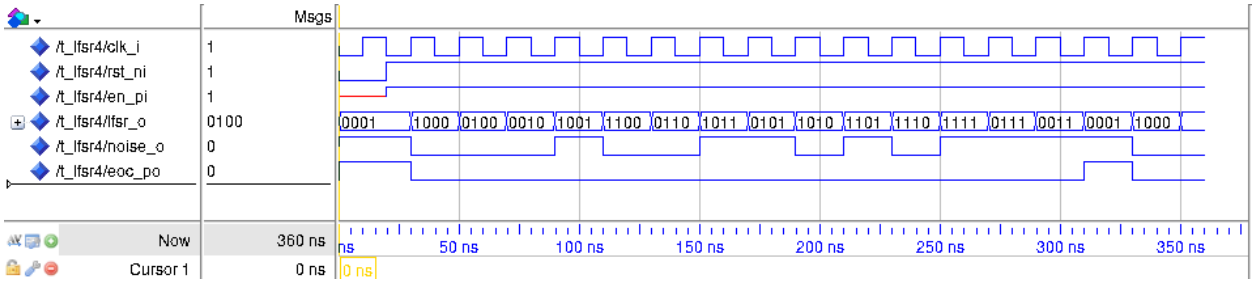


Figure 12: 4-Bit Linear Feedback Shift Register - Simulation Result

Summary

- The fundamental storage element is the **D-type flip-flop** — it samples its input `d_i` on the **rising edge** of `clk_i` and holds the value until the next clock edge
- Three **mandatory design rules** govern all synchronous circuits: one shared clock, no combinational logic on clock paths, reset never gated through additional logic
- An **n-bit register** is a collection of D flip-flops sharing clock and reset — modelled as a single `WHEN … ELSE` statement with `rising_edge(clk_i)`
- An **edge detector** cascades two flip-flops to compare the current and previous signal value — the output logic is a combinational function of both flip-flop states, derived from a truth table
- A **synchronous counter** separates next-state logic (incrementer) from the state register — both parts are modelled with `WHEN … ELSE`, using `unsigned` arithmetic from `ieee.numeric_std`
- A **shift register** implements serial-to-parallel conversion by chaining flip-flop outputs — the shift operation is expressed as a vector concatenation
- A **Linear Feedback Shift Register (LFSR)** generates a pseudo-random binary sequence through XOR feedback — a compact source of test stimuli and noise signals

> **Next Lecture:** Finite State Machine Design: Moore-type FSMs with enumerated state types and state diagrams

Reference

- [Mea25] Mealy, B., Tappero, F.: *Free Range VHDL*, 2025
  - Chapters 4.4, 6.5, 7, 13