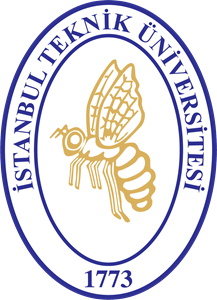
**DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF EXPERIMENT – 5**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

1. **D Flip-Flop**

**SR LATCH:**

A diagram of a truth table

Description automatically generated

Logic Diagram and Truth Table of SR NAND Latch1

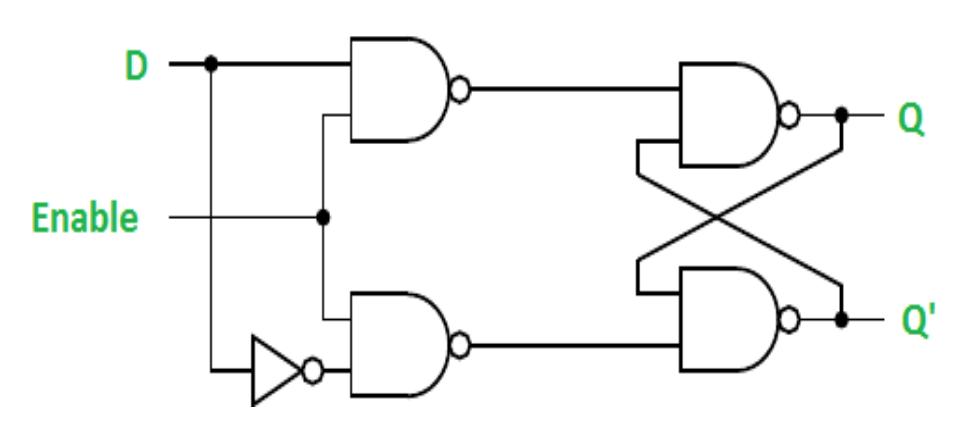
A math equations and formulas

Description automatically generated

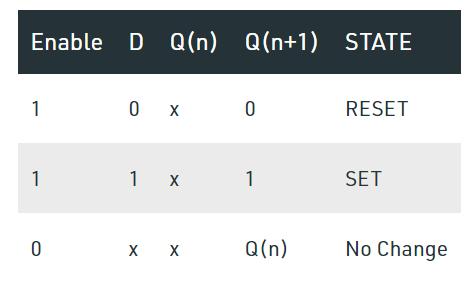
Characteristic Function of the SR NAND Latch

As it can be seen in the truth table the SR Latch has two inputs which are S’ as *SET* and R’ as *RESET.* Due to the designing with NAND gates to avoid affecting latching actions, the inputs are inverted in this circuit (active low). The circuit has 4 conditions which are set (S = 1, R = 0), reset (S = 0, R= 1), hold (S = 1, R = 1) and forbidden (S = 0, R = 0). The circuit sets Q = 1 in set condition, Q = 0 in reset condition, Qnext = Q in hold condition. However, in the forbidden condition the Q and Q’ outputs are both forced to be “1” which creates a logical error and unstable condition. Designer should avoid this condition via never setting S = R = 0.

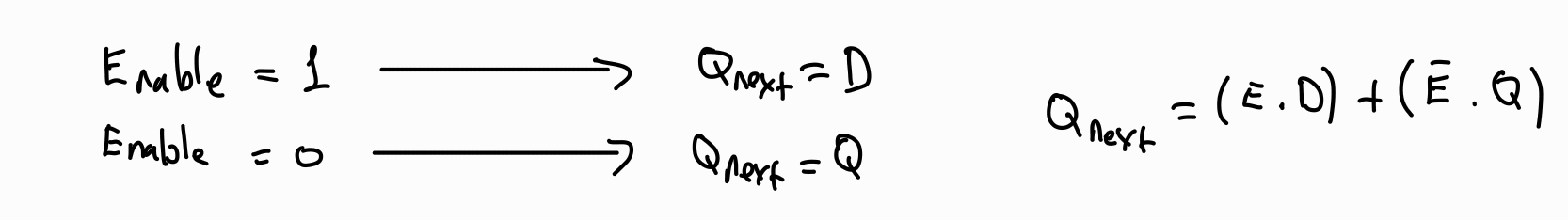
**GATED D LATCH:**



Logic Diagram of the Gated D Latch2



Truth Table of the Gated D Latch2



Characteristic Function of the Gated D Latch

**EDGE-TRIGGERED D FLIP-FLOPs:**

An Edge-Triggered D Flip-Flop is a sequential circuit that stores data on the rising or falling edge of a clock signal. It is a key component in digital systems used for synchronizing data and for creating registers, counters, and memory elements. Unlike a latch, which is level-sensitive, a flip-flop changes its output only on the edge of the clock signal.

A Master-Slave Flip-Flop is a type of edge-triggered flip-flop that uses two latches which are called the master latch, and the slave latch connected in series. In D Flip-Flop Master latch is a level-sensitive D latch controlled by the clock and accepts the input when the clock is low and latches the input when the clock transitions to high state. However, Slave latch is controlled by the inverted clock signal and becomes transparent when the clock is high, latching the output of the master latch.

A diagram of a circuit

Description automatically generated

Master-slave D Flip-Flop 3

When CLK = 0, the input D is propagated to the master latch output Qm and the slave latch is in the latched state, so the final output Q remains unchanged. On the contrary when CLK = 1, the master latch is in the latched state, so that Qm stays the same and the slave latch propagates Qm to the final output Q.

In an edge-sensitive circuit, the output changes only on the edge of a clock signal, either the rising edge (low-to-high transition) or the falling edge (high-to-low transition). In Edge-Triggered D Flip-Flops the input D is sampled and stored only at the clock edge (e.g., rising edge for a positive-edge-triggered D flip-flop). This provides more precise control of data timing, making it ideal for synchronous systems and minimizes the risk of glitches caused by noise or unintended input transitions.3

In a level-sensitive circuit, the output changes continuously if the clock signal is active (high or low). A latch is level sensitive. When the enable signal (or clock) is active, the input propagates to the output immediately. When the enable signal is inactive, the latch holds its previous state. This provides a simpler design and faster response compared to edge-sensitive circuits. However, it has disadvantages over edge-sensitive design like less robust in synchronous designs because inputs can inadvertently propagate to the output multiple times within one clock cycle.3

Due to such differences edge-triggered D Flip-Flops are commonly used in applications such as registers, counters, Finite State Machines (FSM), pipelined system design, etc.

**DFF with Synchronous Reset:**

**A diagram of a circuit

Description automatically generated**

D Flip-Flop with Synched Reset RTL Schematic

A green squares on a black background

Description automatically generated

D Flip-Flop with Synched Reset Behavioral Simulation

D Flip-Flop Synched Reset Verilog Code

`timescale 1ns **/** 1ps

**module** DFF\_sync**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

**output** **reg** Q

**);**

**always** **@(posedge** clk**)** **begin**

**if(**rst**)** Q **<=** 1'b0**;**

**else** Q **<=** D**;**

**end**

**endmodule**

D Flip-Flop Synched Reset Testbench Code

`timescale 1ns **/** 1ps

**module** DFF\_sync\_tb**;**

**reg** clk**,** rst**,** D**;**

**wire** Q**;**

DFF\_sync uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**D**(**D**),**

**.**Q**(**Q**)**

**);**

**always** **begin**

clk **=** 1'b0**;** **#**5**;**

clk **=** 1'b1**;** **#**5**;**

**end**

**initial** **begin**

rst **=** 1'b0**;**

D **=** 1'b0**;**

rst **=** 1'b1**;** **#**15**;**

rst **=** 1'b0**;**

D **=** 1'b1**;** **#**20**;**

D **=** 1'b0**;** **#**20**;**

D **=** 1'b1**;** **#**10**;**

rst **=** 1'b1**;** **#**10**;**

rst **=** 1'b0**;** **#**10**;**

D **=** 1'b1**;** **#**30**;**

D **=** 1'b0**;** **#**30**;**

$finish**();**

**end**

**endmodule**

A screenshot of a computer

Description automatically generated

D Flip-Flop Synched Reset Primitive Usages

A computer screen shot of a computer

Description automatically generated

D Flip-Flop Synched Reset Technoogy Schematic

**DFF with Asynchronous Reset:**

A yellow square with green lines and a black text

Description automatically generated with medium confidence

D Flip-Flop Asynchronous Reset RTL Schematic

**module** DFF\_async**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

**output** **reg** Q

**);**

**always** **@(posedge** clk **or** **negedge** rst**)** **begin**

**if(!**rst**)** Q **<=** 1'b0**;**

**else** Q **<=** D**;**

**end**

**endmodule**

D Flip-Flop Asynchronous Reset Verilog Code

A screenshot of a computer

Description automatically generated

D Flip-Flop Asynchronous Reset Behavioral Simulation

D Flip-Flop Asynchronous Reset Testbench Code

`timescale 1ns **/** 1ps

**module** DFF\_async\_tb**;**

**reg** clk**,** rst**,** D**;**

**wire** Q**;**

DFF\_async uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**D**(**D**),**

**.**Q**(**Q**)**

**);**

**always** **begin**

clk **=** 1'b0**;** **#**5**;**

clk **=** 1'b1**;** **#**5**;**

**end**

**initial** **begin**

rst **=** 1'b0**;**

D **=** 1'b0**;**

rst **=** 1'b1**;** **#**15**;**

rst **=** 1'b0**;**

D **=** 1'b1**;** **#**20**;**

D **=** 1'b0**;** **#**20**;**

D **=** 1'b1**;** **#**10**;**

rst **=** 1'b1**;** **#**10**;**

rst **=** 1'b0**;** **#**10**;**

D **=** 1'b1**;** **#**30**;**

D **=** 1'b0**;** **#**30**;**

$finish**();**

**end**

**endmodule**

A screenshot of a computer

Description automatically generated

D Flip-Flop Asynchronous Reset Technology Schematic

A screenshot of a computer

Description automatically generated

D Flip-Flop Asynchronous Reset Primitive Usages

In the asynchronous design, there is an extra LUT1 used for the design. This is because the asynchronous design requires to be affected by “reset” input independent from clock. A LUT1 is a simple 1-input LUT (look-up table), often used to directly implement a constant or simple logic, such as forcing Q = 0 when rst = 1. However, in a synchronous flip-flop, the reset signal is handled within the clocked data path of the flip-flop itself. It is combined with the data (D) input using internal gates and does not require extra external logic (e.g., a LUT).

1. **8-bit Shift Register**

8-bit Shift Register Verilog Code

`timescale 1ns **/** 1ps

**module** shift8**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

**output** **reg** **[**7**:**0**]** Q

**);**

**always** **@(posedge** clk**)** **begin**

**if(**rst**)** Q **<=** 8'b0**;**

**else** **begin**

Q **<=** **{**Q**[**6**:**0**],** D**};**

**end**

**end**

**endmodule**

A screenshot of a video game

Description automatically generated

8-bit Shift Register Behavioral Simulation

A diagram of a circuit

Description automatically generated

8-bit Shift Register RTL Schematic

A diagram of a computer

Description automatically generated

8-bit Shift Register Technology Schematic

A screenshot of a graph

Description automatically generated

8-bit Shift Register Utilization Summary

A screenshot of a computer

Description automatically generated

8-bit Shift Register Primitives

1. **Clock Divider with Stopwatch Example**

Clock Divider Verilog Code

`timescale 1ns **/** 1ps

**module** clk\_divider **#(parameter** **[**27**:**0**]** CLK\_DIV **=** 100**)(**

**input** clk\_in**,**

**input** rst**,**

**output** **reg** clk\_out

**);**

**reg** **[**27**:**0**]** counter**;**

**always** **@(posedge** clk\_in**)** **begin**

**if(**rst**)** **begin**

counter **<=** 28'b0**;**

clk\_out **<=** 1'b0**;**

**end**

**else** **begin**

**if(**counter **==** CLK\_DIV **-** 1**)** **begin**

counter **<=** 28'b0**;**

clk\_out **<=** **~**clk\_out**;**

**end** **else** **begin**

counter **<=** counter **+** 1**;**

**end**

**end**

**end**

**endmodule**

A screenshot of a phone

Description automatically generated

Clock Divider Behavioral Simulation for 1 MHz

A screenshot of a computer

Description automatically generated

Clock Divider Behavioral Simulation for 100 Hz

A screenshot of a phone

Description automatically generated

Clock Divider Behavioral Simulation for 1 Hz

`timescale 1ns **/** 1ps

**module** stopwatch**(**

**input** clk\_in**,**

**input** rst**,**

**output** **reg** **[**7**:**0**]** AN**,** // Anode : select 7-segment displays

**output** **reg** **[**6**:**0**]** CAT // Catode : select 7-segment LEDs

**);**

**wire** clk\_out1**;** // 1sec clk

**wire** clk\_out100**;** // 10msec clk (100Hz)

**reg** **[**13**:**0**]** cnt**;** // 1sec timer

**wire** **[**15**:**0**]** cnt\_bcd**;** // BCD converted

**reg** **[**3**:**0**]** display**;** // display number

**reg** **[**1**:**0**]** refresh\_cnt**;** // refresh counter

clk\_divider **#(** **.**CLK\_DIV**(**50000000**)** **)**

CLKDIV1**(**

**.**clk\_in **(**clk\_in**),**

**.**rst **(**rst**),**

**.**clk\_out**(**clk\_out1**)**

**);**

// CLK DIVIDER 100Hz

clk\_divider **#(** **.**CLK\_DIV**(**500000**)** **)**

CLKDIV100**(**

**.**clk\_in **(**clk\_in**),**

**.**rst **(**rst**),**

**.**clk\_out**(**clk\_out100**)**

**);**

// Binary to Decimal Conversion

bin2bcd BIN2BCD**(**

**.**bin**(**cnt**),**

**.**bcd**(**cnt\_bcd**)**

**);**

// Timer 1sec

**always** **@(posedge** clk\_out1**,** **posedge** rst**)**

**begin**

**if(** rst **)**

**begin**

cnt **<=** 0**;**

**end**

**else**

**begin**

**if(** cnt **==** 9999 **)**

cnt **<=** 0**;**

**else**

cnt **<=** cnt **+** 1**;**

**end**

**end**

// Refresh 7-segment displays

**always** **@(posedge** clk\_out100**,** **posedge** rst**)**

**begin**

**if(** rst **)**

**begin**

AN **<=** 8'b1111\_0000**;**

display **<=** 4'b0000**;**

refresh\_cnt **<=** 0**;**

**end**

**else**

**begin**

**case(** refresh\_cnt **)**

2'b00**:** **begin**

AN **<=** 8'b1111\_1110**;**

display **<=** cnt\_bcd**[**3**:**0**];**

refresh\_cnt **<=** refresh\_cnt **+** 1**;**

**end**

2'b01**:** **begin**

AN **=** 8'b1111\_1101**;**

display **<=** cnt\_bcd**[**7**:**4**];**

refresh\_cnt **<=** refresh\_cnt **+** 1**;**

**end**

2'b10**:** **begin**

AN **=** 8'b1111\_1011**;**

display **<=** cnt\_bcd**[**11**:**8**];**

refresh\_cnt **<=** refresh\_cnt **+** 1**;**

**end**

2'b11**:** **begin**

AN **=** 8'b1111\_0111**;**

display **<=** cnt\_bcd**[**15**:**12**];**

refresh\_cnt **<=** refresh\_cnt **+** 1**;**

**end**

**endcase**

**end**

**end**

// Select 7-segment LEDs

**always** **@(\*)**

**begin**

**case(**display**)**

4'b0000**:** CAT **=** 7'b0000001**;** // "0"

4'b0001**:** CAT **=** 7'b1001111**;** // "1"

4'b0010**:** CAT **=** 7'b0010010**;** // "2"

4'b0011**:** CAT **=** 7'b0000110**;** // "3"

4'b0100**:** CAT **=** 7'b1001100**;** // "4"

4'b0101**:** CAT **=** 7'b0100100**;** // "5"

4'b0110**:** CAT **=** 7'b0100000**;** // "6"

4'b0111**:** CAT **=** 7'b0001111**;** // "7"

4'b1000**:** CAT **=** 7'b0000000**;** // "8"

4'b1001**:** CAT **=** 7'b0000100**;** // "9"

**default:** CAT **=** 7'b0000001**;** // "0"

**endcase**

**end**

**endmodule**

Stopwatch Verilog Code

**A screenshot of a computer

Description automatically generated**

Stopwatch Behavioral Simulation

1. **Sliding LEDs**

**A screenshot of a computer

Description automatically generated**

Sliding LEDs Behavioral Simulation

Sliding LEDs Verilog Code

`timescale 1ns **/** 1ps

**module** sliding\_leds **#(parameter** **[**23**:**0**]** CLK\_DIV **=** 10000000**)(**

**input** clk**,**

**input** rst**,**

**input** **[**1**:**0**]** SW**,**

**output** **reg** **[**15**:**0**]** LED

**);**

**reg** **[**23**:**0**]** counter**;**

**reg** **[**23**:**0**]** desired\_rate**;**

**always** **@(posedge** clk**)** **begin**

**if(**rst**)** **begin**

counter **<=** 24'b0**;**

LED **<=** 16'b0000\_0000\_0000\_0001**;**

**end** **else** **begin**

**case(**SW**)**

2'b00**:** desired\_rate **<=** 24'b0**;**

2'b01**:** desired\_rate **<=** CLK\_DIV**;** //10 Hz

2'b10**:** desired\_rate **<=** CLK\_DIV **/** 2**;** //20 Hz

2'b11**:** desired\_rate **<=** CLK\_DIV **/** 5**;** //50 Hz

**default:** desired\_rate **<=** 24'b0**;**

**endcase**

**if(**desired\_rate **!=** 24'b0**)** **begin**

**if(**counter **==** desired\_rate **-** 1**)** **begin**

counter **<=** 24'b0**;**

LED **<=** **{**LED**[**14**:**0**],**LED**[**15**]};**

**end** **else** **begin**

counter **<=** counter **+** 1**;**

**end**

**end**

**end**

**end**

**endmodule**

**A screenshot of a computer

Description automatically generated**

clk\_pin Path Delays Setup

**A screenshot of a computer

Description automatically generated**

clk\_pin Path Delays Setup

**A screenshot of a computer

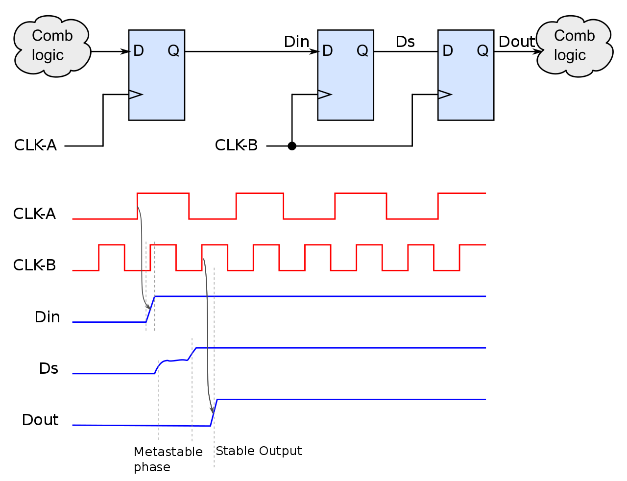
Description automatically generated**

Post-Implementation Timing Simulation

1. **Research**

**Static Timing Analysis (STA)** is a technique used to validate the timing of a digital circuit without requiring simulation. It calculates the propagation delays through combinational paths and verifies that setup and hold time constraints are satisfied for all sequential elements, such as flip-flops and latches. Setup time is the minimum period before a clock edge during which the input data must remain stable to be correctly captured. Conversely, hold time refers to the minimum period after the clock edge that data must remain stable to ensure proper latching. Violating these constraints can lead to errors, making STA critical for ensuring reliable operation. The maximum clock frequency of a sequential circuit is determined by the total propagation delay, clock-to-Q delay, setup time, and any clock skew. This frequency is calculated using fmax=1/Tclockf\_{max} = 1 / T\_{clock}fmax​=1/Tclock​, where TclockT\_{clock}Tclock​ includes all timing delays.5

**Metastability** occurs when a flip-flop enters an unstable state due to a violation of setup or hold times. This situation often arises when asynchronous signals or signals from different clock domains interact with the clocked system. In a metastable state, the flip-flop's output hovers unpredictably between logic levels, potentially causing incorrect behavior. To prevent metastability, designers use techniques such as double-flopping, where a chain of flip-flops stabilizes asynchronous inputs. Proper timing margins, slower clock speeds, and encoding multi-bit signals with Gray Code further reduce the likelihood of metastable states. Additionally, asynchronous FIFO designs are effective for managing clock domain crossings.4



Metastability Figure4

**References**

1. <https://electronics-course.com/sr-nand-latch>
2. <https://evertutorial.com/articles/DigitalDesign/Gated_D_Latch>
3. Brown, S. D., & Vranesic, Z. G. (2013). Fundamentals of Digital Logic with Verilog Design. (p. 247-331).
4. <https://en.wikipedia.org/wiki/Metastability_(electronics)#:~:text=In%20electronics%2C%20metastability%20is%20the,unstable%20equilibrium%20or%20metastable%20state>.
5. ChatGPT