

**DIGITAL SYSTEM DESIGN APPLICATION**

**EHB436E CRN: 11280**

**Salih Ömer Ongün**

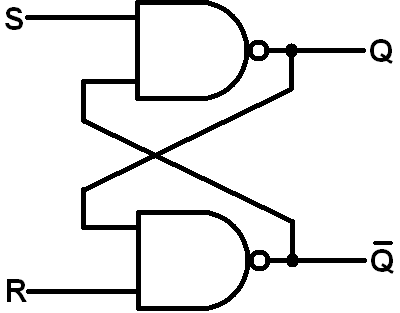
**040220780**

**Experiment 5**

**D FLIP -FLOP**

**SR NAND LATCH**

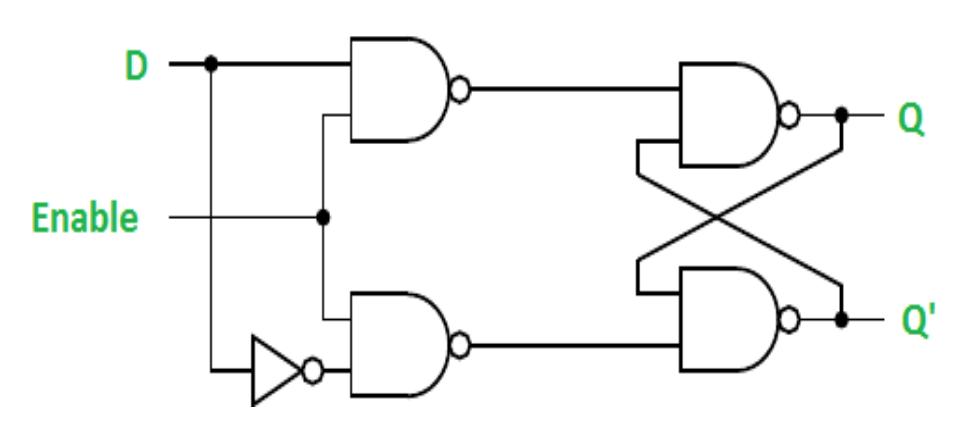
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S** | | **R** |  | **Q** | **Q’** |
| **0** | **1** | |  | **1** | **0** |
| **1** | **1** | | **No Change** | |
| **1** | **0** | | **0** | **1** |
| **0** | **0** | |  | **Forbidden** | |



**Characteristic Function: Q(t+1) = R’ + S’ \* Q(t)**

When the S = 0, the output Q = 1. This is called the "Set" state. When the R = 1, the output Q = 0. This is called the "Reset" state. When S = 1 and R = 1, the latch retains its previous state. The outputs Q and Q' do not change. When S = 0 and R = 0, the outputs Q and Q' both become 1. Q and Q’ have the same logic value. This is an invalid or forbidden state and should be avoided.

|  |  |  |  |
| --- | --- | --- | --- |
| **E** | **D** |  | **Q(n+1)** |
| **0** | **x** | **No change (Q(n))** |
| **1** | **0** | **0** |
| **1** | **1** |  | **1** |

**GATED D LATCH**

When the E =0, output cannot change independent of D. When E = 1 and D =0, output is logic 0. This is called “Reset” state. When E = 1 and D =1, output is logic 1. This is called “Set” state.

**Characteristic Function: Q(n+1) = E.D + E’.Q(n) *[1]***

**Edge Trigged D Flip Flop**

diyagram, çizgi, ekran görüntüsü, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

This is Master-Slave falling edge D flip flop. It consists of two latches. If CLK = 1, Master load with new data and transfer to Qm, Slave cannot transfer Qm to output. If CLK =0, Slave transfer to Qm value to Q output but Master lacth cannot transfer new data to Qm.

Flip flops transfer data to the output when CLK transitions from 1 to 0 or from 0 to 1. This is called Edge Sensitivity. Flip flops transfer data to output when CLK is 1 or 0. This is called Level Sensitivity.

**DFF With Synchronous Reset**

**Design Source**

**module** DFF\_sync

**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

**output** Q

**);**

**reg** Q\_reg**;**

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

**if(**rst**==**1'b1**)** **begin**

Q\_reg **<=** 1'b0**;**

**end**

**else** **begin**

Q\_reg **<=** D**;**

**end**

**end**

**assign** Q **=** Q\_reg**;**

**Simulation Source**

**module** DFF\_sync\_tb**();**

**reg** CLK**=** 1'b0**;**

**reg** RST**=** 1'b0**;**

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

**wire** Q**;**

DFF\_sync uut **(**

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

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

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

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

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;**

**end**

**initial**

**begin**

RST **=** 1**;**

**#**10**;**

RST **=** 0**;**

**#**10**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

RST **=** 1**;**

**#**10**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

$finish**;**

**end**

**Simulation Wave**

ekran görüntüsü, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

As seen in the simulation, the module works as it should. Reset signal depend on the rising edge of clock signal. If the reset signal is logical, Q will be logical 0. Otherwise, Q will be loaded with the value D, which is the rising edge of the clock signal.

**RTL Schematic**

diyagram, ekran görüntüsü, çizgi, plan içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Technology Schematic**

metin, diyagram, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Utilization Report**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**DFF With Active – Low Asynchronous Reset**

**Design Source**

**module** DFF\_async

**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

**output** Q

**);**

**reg** Q\_reg**;**

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

**if(**rst**==**1'b0**)** **begin**

Q\_reg **<=** 1'b0**;**

**end**

**else** **begin**

Q\_reg **<=** D**;**

**end**

**end**

**assign** Q **=** Q\_reg**;**

**endmodule**

**Simulation Source**

**module** DFF\_async\_tb**();**

**reg** CLK**=** 1'b0**;**

**reg** RST**=** 1'b0**;**

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

**wire** Q**;**

DFF\_async uut **(**

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

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

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

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

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;**

**end**

**initial**

**begin**

RST **=** 1**;**

**#**10**;**

RST **=** 0**;**

**#**10**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

RST **=** 1**;**

**#**10**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

$finish**;**

**end**

**endmodule**

**ekran görüntüsü, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturulduSimulation Wave**

As seen in the simulation, the module works as it should. The reset signal is independent of the rising edge of the clock signal. Output Q can be switched when there is a rising edge of the clock signal and a falling edge of the reset signal. If the reset signal is logic 0, Q will be logic 0. Otherwise, Q will be loaded with input D value.

**RTL Schematic**

**diyagram, plan, çizgi, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Technology Schematic**

**metin, diyagram, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturulduUtilization Report**

Both synchronous and asynchronous circuits have the same resource usage with one exception. Asynchronous reset is an active low reset so it uses an extra LUT.

**8 – BIT SHIFT REGISTER**

**Design Source**

**module** shift8

**(**

**input** clk**,**

**input** rst**,**

**input** D**,**

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

**);**

**reg** **[**7**:**0**]** Q\_reg**;**

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

**if(**rst**==**1'b1**)** **begin**

Q\_reg **<=** 8'b0**;**

**end**

**else** **begin**

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

**end**

**end**

**assign** Q **=** Q\_reg**;**

**endmodule**

**Simulation Source**

**module** shift8\_tb**();**

**reg** CLK**=**1'b0**;**

**reg** RST**=**1'b0**;**

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

**wire** **[**7**:**0**]** Q**;**

shift8 uut **(**

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

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

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

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

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;**

**end**

**initial**

**begin**

RST **=** 1**;**

**#**10 RST **=** 0**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

D **=** 1**;**

**#**10**;**

RST **=** 1**;**

D **=** 1**;**

**#**10**;**

D **=** 0**;**

**#**10**;**

$finish**;**

**end**

**endmodule**

**ekran görüntüsü, multimedya yazılımı, grafik yazılımı, renklilik içeren bir resim

Açıklama otomatik olarak oluşturulduSimulation Wave**

As seen in the simulation, the module works as it should. When a clock signal is given, the least significant bit of Q is loaded via input D and the outputs are shifted one bit towards the most significant bit. When there is a rising edge of the clock signal, reset is logic 1, Q outputs have logic 0 value.

**RTL Schematic**

**diyagram, çizgi, plan, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Technology Schematic**

**diyagram, metin, plan, şematik içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Utilization Report**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**CLOCK DIVIDER**

**Design Source**

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

**(**

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

**input** rst**,**

**output** clk\_out

**);**

**reg** clk\_out\_reg**;**

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

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

**if(**rst**==**1'b1**)** **begin**

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

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

**end**

**else** **begin**

**if(**counter **==((**CLK\_DIV**/**2**)-**1**))begin**

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

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

**end**

**else** **begin**

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

**end**

**end**

**end**

**assign** clk\_out **=** clk\_out\_reg**;**

**endmodule**

**Simulation Source and Simulation Wave**

**Simulation for 1Hz clock**

**module** clk\_divider\_tb**();**

**parameter** **[**27**:**0**]** CLK\_DIV **=** 100000000**;** //clk\_out = 1Hz

**reg** CLK\_IN**=**1'b0**;**

**reg** RST**=**1'b0**;**

**wire** CLK\_OUT**;**

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

**(**

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

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

**.**clk\_out**(**CLK\_OUT**)**

**);**

**always** **begin**

**#**5 CLK\_IN **=** **~**CLK\_IN**;**

**end**

**initial** **begin**

RST **=** 1**;**

**#**10**;**

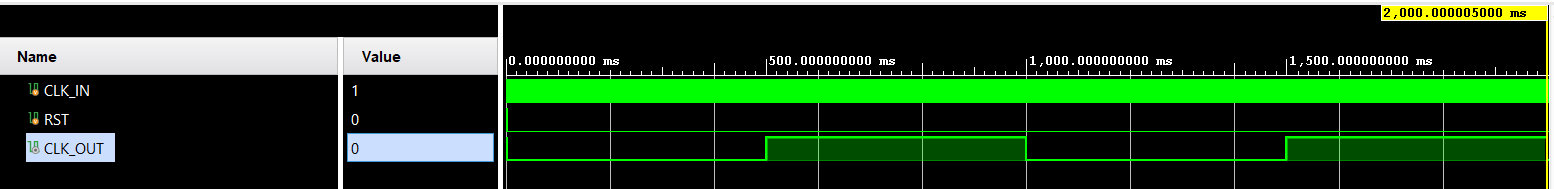
RST **=** 0**;**

**#**2000**;**

$finish**;**

**end**

**endmodule**

**** We have a 100 MHz (10^8 Hz) internal clock frequency. If we want to have 1 Hz clock frequency, we have to divide the internal clock frequency by 10^8. The period of a 1Hz signal is 1 second. We have a simulation interval of 2000 ms(2 s). Therefore, there are two periods of 1Hz clock period in the wave.

**Simulation for 100 Hz clock**

**module** clk\_divider\_tb**();**

**parameter** **[**27**:**0**]** CLK\_DIV **=** 1000000**;** //clk\_out = 100Hz

**reg** CLK\_IN**=**1'b0**;**

**reg** RST**=**1'b0**;**

**wire** CLK\_OUT**;**

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

**(**

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

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

**.**clk\_out**(**CLK\_OUT**)**

**);**

**always** **begin**

**#**5 CLK\_IN **=** **~**CLK\_IN**;**

**end**

**initial** **begin**

RST **=** 1**;**

**#**10**;**

RST **=** 0**;**

**#**2000**;**

$finish**;**

**end**

**endmodule**

ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

We have a 100 MHz (10^8 Hz) internal clock frequency. If we want to have 100 Hz clock frequency, we have to divide the internal clock frequency by 10^6. The period of a 100Hz signal is 0.01(10 ms) second. We have a simulation interval of 20 ms. Therefore, there are two periods of 100Hz clock period in the wave.

**Simulation for 1MHz clock**

**module** clk\_divider\_tb**();**

**parameter** **[**27**:**0**]** CLK\_DIV **=** 100**;** //clk\_out = 1MHz

**reg** CLK\_IN**=**1'b0**;**

**reg** RST**=**1'b0**;**

**wire** CLK\_OUT**;**

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

**(**

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

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

**.**clk\_out**(**CLK\_OUT**)**

**);**

**always** **begin**

**#**5 CLK\_IN **=** **~**CLK\_IN**;**

**end**

**initial** **begin**

RST **=** 1**;**

**#**10**;**

RST **=** 0**;**

**#**2000**;**

$finish**;**

**end**

**endmodule**

ekran görüntüsü, multimedya yazılımı, grafik yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

We have a 100 MHz (10^8 Hz) internal clock frequency. If we want to have 1 MHz clock frequency, we have to divide the internal clock frequency by 10^2. The period of a 1MHz signal is 10^-6(100 us = 1000 ns) second. We have a simulation interval of 2000 ns. Therefore, there are two periods of 1MHz clock period in the wave.

**STOP\_WATCH**

**Design Source**

// 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**

**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

///////////////////////////////////////////////

////\*\*\*\* INSTANTIATE YOUR CLK\_DIVIDERs \*\*\*\*////

///////////////////////////////////////////////

// CLK DIVIDER 1Hz

clk\_divider **#(** **.**CLK\_DIV**(**100000000**)** **)**

CLKDIV1**(**

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

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

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

**);**

// CLK DIVIDER 100Hz

clk\_divider **#(** **.**CLK\_DIV**(**1000000**)** **)**

CLKDIV100**(**

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

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

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

**);**

////////////////////////////////////////////////

////////////////////////////////////////////////

// Binary to Decimal Conversion

bin2bcd BIN2BCD**(**

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

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

**);**

**Simulation Source**

**module** stopwatch\_tb**();**

**reg** clk\_in **=** 0**;**

**reg** rst **=** 0**;**

**wire** **[**7**:**0**]** AN**;** // Anode : select 7-segment displays

**wire** **[**6**:**0**]** CAT**;** // Catode : select 7-segment LEDs

stopwatch STW**(**

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

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

**.**AN**(**AN**),**

**.**CAT**(**CAT**)**

**);**

**always** **#**5 clk\_in **=** **~**clk\_in**;**

**initial**

**begin**

**repeat(**100**)** **@(posedge** clk\_in**);**

rst **=** 1**;**

**repeat(**100**)** **@(posedge** clk\_in**);**

rst **=** 0**;**

**repeat(**100**)** **@(posedge** clk\_in**);**

**end**

**endmodule**

ekran görüntüsü, bilgisayar, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

As seen in the simulation, the module works as it should. Cathode and anode signals change respectively.

**SLIDING LEDs**

**Design Source**

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

**(**

**input** clk**,**

**input** rst**,**

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

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

**);**

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

**reg** **[**15**:**0**]** led\_reg**;**

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

**if(**rst**==**1'b1**)** **begin**

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

led\_reg **<=** 16'b0000000000000001**;**

**end**

**else** **begin**

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

2'b00**:** led\_reg **<=** led\_reg**;**

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

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

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

**if(**led\_reg **==** 16'b1000000000000000**)** **begin**

led\_reg **<=** 16'b0000000000000001**;**

**end**

**else** **begin**

led\_reg **<=** led\_reg **<<** 1**;**

**end**

**end**

**else** **begin**

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

**end**

**end**

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

**if(**counter **==** **(**CLK\_DIV**/**2**)** **-**1**)** **begin**

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

**if(**led\_reg **==** 16'b1000000000000000**)** **begin**

led\_reg **<=** 16'b0000000000000001**;**

**end**

**else** **begin**

led\_reg **<=** led\_reg**<<**1**;**

**end**

**end**

**else** **begin**

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

**end**

**end**

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

**if(**counter **==** **(**CLK\_DIV**/**5**)** **-**1**)** **begin**

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

**if(**led\_reg **==** 16'b1000000000000000**)** **begin**

led\_reg **<=** 16'b0000000000000001**;**

**end**

**else** **begin**

led\_reg **<=** led\_reg**<<**1**;**

**end**

**end**

**else** **begin**

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

**end**

**end**

**default:** led\_reg **<=** led\_reg**;**

**endcase**

**end**

**end**

**assign** LED **=** led\_reg**;**

**endmodule**

**Simulation Source**

**module** sliding\_leds\_tb**();**

**parameter** **[**23**:**0**]** CLK\_DIV **=** 10000000**;**

**reg** CLK **=** 1'b0**;**

**reg** RST**=**1'b1**;**

**reg** **[**1**:**0**]** SW **=** 2'b00**;**

**wire** **[**15**:**0**]** LED**;**

sliding\_leds **#(.**CLK\_DIV**(**CLK\_DIV**))** uut

**(**

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

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

**.**SW**(**SW**),**

**.**LED**(**LED**)**

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;** // 100MHz

**end**

**initial**

**begin**

RST **=** 1**;**

**#**10**;**

RST **=** 0**;**

**#**10**;**

SW **=** 2'b01**;**

**#**400000000**;**

SW **=** 2'b10**;**

**#**100000000**;**

SW **=** 2'b11**;**

**#**100000000**;**

$finish**;**

**end**

**endmodule**

**ekran görüntüsü, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturulduSimulation Wave**

As seen in the simulation, the module works as it should. When SW = 01, we have 10 Hz frequency and 0.1 second period. Therefore, we have four logic 1 cycle which have 0.1 second period. When SW = 10, we have 20 Hz frequency and 0.05 second period. Therefore, we have two logic 1 cycle which have 0.05 second period. When SW = 11, we have 50 Hz frequency and 0.02 second period. Therefore, we have thirty four logic 1 cycle which have 0.1 second period

Post-Implementation Timing Simulation

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**Timing Report**

This photo shows setup and hold times for inputs. SW[1] have the worst case for setup/hold time compared with SW[0].

metin, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

This photo shows max/min delays for outputs. LED[15] have the worst case in terms of delays. Therefore, the path which is connected to LED[15] is the critical path.

**RESEARCH**

Timing analysis is one of the most significant part of digital design. Designers can add the project timing constraints. Static Timing Analysis (STA) verify all paths meet the timing constraints and required performance in terms of delays. It calculates delays for all paths and verifies whether constraints are met.

Critical Path: The critical path is the longest path that a signal can take from one flip- flop to another in a synchronous circuit. It is the worst case in terms of delays.

Clock Periods: The duration of one clock cycle, which determines the maximum speed at which the FPGA can operate.

Input/Output Delays: The time taken for signals to propagate from input pins to internal logic and from internal logic to output pins.

Setup/Hold Times: Setup time is the minimum time before the clock edge that data must be stable, while hold time is the minimum time after the clock edge that data must remain stable. ***[2]***

1. In digital design, flip-flops have a specified set-up time and hold time. The minimum time before the clocking activity by which the input signal must be stable is called set-up time. The minimum time after the clocking activity, during which the input signal must remain stable, is called hold time. The input should not change during the setup and hold period for stable system. On the contrary, if input changes during setup and hold time, outputs have unknown results. This is called a metastable state. Metastability is the propagation of the metastable state. ***[3]***

A synchronizer typically consists of multiple cascaded flip-flops. This structure allows the signals to be stabilized before the clock signal. This works like buffer.

Increasing setup and hold time can solve metastability problem. However, it can increase period, decrease frequency and it can affect performance of system.

Being able to provide input and clock signals reaching different paths synchronously can also solve the metastability problem. ***[4]***

taslak, çizim, çizgi sanatı, kırpıntı çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

***[4]***

**REFERENCES**

[1] GeeksforGeeks. (2024, May 20). *Latches in digital logic*. GeeksforGeeks. <https://www.geeksforgeeks.org/latches-in-digital-logic/>

[2] Harvie, L. (2024, July 31). How to perform Static Timing Analysis (STA). *RunTime Recruitment*. <https://runtimerec.com/how-to-perform-static-timing-analysis/>

[3] Cadence. (2024, August 12). *How to avoid metastability in digital circuits*. Advanced PCB Design Blog | Cadence. <https://resources.pcb.cadence.com/blog/2022-how-to-avoid-metastability-in-digital-circuits>

[4] Pandey, P. (2023, June 14). *(7) Unveiling metastability in VLSI: Taming the Unpredictable Beast! | LinkedIn*. <https://www.linkedin.com/pulse/unveiling-metastability-vlsi-taming-unpredictable-beast-priya-pandey/>