GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x6 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersi

Hazırlayanlar
1) 171024086-Berat KIZILARMUT
2) 1801022037-Ömer Emre POLAT

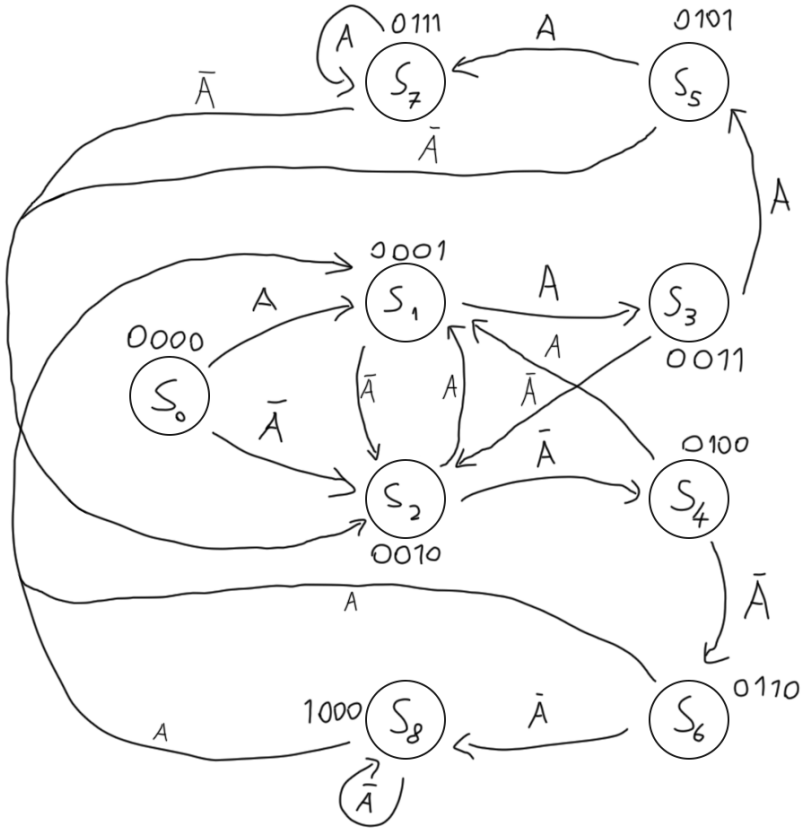
1. Giriş

Bu deneyde State Transition Diagram oluşturacağız ve analiz edeceğiz. State Transition Diagram'larını oluşturacağımız devreler Finite State Machine'ler olacak. Bu oluşturulan Finite State Machine'ler, SystemVerilog diline çevirerek devreler sentezleyeceğiz.

2. Problemler

2.1. Problem I – Pattern Yakalayıcı

A. State Transition Diagram



Bu devre için oluşturmuş olduğumuz State Transition Diagram, Moore Finite State Machine türünde olup, iç döngülere sahiptir.

Toplam 9 State'ten oluştuğu için, 4 register kullanılması gerekmektedir.

Her bir state için binary encoding yapılmıştır.

B. HDL Yazılımı

Bir önceki adımda paylaştığımız State Transition Diagram, SystemVerilog diline geçirilmiştir. Öncelikle typedef enum tanımıyla stateler tanımlandı. Ardından sürekli geçerli olacak veri akışını sağlamak için always_ff bloğu ile her clockta, bir sonraki state'in şuanki state'e atanması sağlanıldı. Bu işlemden sonra ise combinational logic işlemleri için always_comb bloğu içine oluşturmuş olduğumuz State Transition Diagram takip edilerek statelerin ilişkileri oluşturuldu. Latch oluşmaması için always_comb'unun içine default durum yazıldı.

```
/* lab6_g41_p1.sv
 * Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
 * Notlar: Pattern Yakalayıcı */
module lab6_g41_p1(
    input logic a, clk,
    output logic y );

typedef enum {S0, S1, S2, S3, S4, S5, S6, S7, S8} statetype;
statetype state, nextstate;
always_ff @(posedge clk)
 begin
    state <= nextstate;
 end

always_comb
 begin
    case(state)
        S0:
        begin
            if(a)
                nextstate = S1;
            else
                nextstate = S2;
        end
        S1:
        begin
            if(a)
                nextstate = S3;
            else
                nextstate = S2;
        end
        S2:
        begin
            if(!a)
                nextstate = S4;
            else
                nextstate = S1;
        end
        S3:
        begin
            if(a)
                nextstate = S5;
            else
                nextstate = S2;
        end
        S4:
        begin
            if(!a)
                nextstate = S6;
            else
                nextstate = S1;
        end
        S5:
        begin
            if(a)
                nextstate = S7;
            else
                nextstate = S2;
        end
        S6:
        begin
            if(!a)
                nextstate = S8;
            else
                nextstate = S1;
        end
        S7:
        begin
            if(a)
                nextstate = S7;
            else
                nextstate = S2;
        end
        S8:
        begin
            if(!a)
                nextstate = S8;
            else
                nextstate = S1;
        end
        default: nextstate = S0;
    endcase

    case(state)
        S7:
        begin
            y = 1;
        end
        S8:
        begin
            y = 1;
        end
        default: y = 0;
    endcase
 end
endmodule
```

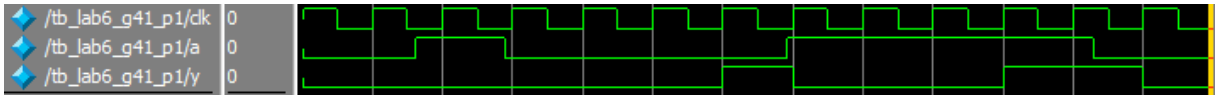
C. Testbench Kurulumu

Kurulan devreyi sınamak için duruma ve devreye uygun bir testbench devresi kuruldu. Devre explicit instantiation şekilde çağırıldı. 10 ps periyoda sahip bir Clock sinyali tanımlandı. Farklı kombinasyonlarda inputlar sınandı. Testbench devreyi istenildiği gibi doğru bir şekilde sınadı.

```
/* tb_lab6_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Pattern Yakalayıcı testbenchi */
`timescale 1ns/1ps
module tb_lab6_g41_p1();
logic a, clk, y;
lab6_g41_p1 dut1(.a(a), .clk(clk), .y(y));
always
 begin
    clk = 1; #5;
    clk = 0; #5;
 end
initial
 begin
    a = 0; #16;
    a = 1; #13;
    a = 0; #40;
    a = 1; #44;
    a = 0; #17;
    $stop;
 end
endmodule
```

D. Waveshape

Testbench sonuçlarını gösteren dalga formu. A sinyalinde gecikmeler gerçekleşerek oluştu. Devre beklenildiği sonuçları verdi. A girişi 4 kere 0 olduğunda Y çıkışı 1 olmuştur. A girişi 4 clock boyu 1 olunca Y çıkışı 1 vermiştir. A girişi 1 olmaya devam edince tekrar 1 vermiştir.

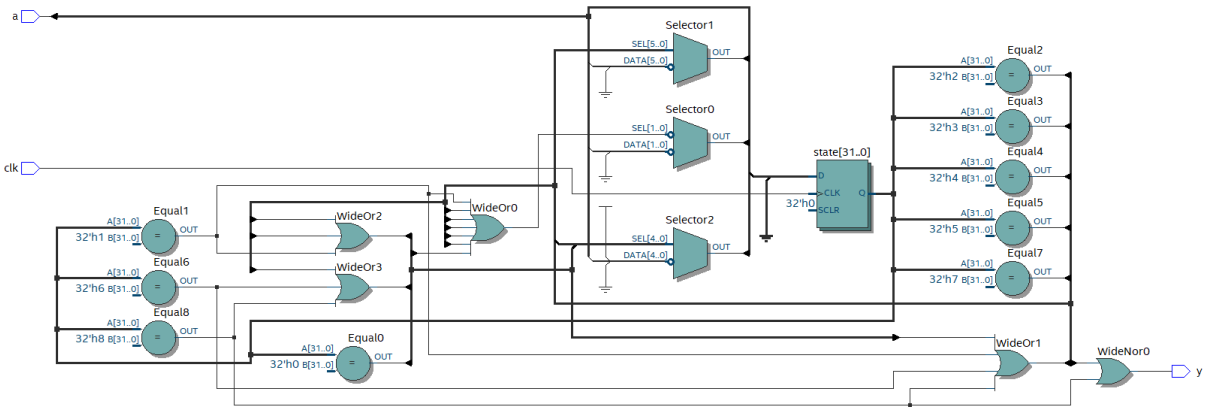


E. Problem I Analiz ve Yorumlar

SystemVerilog yazılımımız beklenildiği gibi çalışmıştır. Devrede, A sinyali dört clock boyunca 0 kaldığında Y çıkışı 1 olmuştur. A sinyali dört clock boyunca 1 kaldığında da Y çıkışı 1 olmuştur. Bu devrenin kurulumunda 4 logic register kullanılmıştır. Devrede 7 ALUT kullanılmıştır. Devremizin Fmax'i 574.41 MHz çıkmıştır. Restricted Fmax'imi 250 MHz gözükmektedir, bunun sebebi devrenin sentezlendiği kartın input output değiştirme sıklığından dolayı limitlenmesidir.

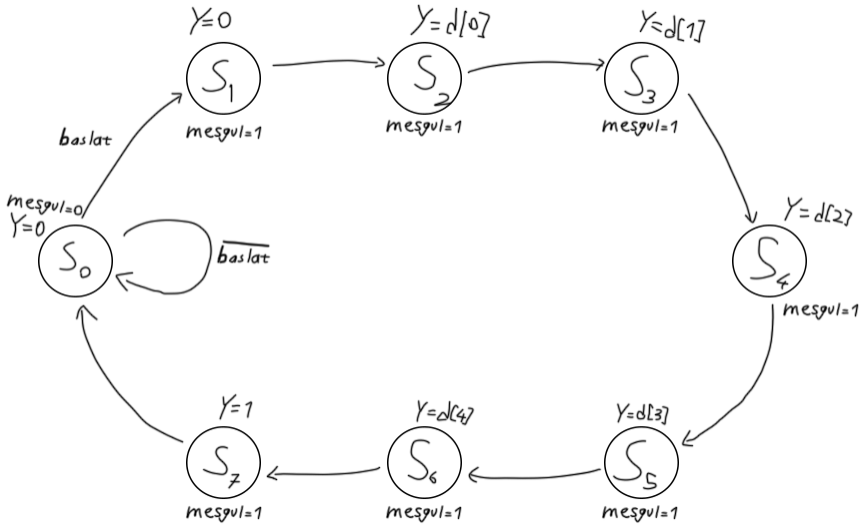
Fmax	Restricted Fmax	Clock Name	Note
574.71 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|lab6_g41_p1	7 (7)	4 (4)	0	0	0	0	0	3



2.2 Problem II – Sıralayıcı

A. State Transition Diagram



İstenilen görevleri gerçekleştiren State Transition Diagram çizilmiştir. Bir önceki örnekteki gibi, bu devre de bir Moore Finite State Machine devresi olacaktır. Devrede 8 state bulunduğu için 3 register yeterli olmaktadır.

B. HDL Yazılımı

Problem 1'de yaptığımız gibi, bir önceki adımda paylaştığımız State Transition Diagram, SystemVerilog diline geçirildi. Başlangıç olarak, logic input output tanımlarının ardından, typedef enum tanımıyla stateler tanımlandı. Ardından sürekli geçerli olacak veri akışını sağlamak için always_ff bloğu ile her clockta, bir sonraki state'in şuanki state'e atanması sağlanıldı. Active low reset tanımlandı. Bu işlemden sonra ise combinational logic işlemleri için always_comb bloğu içine oluşturmuş olduğumuz State Transition Diagram takip edilerek statelerin ilişkileri oluşturuldu. Latch oluşmaması için always_comb'unun içine default durum yazıldı.

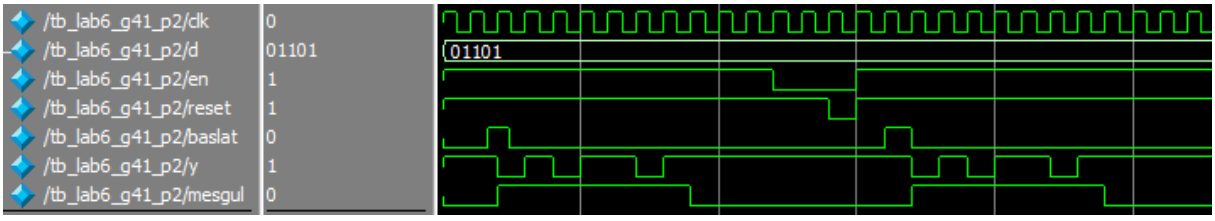
```
/* lab6_g41_p2.sv
 * Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
 * Notlar: Sıralayıcı */
module lab6_g41_p2 (
input logic clk, reset, en,
input logic [4:0] d,
input logic baslat,
output logic y,
output logic meşgul );
typedef enum {S0, S1, S2, S3, S4, S5, S6, S7} statetype;
statetype state, nextstate;
always_ff @(posedge clk)
begin
    if(!reset)
        state <= S0;
    else
    if(en)
        state <= nextstate;
end
always_comb
begin
    case(state)
    S0:
    begin
        y = 1;
        mesgul = 0;
        if(baslat)
            nextstate = S1;
        else
            nextstate = S0;
    end
    S1:
    begin
      mesgul = 1;
        nextstate = S2;
        y = 0;
    end
    S2:
    begin
        mesgul = 1;
        nextstate = S3;
        y = d[0];
    end
    S3:
    begin
        mesgul = 1;
        nextstate = S4;
        y = d[1];
    end
    S4:
    begin
        mesgul = 1;
        nextstate = S5;
        y = d[2];
    end
    S5:
    begin
        mesgul = 1;
        nextstate = S6;
        y = d[3];
    end
    S6:
    begin
        mesgul = 1;
        nextstate = S7;
        y = d[4];
    end
    S7:
    begin
        mesgul = 1;
        nextstate = S0;
        y = 1;
    end
    default: nextstate = S0;
    endcase
end
endmodule
```

C. Testbench Kurulumu

Her zaman yaptığımız gibi devremizi sınamak için devreye özel bir testbench kurduk. Herhangi bir problem yaşanmaması için devre explicit instantiation şekilde çağırıldı. Periyodu 10 ps olan bir clock tanımlandı. Inputlar değiştirelerek sınandı.

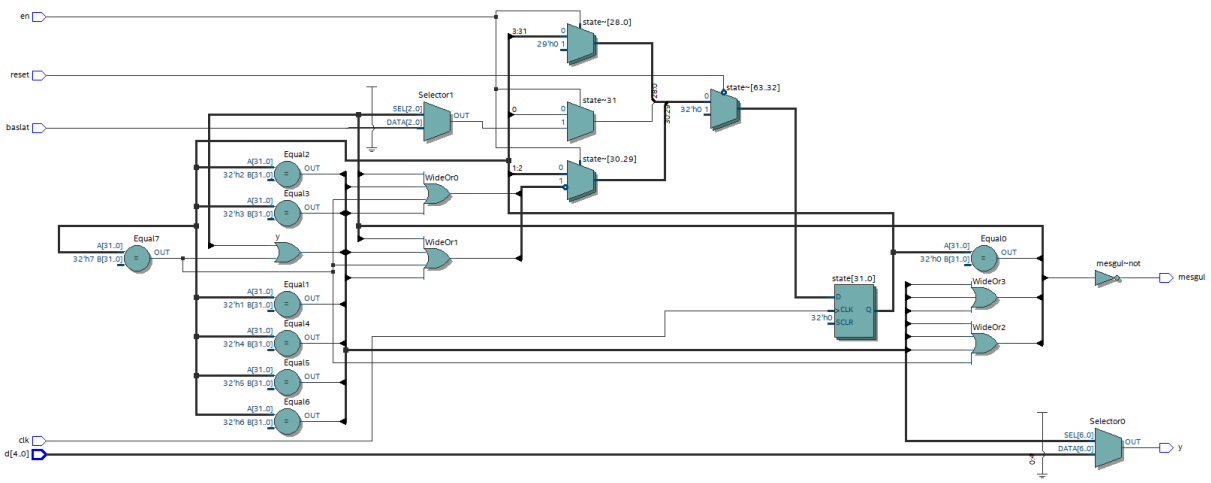
```
/* tb_lab6_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Sıralayıcı testbenchi */
`timescale 1ns/1ps

module tb_lab6_g41_p2();

logic clk, baslat, y, mesgul, en, reset;
logic [4:0] d;

lab6_g41_p2 dut1(.clk(clk), .d(d), .baslat(baslat), .y(y), .mesgul(mesgul), .en(en), .res
et(reset));

always
begin
    clk = 1; #5;
    clk = 0; #5;
end

initial
begin
    en = 1;
    reset = 1;
    d = 5'b01101;
    baslat = 0; #16;
    baslat = 1; #8;
    baslat = 0; #96;
    en = 0; #20;
    reset = 0; #10;
    reset = 1; en = 1; #10;
    baslat = 1; #10;
    baslat = 0; #10;
    #100;
    $stop;
end
endmodule
```

D. Waveshape



Dalga formunda da gözlendiği gibi, devre başlat girişi 1 olana kadar herhangi bir işlem yapmamıştır. Başlat girişi 1 olduğu anda Y çıkışı 0 vermiştir. Least significant bit'den başlayarak girilen sayısı çıkış olarak vermiştir. D sinyali paralel okunup seri bir şekilde çıkıştan yollandıktan sonra bitiş sinyali olarak Y çıkışı 1 vermiştir. Dalga formunda enable ve reset sinyallerinin de doğru çalışıp çalışmadığı test edilmiştir.

E. Problem II Analiz ve Yorumlar



SystemVerilog yazılımımız beklenildiği gibi çalışmıştır ve istenilen fonksiyonları yerine gerçekleştirmiştir. Devre enable kapalı iken çalışmamıştır. Reset sıfırlandığında işlem başa dönmüştür. Y çıkışı gerektiği gibi, başa sıfır, sona 1 ekleyerek, giriş sayısını least significant bit'ten, most significant bit'e doğru sırayla vermiştir. Bu devrenin kurulumunda 3 logic register kullanılmıştır. Devrede 10 ALUT kullanılmıştır. Devremizin Fmax'i 595.59 MHz çıkmıştır.

	Fmax	Restricted Fmax	Clock Name	Note
1	595.59 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|lab6_g41_p2	10 (10)	3 (3)	0	0	0	0	0	11

3. Genel Yorumlar ve Kazanımlar

Bu deneyde öncelikle istenilen yönergelere göre bir State Transition Diagram oluşturmayı öğrendik. Ardından bu oluşturduğumuz State Transition Diagram'ı, Finite State Machine'lere çevirmeyi öğrendik.

4. Kaynaklar

1. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015