TOBB Ekonomi ve Teknoloji Üniversitesi

BİL 265/264-Mantıksal Devre Tasarımı

Proje Raporu

Adı Soyadı	Tutku Ekin Canpolat Mustafa Ege Atay Can Hazırol
Numara	201201078 201201043 171201020
Tarih	18.12.22

İÇİNDEKİLER

1. GİRİŞ

1.1 Projenin Amacı

Projemizin amacı, FPGA kullanarak derste öğrendiğimiz bilgilerimizi uygulamak, gerçek hayatta karşılığını görmek ve basit bir UART haberleşme protokolü çalışma mantığını öğrenmemizdir.

1.2 İş Bölümü

- Tutku Ekin Canpolat
  - ✓ Ripple Carry Adder kodlanması ve rapora yazımı
  - ✓ Behavioral Adder raporda yazımı
  - ✓ Carry-select adder raporda yazımı
  - ✓ Testbench yazımı
  - ✓ Constrain yazımı

- Mustafa Ege Atay
  - ✓ Carry Look-Ahead kodlanması ve rapora yazımı
  - ✓ Time Analysis bulunması
  - ✓ Ana modül kodlanma süreci

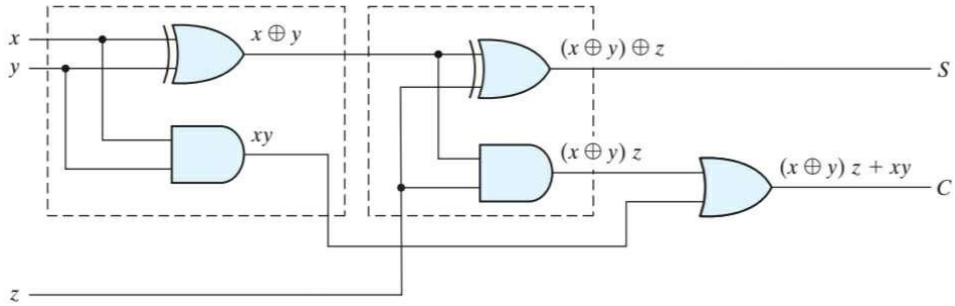
- Can Hazırol
  - ✓ Carry-select adder kodlanması ve raporda yazımı
  - ✓ Behavioral Adder kodlanması
  - ✓ Ana modül kodlanması ve rapora yazımı

2. KODLAR

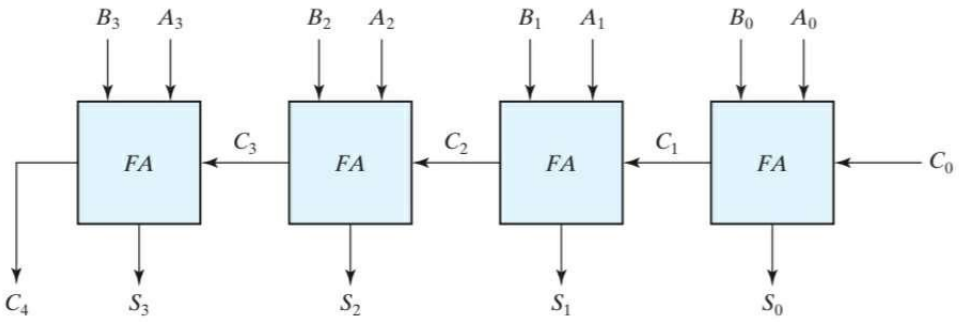
2.1 Ripple-Carry Adder

2.1.1 Çalışma Prensibi

Ripple-Carry Adder, mantığında toplama sonucu gelen eldenin sonraki bloktaki toplama işlemine verilmesini içerir. Kitaptan aldığım şemalara göre;



şeklinde bir full-adder ile gelen elde işleme sokulur ve sonraki işlem için sonuçtaki elde hazırlanır. Full-Adder bloklarının birleşmesiyle işe aşağıda görülen Ripple Carry Adder şeması oluşur.



2.1.2 Yazılış Mantığı ve Süreci

Öncelikle burada yapmamız gereken şey bir full-adder bloğunda teker teker elde çıkarmak ve sonraki değerde bu eldeyi kullanmak. Basitçe gösterirsek;

```
sout[0] = a[0] ^ b[0] ^ cin_next[0];
cin_next[1] = (a[0] & b[0]) | ((a[0] | b[0]) & cin_next[0]);
```

İşlemini her bit için uygulamamız gereklidir.

Başta, bir for bloğu ile ilerlemeyi düşündük ancak for bloğunun yaratabileceği sıkıntılardan kaçmak için her biti taker taker atama kararı aldık. Bu kararımda, hem elle yazmanın yaratabileceği dikkatsizlik hem de vakit kaybından kaçınmak adına, C dilinde

```c
#include <stdio.h>
#include <math.h>

int main() {
    int i;

    for (int i = 0; i < 64; ++i) {

        printf( _Format: "sout[%d] = a[%d] ^ b[%d] ^ cin_next[%d];\n", i, i, i, i);
        printf( _Format: "cin_next[%d] = (a[%d] & b[%d]) | ((a[%d] | b[%d]) & cin_next[%d]);\n", i + 1, i, i, i, i, i);
        printf( _Format: "\n");

    }

    return 0;
}
```

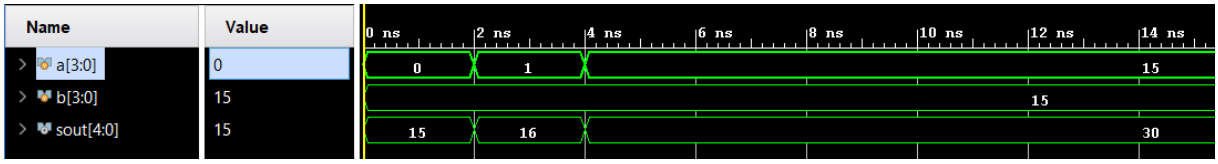
kodunu yazdık, bu sayede her basamak için sonucumuzu aldık.

2.1.3 Kod ve Testbench

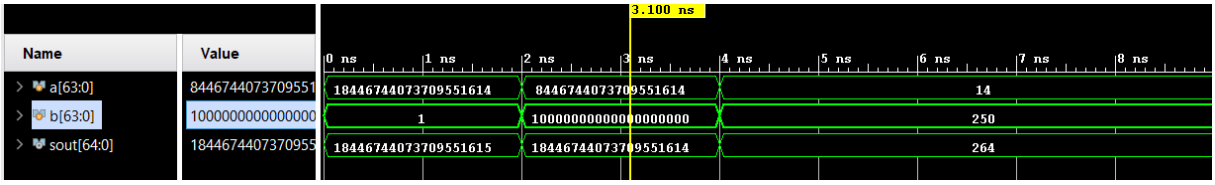
Kodumuzu en başta daha küçük bir modül halinde yazdık, bu sayede çalışma hatası olup olmadığını yakalama şansı bulduk.

```
module adder_2(
    input [3:0]a,
    input [3:0]b,
    output reg [4:0]sout
);
    reg [3:0]cin_next=4'b0000;


    always@(*)begin
        sout[0] = a[0] ^ b[0] ^ 0;
        cin_next[1] = (a[0] & b[0]) | ((a[0] | b[0]) & cin_next[0]);

        sout[1] = a[1] ^ b[1] ^ cin_next[1];
        cin_next[2] = (a[1] & b[1]) | ((a[1] | b[1]) & cin_next[1]);

        sout[2] = a[2] ^ b[2] ^ cin_next[2];
        cin_next[3] = (a[2] & b[2]) | ((a[2] | b[2]) & cin_next[2]);

        sout[3] = a[3] ^ b[3] ^ cin_next[3];
        sout[4] = (a[3] & b[3]) | ((a[3] | b[3]) & cin_next[3]);


    end
endmodule
```

Sonrasında testbenche soktuğumda;



Doğru sonucu aldık. Sonrasında kodu 64 bit için tamamladığımızda, yeniden testbenchte denedik. Bunu yaparken de 2^64 değerini araştırıp, girişlerimi ona uygun değerler verdik.



Şeklinde doğru sonuca ulaştık.

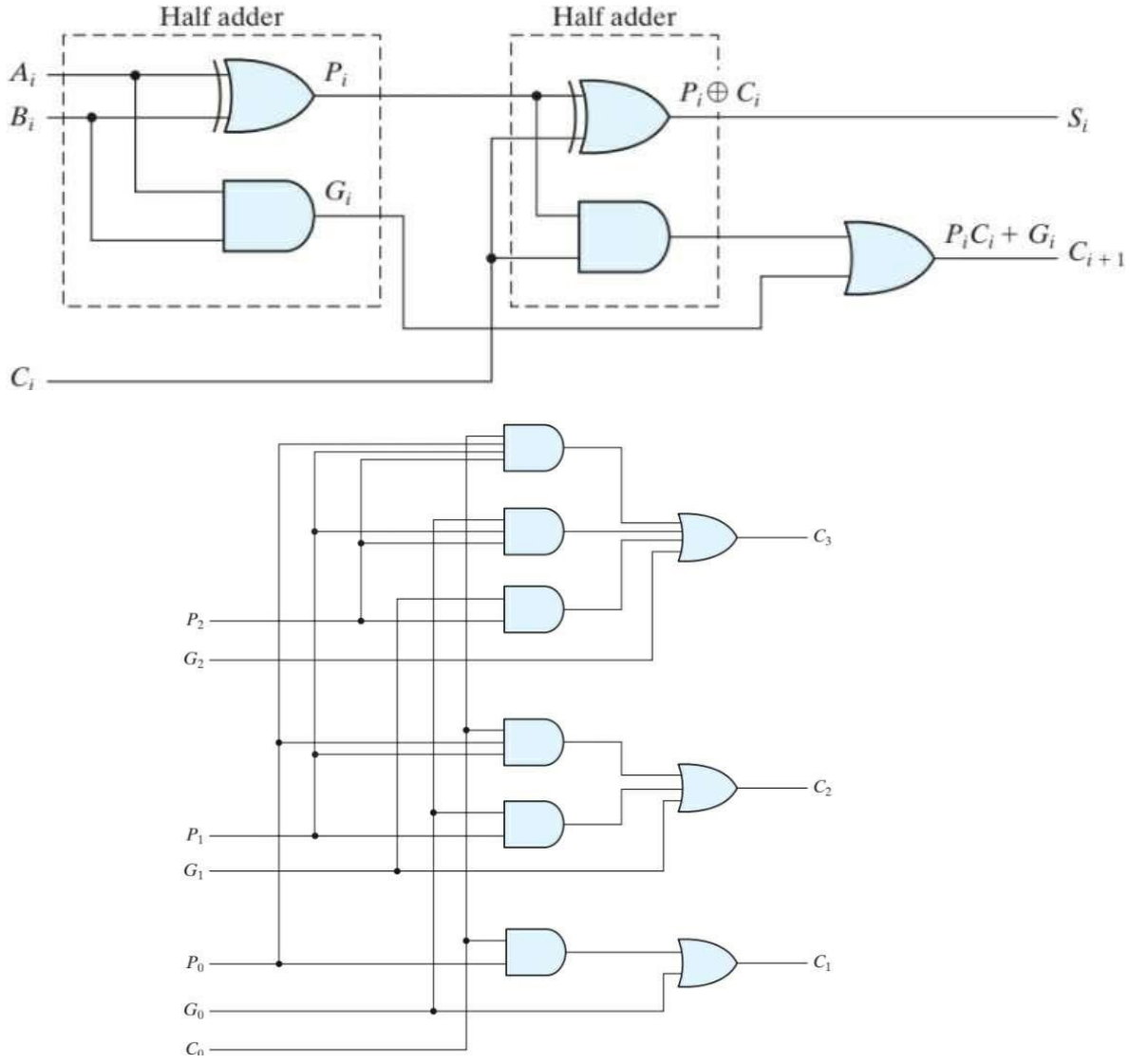
Tüm adder modüllerimizi aşağıdaki testbench'e sokuyoruz.

```
module tb_adder1(
);
    reg [63:0]a;
    reg [63:0]b;
    wire [64:0]sout;
    adder1 uut(a,b,sout);
    initial begin
    a=64'd18446744073709551614;
    b=64'd1;

    #2
    a=64'd084467440737095551614;
    b=64'd10000000000000000000;

    #2
    a=64'd184;
    b=64'd1256; //beklenen toplam 1440

    #2
    a=64'd14;
    b=64'd7;

    #2
    a=64'd156596564;
    b=64'd125556; //beklenen toplam 156722120

    end
endmodule
```

2.2 Carry Look Ahead

2.2.1 Çalışma Prensibi

Carry look ahead adder, eldelerin(C) önceden hesaplanıp eldelerden hesaplanan generate(G) ve propagate(P) değişkenlerinin hesaplanmasıyla toplam değerine ulaşılır. Buradaki amaç eldelerin önceden hesaplanıp zaman kaybının azaltılmasıdır.



Propagate(P), girişlerin bitwise xor işleminden geçirilmesiyle oluşmuştur. Generate(G) ise girişlerin bitwise and işleminden geçirilmesiyle oluşmuştur. n. carry(C) hesabı için ise n-1.

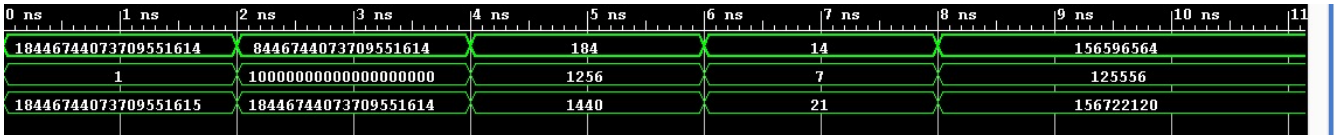
propagate ve n-1. Carry and işleminden geçirilip n-1. generate ile or işlemine sokulur ve sonuca ulaşılır.

2.2.2 Yazılış Mantığı ve Süreci

Kodun yazılışı için çalışma prensibinde tanımladığımız değişkenleri hesapladık ve gerekli toplam bitlerine yazdık.

```
module cla(
input [63:0]a_i,
input [63:0]b_i,

output [64:0]sum_o

);

reg [64:0]carry=65'd0;
reg [63:0]sum,g,p;


integer i;



always@(*)begin


            g[63:0] = a_i[63:0] & b_i[63:0];

            p[63:0] = a_i[63:0] ^ b_i[63:0];


            for (i=0;i<64;i=i+1)begin
            carry[i+1] = g[i] | p[i] & carry[i] ;

           end


            sum[63:0]  = carry[63:0] ^ p[63:0];


end


assign sum_o = {carry[64],sum[63:0]};

endmodule
```

2.2.3 Kod ve Testbench



2.3 Behavioral Adder

2.3.1 Çalışma Prensibi

Bu toplayıcı türünde, elde yok sayılarak her bitin karşılık geldiği bitle toplanması ile sonuç elde edilir.

2.3.2 Yazılış Mantığı ve Süreci

Bu kodu yazarken, + operatöründen faydalandık.

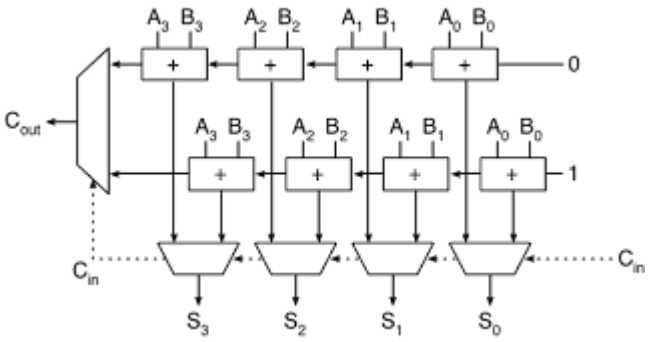
2.3.3 Kod ve Testbench

```
//behavioral adder
module adder4(
input [63:0] a_i,
input [63:0] b_i,
output reg [64:0]sum_o // taşma durumunu dikkate almadım.
    );
initial begin
sum_o[64]=1'b0;
end
always@* begin
        sum_o = a_i + b_i;
    end

endmodule
```

2.4 CARRY SELECT ADDER

2.4.1 Çalışma Prensibi

Bu tip toplayıcılar, ripple carry adder ve multiplexerlardan oluşmaktadır. Her iki muhtemel carry girişi için çıkışları önceden hesaplamaktadır, bu sayede süreci hızlandırmaktadır. En sonda ise mux kullanarak girilen degree uygun doğru çıkışı almamızı sağlamaktadır. Aşağıda, şeması gösterilmiştir.



2.4.2 Yazılış Mantığı ve Süreci

Öncelikle bunu yazarken full adder ve multiplexer larımızı bu modül içerisinde bir modül olarak tanımladık.

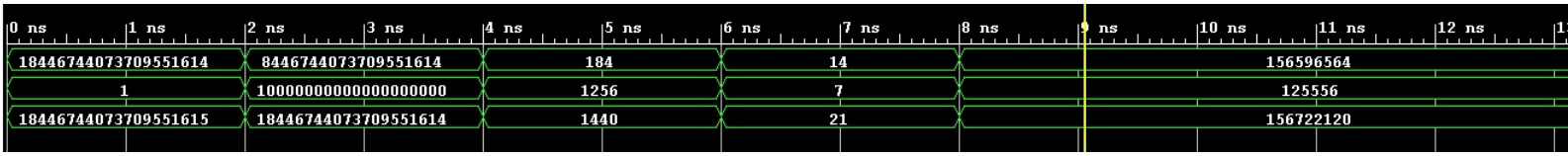
```
44  module multiplexer2
45          (   input i0,i1,sel,
46              output reg bitout
47              );
48  always@(i0,i1,sel)
49  begin
50  if(sel == 0)
51      bitout = i0;
52  else
53      bitout = i1;
54  end
55  endmodule
56
57
58  module fulladder
59          (   input a,b,cin,
60              output sum,carry
61              );
62  assign sum = a ^ b ^ cin;
63  assign carry = (a & b) | (cin & b) | (a & cin);
64  endmodule
```

Sonrasında, yukarıdaki şemada da gösterildiği sırasıyla bloklara girmesi için bir for döngüsü oluşturduk, bu sayede her bir değerin ataması gerçekleşti.

```
16  //for carry 0
17  fulladder f0(a_i[0],b_i[0],1'b0,temp0[0],carry0[0]);
18  genvar i;
19  generate
20  for (i=1; i<64; i=i+1) begin
21  fulladder (a_i[i],b_i[i],carry0[i-1],temp0[i],carry0[i]);
22  end
23  endgenerate
24
25  //for carry 1
26  fulladder f1(a_i[0],b_i[0],1'b1,temp1[0],carry1[0]);
27  genvar j;
28  generate
29  for (j=1; j<64; j=j+1) begin
30  fulladder (b_i[j],a_i[j],carry1[j-1],temp1[j],carry1[j]);
31  end
32  endgenerate
```

Sonrasında hem sum'ları hem de en son çıkan carry'leri multiplexer'a sokarak sonucumuzu elde ediyoruz.

2.4.3 Kod ve Testbench



2.5 ANA MODÜL

```
reg rx;
reg [151:0]paket1; //PC'den FPGA kartına gönderilecek olan paket içeriği
reg [87:0]paket2; //FPGA kartından PC'ye toplama sonucunu gönderecek olan paket içeriği
reg [63:0]sayi1,sayi2;
reg [15:0]baslik1;
reg [15:0]baslik2;
reg [7:0]checksum1;
reg [7:0]checksum2[0:3];
reg [65:0]temp_toplam;
reg [65:0]temp_toplam2[0:3];
reg [64:0]sonuc1,sonuc2,sonuc3,sonuc4;
reg btnl,btnu,btnr,btnd;

wire [63:0]sonuc1w,sonuc2w,sonuc3w,sonuc4w;
```

Proje pdf'inde belirtilmiş modül giriş çıkışlarının ardından yapılan reg ve wire tanımlamaları bu şekildedir. Sequential block içinde buton girişlerine göre değeri değişen tanımlamalar ve aktif-düşük reset sinyali ile değerinin sıfırlanması gereken yazmaçlar bu şekilde belirtilmiştir.

Combinational kısımda, seçilen buton aracılığıyla sonucu dönülecek adder'ın oluşturduğu checksum değerlerinin hesabında kullanılmak amacıyla veri kaybı yaşanmama koşulu gözetilerek yazmaç bit değerleri hesaplanmış ve oluşturulan arrayler işlemlerde kullanılmıştır.

Wire'ların oluşturulma sebebi, ana modüle çağırılan adderların çıkışlarını tutmasıdır. Bu wire değerleri ana modülde aynı bit sayılarıyla oluşturulmuş yazmaç değerlerine kaydedilmiştir.

```
initial begin //SAYI GİRİŞLERİNİ YAPACAĞIMIZ KISIM
paket1[79:16] = 64'd12;  // SAYI1
paket1[143:80] = 64'd13; // SAYI2
end

adder1 a(sayi1,sayi2,sonuc1w);
adder2 b(sayi1,sayi2,sonuc2w);
adder3 c(sayi1,sayi2,sonuc3w);
adder4 d(sayi1,sayi2,sonuc4w);
```

Bu kısımda PC'den yapılacak sayı girişlerine initial bloğu ile değer atanmaktadır. Always bloğuna giriş yapmadan adderlar çağırılmış toplama işlemleri gerçekleştirilmiştir. Sonuçlar, yukarıda oluşturulmuş wire değerlerine atanmıştır.

```
40  always @* begin
41
42      sonuc1 = sonuc1w;
43      sonuc2 = sonuc2w;
44      sonuc3 = sonuc3w;
45      sonuc4 = sonuc4w;
46
47      paket1[15:0] = 16'hBACD; //PC'den FPGA kartına gönderilecek olan paket başlığı
48      baslik1 = paket1[15:0];
49      sayi1 = paket1[79:16];
50      sayi2 = paket1[143:80];
51
52      paket2[15:0] = 16'hBAFD; // FPGA kartından PC'ye toplama sonucunu gönderecek olan paket
53      baslik2 = paket2[15:0];
```

Çağırılan modüllerin wire'lara atanan sonuçları always bloğu içinde reg değerlerine atanmıştır. Sebebi, combinational ve sequential always blokları içinde yapılacak atamalarda her iki tarafta da kullanılacak olmalarıdır. PC'den FPGA'e gönderilecek paketin de, FPGA'den PC'ye gelecek sonuç içerikli paketin de içerikleri belirtilen bölgelere ayrılırken MSB gözetilmemiştir. Zaten hangi bölümün ne olduğu belirtildiğinden ihtiyaç duyulmamıştır.

```
55      checksum1 = paket1[151:144];
56      temp_toplam = baslik1 + sayi1 + sayi2;
57      checksum1 = temp_toplam % 256;
58
59      temp_toplam2[0] = baslik2 + sonuc1;
60      temp_toplam2[1] = baslik2 + sonuc2;
61      temp_toplam2[2] = baslik2 + sonuc3;
62      temp_toplam2[3] = baslik2 + sonuc4;
63
64      checksum2[0] = temp_toplam2[0] % 256;
65      checksum2[1] = temp_toplam2[1] % 256;
66      checksum2[2] = temp_toplam2[2] % 256;
67      checksum2[3] = temp_toplam2[3] % 256;
68
69  end
```

PC'den FPGA gönderilecek paketin Checksum değeri clk sinyaline bağlı olarak değişen bir değer olmadığı için combinational kısımda hesaplanmıştır. FPGA'den PC'ye dönecek toplam sonucu içeren paketin checksum değerleri ise sequential block'ta seçilen butona göre dönülecektir, bu sebeple burada bir array'e kaydedilmiştir. Bu bölüm ile Combinational Always bloğu sonlanmıştır.

```
71  always @(posedge clk) begin
72      if (rst_n==1'b0) begin
73          rx <= 1'b0;
74          tx_o <= 1'b0;
75          btnl <= 1'b0;
76          btnu <= 1'b0;
77          btnr <= 1'b0;
78          btnd <= 1'b0;
79          sayi1 <= {64{1'b0}};
80          sayi2 <= {64{1'b0}};
81          sonuc1 <= {65{1'b0}};
82          sonuc2 <= {65{1'b0}};
83          sonuc3 <= {65{1'b0}};
84          sonuc4 <= {65{1'b0}};
85      end
```

Bu kısımda aktif – düşük reset sinyali ile yukarıdaki yazmaçlara belirtilen değerler atanmıştır.

```
86      else begin
87          btnl <= btnl_i;
88          btnu <= btnu_i;
89          btnr <= btnr_i;
90          btnd <= btnd_i;
91          rx <= rx_i;
```

Buton ve rx'ler için oluşturulmuş flip-floplar.

```
92      //pc'den fpga'e giden paketin başlığı bu ise,
93      //sayı1 ve sayı2 toplam sonuçlarını btn inputlarına göre verebiliyoruz.
94        if((baslik1 == 16'hBACD) && (rx == 1'b1)) begin
95           if(btnl == 1'b1)begin
96               paket2[15:0] <= baslik2;
97               paket2[79:16] <= sonuc1;
98               paket2[87:80] <= checksum2[0];
99               //deneme <= paket2;
100              tx_o <= 1'b1;
101          end
102          else if (btnu == 1'b1) begin
103              paket2[15:0] <= baslik2;
104              paket2[79:16] <= sonuc2;
105              paket2[87:80] <= checksum2[1];
106              //deneme <= paket2;
107              tx_o <= 1'b1;
108          end
109          else if (btnr == 1'b1) begin
110              paket2[15:0] <= baslik2;
111              paket2[79:16] <= sonuc3;
112              paket2[87:80] <= checksum2[2];
113              //deneme <= paket2;
114              tx_o <= 1'b1;
115          end
116          else if (btnd == 1'b1) begin
117              paket2[15:0] <= baslik2;
118              paket2[79:16] <= sonuc4;
119              paket2[87:80] <= checksum2[3];
120              //deneme <= paket2;
121              tx_o <= 1'b1;
122          end
```

PC'den FPGA'e gönderilen paketin başlığı belirtildiği gibiyse ve recieve sinyali aktif ise, FPGA'den gelen buton inputuna göre combinational kısımda hesaplanmış başlık bilgisi, buton input'unun temsil ettiği adder'dan gelen sonuç ve yine combinational kısımda hesaplanmış checksum sonucu çıkış olarak FPGA'den PC'ye gönderilen sonuç paketine atanmıştır. tx_o çıkışı da 1 yapılmıştır. Her seferinde yalnızca bir buton girişinin 1 olacağı kabul edilmiştir.

2.5.2 ANA MODÜL SİMÜLASYONU

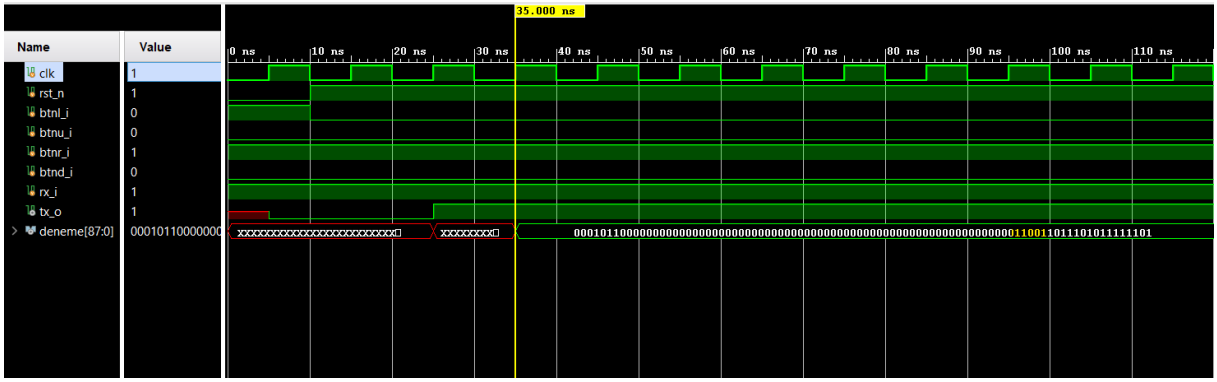
Ana modülümüzün işlevini gerçekleştirip gerçekleştirmediğini görmek amacıyla bir testbench oluşturduk. Ana modüle FPGA'den PC'ye dönecek paket sonucunun doğru olup olmadığını görmek için deneme isimli bir çıkış ekledik. Öncelikle modül içerisinde atadığımız değerler bu şekildedir.

```
initial begin
paket1[79:16] = 64'd12;  // SAYI1
paket1[143:80] = 64'd13; // SAYI2
end
```

Öncelikle modül içerisinde atadığımız değerler bu şekildedir.

```
    initial begin

    rst_n=1'b0;
    clk=1'b0;
    rx_i=1'd1;
    btnl_i=1'b1;
    btnu_i=1'b0;
    btnr_i=1'b1;
    btnd_i=1'b0;
    #10;
    rst_n=1'b1;
    btnl_i=1'b0;
    btnu_i=1'b0;
    btnr_i=1'b1;
    btnd_i=1'b0;

    end
endmodule
```

İnitial bloğunda deneme yaparken ilk kısımda vermiş olduğumuz buton değerlerinin bir önemi yok çünkü rst_n=1'b0 verildiğinden aktif-düşük reset çalışmakta. Bir sonraki iterasyonda rst_n=1'b1'e getirilip btnr_i=1'b1 değeriyle aktif edilmiştir.



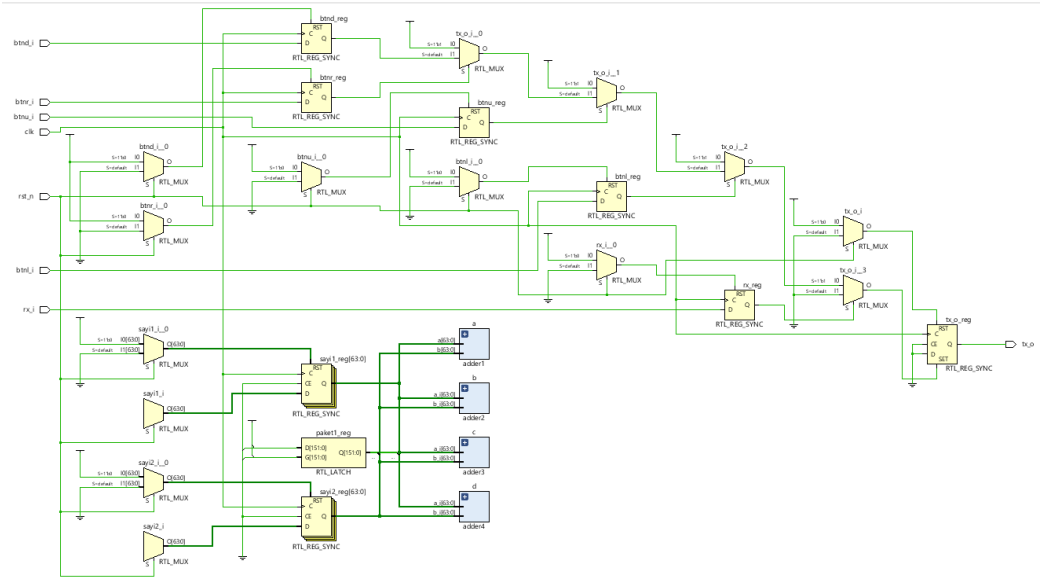
PC’den FPGA’e gidecek pakette verdiğimiz sayi1 ve sayi2 değerlerinin toplamı 64’d25 olmalıydı. [15:0]’lık başlık bitinin ardından ikilik tabanda 11001 yani 10’luk tabanda 25 sonucuna ulaşılmış ve simülasyon başarı ile gerçekleştirilmiştir.

Zaman analizi ve Elaborated Design’a raporda yer verilmiştir.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7,577 ns	Worst Hold Slack (WHS):	0,134 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2	Total Number of Endpoints:	2	Total Number of Endpoints:	7

All user specified timing constraints are met.



2.6 FPGA CONSTRAINS

FPGA ile yazılan kodun uyumlanması açısından “constrain” içerisinde tuşlara, input değerlerinin karşılığını atamamız gerekmekte. Bunu yaparken, FPGA üzerindeki yazılan yerlere dikkat ettik. Constrainleri ise kaynakçada belirtilen siteden bulduk. İstenilmeyen inputların başına # ekledik.

Projemizin isteminden ötürü; SW0→rst sinyali aldı.

BTN kısımlarında da yönlerine göre atama yaptık.

```
set_property PACKAGE_PIN V17 [get_ports {rst_n}]
    set_property IOSTANDARD LVCMOS33 [get_ports {rst_n}]
#set property PACKAGE PIN V16 [get ports {sw[1]}]
```

```
##Buttons
#set_property PACKAGE_PIN U18 [get_ports btnu_i]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnu_i]
set_property PACKAGE_PIN T18 [get_ports btnu_i]
    set_property IOSTANDARD LVCMOS33 [get_ports btnu_i]
set_property PACKAGE_PIN W19 [get_ports btnl_i]
    set_property IOSTANDARD LVCMOS33 [get_ports btnl_i]
set_property PACKAGE_PIN T17 [get_ports btnr_i]
    set_property IOSTANDARD LVCMOS33 [get_ports btnr_i]
set_property PACKAGE_PIN U17 [get_ports btnd_i]
    set_property IOSTANDARD LVCMOS33 [get_ports btnd_i]
```

4. KAYNAKLAR

- Digital Design with an Introduction to the Verilog by M.Morris Mano and Michael D. Ciletti
- https://github.com/Digilent/Basys3/blob/master/Projects/XADC_Demo/src/constraints/Basys3_Master.xdc
- https://www.youtube.com/watch?v=ECLJU3-0SzU&list=PLZyLAHn509339oyv3vi-3Gdyb8bfPx7Ro&index=23&t=15s

- https://www.youtube.com/watch?v=VuwI-kQ1Vwg&list=PLZyLAHn509339oyv3vi-3Gdyb8bfPx7Ro&index=23
- https://faculty.kfupm.edu.sa/COE/aimane/coe405/FPGA%20Prototyping%20with%20Verilog%20examples.pdf
- https://verilogcodes.blogspot.com/2017/11/verilog-code-for-carry-select-adder.html