**GRUP 1**

**PROJE RAPORU**

**Grup üyeleri:**

* Hasan TUNA – Bilgisayar Mühendisliği

**Proje Amacı:**

1. *Basys3 FPGA* kartı ile bilgisayardan *UART* protokolü yardımıyla veri almak,
2. Gelen sayı üzerinde karekök, sin, cos, asallık kontrolü işlemlerini gerçekleştirmek,
3. Sonucu *Basys3* üzerindeki 7 bölgeli ekranda *(7-Segment Display)* görüntülemek.

**Bu kısımdan itibaren modül açıklamaları başlamıştır. Kod bloklarına çift tıklayarak başka dosya üzerinde *text* olarak açabilirsiniz. (Kopyalamak veya daha rahat okumak için bunu kullanın)**

**Dosyaların tam koduna erişmek için eklere bakınız. (Dosyalarda bolca açıklama vardır)**

**Kullanım: A-a => karekök, S-s => sin, D-d => cos, F-f => asal kontrol**

1. **UART**

*UART (Universal Asynchronous Receiver/Transmitter)* protokolü, iki cihazın *USB (Universal Serial Bus)* girişleri üzerinden veri gönderip okuyarak birbiriyle konuşmasını sağlar. Bu veri transferi, iki cihazın aynı frekansta bit gönderip almasıyla meydana gelir. Bu frekansa *Baud Rate* denir. Bu proje için *Baud Rate* değeri 9600 seçilmiştir. Yani saniyede 9600 adet bit transfer edilir. Bu transfer işlemi, projenin *basys3.xdc* adlı dosyasında tanımlanan *RsRx (Alıcı pin)* ve *RsTx (Verici pin)* pinleri kullanılarak yapılmıştır.

*basys3.xdc (Alıcı-Verici Tanımlama Bölümü)*



Herhangi bir veri gelmezken *RsRx* pinine mantık-1 değeri gelir. Gelen ilk mantık-0 değeri *(start bit)*, veri transferinin başladığını belirtir. Bundan sonra gelecek olan 8 bitlik değer *(data)*, veriyi oluşturur. En son mantık-1 değeri *(stop bit)* gelir ve bu transferin bittiğini belirtir. Böylece 10 bit ile bir veri transferi tamamlanmış olur.

Bunu gerçekleştirmek için öncelikle 9600 olan *baud rate* ayarlamamız gerekir. *FPGA* üzerindeki *100MHz (100.000.000Hz)* frekansa sahip *clock’u* uygun şekilde yavaşlatmalıyız.

***Clock* Yavaşlatma**

Saniyede 9600 kere veri alımının gerçekleşmesini istiyoruz. Bunu sağlamak için *clock* her 10416. döngüsünü tamamladığında 1 biti işleme almalıyız. Bu şekilde saniyede *100.000.000/10416 ≈ 9600* kere 1 biti işleme almış oluruz. Bu da bilgisayardan gönderilen her bitin yalnızca 1 kez işleme alınmasını garantilemiş olur.

*uart.v (Clock Yavaşlatma)*



**Gelen Bitleri İşleme (Durum makinesi)**

*Clock’u* yavaşlattığımıza göre artık gelen bitleri değerlendirebiliriz. Bunun için bir durum makinesi kullanacağız.

*0. Durum:* Bekleme durumu. Bu durumdayken herhangi bir veri gelmiyordur ve *RsRx* değeri mantık-1 dir. RsRx değeri mantık-0 olduğu zaman veri akışı başlamıştır. Makine 1. duruma geçer.

*1. Durum:* Veri gelmeye başlamıştır. Bu durumdayken 8 kere *RsRx’e* gelen bit okunur ve kaydedilir. 8. okumanın ardından makine 2. duruma geçer.

*2. Durum:* Bu aşamada veri transferi bitmiştir. 1. durumda kaydedilen 8 bitlik veri değerlendirilir. Makine 0. duruma geri döner ve yeni veri transferi için hazırlanır.

*uart.v (Gerekli Değişkenler)*



*uart.v (0.Durum)*



*uart.v (1. Durum)*



*uart.v (2. Durum)*



Bu sayede *uart.v* modülü içerisinde bilgisayardan gönderilen verileri okuyup değerlendirmiş olduk.

1. **7-SEGMENT DISPLAY**

*7-Segment Display’in* 3 adet kontrol değişkeni vardır. Bunlardan birisi seg (7-bit), ekranda yazacak sayının belirlenmesinde kullanılır. Bir diğeri an (4-bit) kullanılacak basamağı belirtir. Son olarak dp (1-bit) noktanın gözüküp gözükmeyeceğini belirtir.

*basys3.xdc (Ekran Kontrolcülerin Tanımlama Bölümü)*



**Sayının Gönderilmesi**

Ekranda anlamlı sayılar yazmak için “seg” değişkenine uygun sırada bit dizisi verilmelidir. Mantık-1 verilen segmentler yanmazken mantık-0 verilenler yanar. Bunun kontrolü için bir *always* bloğu yazdık.

*ekran.v (Sayıyı ekrana yazan bölüm)*



**Basamakların Kontrolü**

*7-Segment Display’in* farklı basamaklarında aynı anda farklı sayılar yazmak mümkün değildir. Basamaklar çok hızlı açılıp kapatılarak aynı anda farklı sayılar yazıyormuş algısı yaratılır. Bunun için *FPGA clock’una* bağlı bir durum makinesi yazdık.

*ekran.v (İstenilen basamağa yazan kısım)*



Böylece hesaplanan sonuçları ilgili basamaklara yazmış olduk.

1. **KAREKÖK**

Karekök hesaplaması yapmak için *Newton-Raphson Method* algoritması kullandık.

**Algoritma**

1. Karekökü istenilen sayının (X olsun) karekökü hakkında bir tahmin yapılır. (i olsun)
2. *i = (i + (X/i))/2* işlemi birkaç kere tekrarlanır. Ne kadar çok tekrarlanırsa gerçek sonuca o kadar yaklaşırız.

Burada dikkat etmemiz gereken şey *FPGA’in* ondalıklı sayıları hafızada tutamaması. Dolayısıyla karekök(10) = 3.16 yerine 3 hesaplanacaktır. Bunun üstesinden gelmek için karekökü sorulan sayısı 10000 ile çarparız. Karekök(100000) = 316 sonucu elde edilir. Sonucun basamaklarını uygun şekilde ayırarak ekrana yazar ve noktayı olması gerektiği yere koyarız.

*karekok.v (İşlem Bloğu)*



**4. SİN**

Önce sinüs fonksiyonu için *Taylor Serisinden* gelen bir formül olan sinüs polinomunu kullanmıştık. Simülasyonda çalışan bu yolun radyan hesaplaması için 3.14 sayısını kullanmasından dolayı *FPGA* üzerinde çalışmaması üzerine öncelikle açının ait olduğu bölgeyi hesaplayan ve sonra da hazır bir *lookup* tablosundan veri çeken bir kod yazdık. *Lookup* tablosunu oluşturmak için bir java kodu yazdık.

*sinLookup.java (Lookup Üretici Main Kodu)*



Ürettiğimiz *lookup* tablosunu kullanarak şu modülü yazdık.

*sin.v (İşlem Bloğu)*



**5. COS**

Önce cosinüs fonksiyonu için *Taylor Serisinden* gelen bir formül olan cosinüs polinomunu kullanmıştık. Simülasyonda çalışan bu yolun radyan hesaplaması için 3.14 sayısını kullanmasından dolayı *FPGA* üzerinde çalışmaması üzerine öncelikle açının ait olduğu bölgeyi hesaplayan ve sonra da hazır bir *lookup* tablosundan veri çeken bir kod yazdık. *Lookup* tablosunu oluşturmak için bir java kodu yazdık.

*cosLookup.java (Lookup Üretici Main Kodu)*



Ürettiğimiz *lookup* tablosunu kullanarak şu modülü yazdık.

*cos.v (İşlem Bloğu)*



**6. ASAL**

Asal sayı kontrolü için istenilen sayıyı kendisinden küçük bütün pozitif sayılara (1 hariç) bölünüyor mu diye baktık. Bölündüğü 1 durum bile bulduysak asal olmadığına ulaştık.

*asal\_sayi.v (İşlem Bloğu)*



**7. KONTROLCÜ**

Tüm modüllerimiz hazırdı ve bu modülleri uygun şekilde kontrol edecek bir ana modüle ihtiyaç vardı. Bunun için kontrolcü modülünü yazdık. Giriş-çıkışları uygun oldukları yerlere bağlayarak uart.v modülünden gelen sayıyı işlemlere bağlayıp sonuçları hesaplıyor, yine uart modülünden gelen “islem” değerine göre istenilen sonucu ekrana gönderiyor.

*kontrolcu.v (Modül Giriş-Çıkışları)*

**

*kontrolcu.v (Diğer Modüllerin Oluşturulması)*

**

*kontrolcu.v (Uygun Sonucun Ekrana Gönderilmesi)*

**

Böylece devre tasarımımız tamamlandı ve istenilen işlevlere sahip *FPGA* kodlandı.

**RAPOR SONU**