KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

GEZGİN HABERLEŞME UYGULAMALARI – TMS320C6711 İLE GAUSS MİNİMUM KAYDIRMALI ANAHTARLAMA MODÜLASYON VE DEMODÜLASYON UYGULAMASI

YÜKSEK LİSANS

Elektronik ve Haberleşme Müh. Sevil ÇELİKLER

Anabilim Dalı : Elektronik ve Haberleşme Mühendisliği

Danışman : Yrd.Doç.Dr.Sıtkı ÖZTÜRK

KOCAELİ,2007

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

GEZGİN HABERLEŞME UYGULAMALARI – TMS320C6711 ile GAUSS MİNİMUM KAYDIRMALI ANAHTARLAMA MODÜLASYON VE DEMODÜLASYON UYGULAMASI

YÜKSEK LİSANS TEZİ

Elektronik ve Haberleşme Müh. Sevil ÇELİKLER

Tezin Enstitüye Verildiği Tarih: 27 Mayıs 2007

Tezin Savunulduğu Tarih: 23 Temmuz 2007

Tez Danışmanı	Üye	Üye
Yrd.Doç.Dr.Sıtkı ÖZTÜRK	Prof.Dr.Hasan DİNÇER	Yrd.Doç.Dr.Soner ÖZGÜNEL
(.....)	(.....)	(.....)

KOCAELİ,2007

ÖNSÖZ ve TEŞEKKÜR

Bilindiği gibi haberleşme sürekli gelişmekte ve yeni uygulamalara açık bir konudur. Çeşitli modülasyon türleri ve uygulamaları ile daha verimli ve uygulamaya daha elverişli sistemler geliştirilmektedir. DSP işlemcilerin en büyük üstünlüğü ise hızlı olmalarıdır. Bu özellikleri ile haberleşme alanında birçok kolaylıklar getireceği açıktır. Bu alanda çalışmak isteyen arkadaşlara katkıda bulunması dileğiyle .

Tez çalışmaları esnasında beni anlayışla karşılayan, her zaman maddi ve manevi desteğiyle ile yanımda olan eşim Önder Yücel ve kızım Çağla Yücel'e, bu alanda çalışma isteğimi belirttiğimde her türlü yardım ve teşviklerinden dolayı tez danışmanım Sıtkı Öztürk'e, bu konuda çalışmamı destekleyen ve çalışma fırsatı veren proje yöneticilerim İ. Can Çevikbaş ve Metin Dalkılıç'a, proje ve tez aşamasında fikirleri ile beni yönlendiren ve yardım eden arkadaşlarım M. Salih Çalışkan, Celal Mızrak, Yeşim Demirkol ve Elanur Şireli'ye, ayrıca hiçbir zaman yardımını esirgemeyen Fen Bilimleri Enstitüsü Öğrenci İşleri Bölümü'nden Melek Özer'e teşekkür ederim .

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SEMBOLLER

T : Periyot (sn)
f : Frekans (Hz)
w : Radyan Frekans
φ : Faz
* : Konvolüsyon işlemi
B : Band Genişliği
Δφ : Faz Farkı

Kısaltmalar

GMSK : Gauss Minimum Shift Keying
DSP : Digital Signal Processors
MSK : Minimum Shift Keying
QPSK : Quadrature Phase Shift Keying
OQPSK : Offset Quadrature Phase Shift Keying
CCS : Code Composer Studio
RTDX : Real Time Data Exchange
DSK : Dsp Starter Kit
BER : Bit Error Rate
SNR : Signal Noise Ratio

TMS320C6711 İşlemcisi ile GMSK Modülasyon ve Demodülasyon Uygulaması

Sevil ÇELİKLER

Anahtar Kelimeler : GMSK, DSP, Modülasyon, Demodülasyon, TMS320C6711

ÖZET: Haberleşme sistemlerinde, asıl sorun sınırlı donanımsal kaynaklarla yüksek hızda veri iletmek ve kanal bant genişliğidir. Alıcı taraftaki problem ise en iyi sezme başarımının sağlanabilmesi için zamanlama bilgisinin doğru elde edilerek düşük bit hata oranı ve simgelerarası karışma olmadan veriyi yeniden elde etmektir. Senkronizasyon ve işareti alma için varolan çeşitli analog tekniklerine rağmen, sayısal işaret işlemedeki avantajlar, tüm sayısal gerçeklemelerde kullanılan algoritmaların geliştirilmesini teşvik etmektedir. Bu çalışmada amaç veriler üzerinde TMS320C6711 DSP işlemcisi kullanılarak GMSK modülasyon ve demodülasyon yapmayı hedeflemektedir. GMSK modülasyon bölümünde sayısal veri GMSK işaretine çevrilir ve kanal içinden iletilir. GMSK demodülasyon bölümünde ise, bir bit diferansiyel sezme ve örnekleme fazının düzeltilmesi için örneklenerek alınan veriler üzerinden Gardner eşzamanlama yöntemini kullanarak veri tekrar elde edilir. Sistem performansı farklı işaretlerde bit hata oran değerine göre test edilerek, gerçek zaman için istenen gereksinimleri karşıladığı belirtilecektir.

GMSK Modulation and Demodulation With TMS320C6711

Sevil ÇELİKLER

Keywords : GMSK, DSP, Modulation, Demodulation, TMS320C6711

ABSTRACT : In communication systems, a major requirement is to transmit the data at high rate using the limited resources of hardware and channel bandwidth. Another requirement of the receiver is to estimate symbol timing so that symbol decisions may be made reliably at the instant of smallest inter-symbol interference to get very low bit error rate. Though there exist various analog techniques for synchronization and detection, advances in digital signal processors (DSP) have prompted the development of algorithms that use an all-digital realization. The GMSK transmitter transmits data to the channel by converting it into GMSK signal, and receiver employs one-bit differential detection and uses Gardner's timing-error algorithm on sampled samples for sampling phase correction. The performance of the system is tested for the cases of bit error rates for different signal to noise ratio and time it takes for real time requirements of the system.

BÖLÜM 1. GİRİŞ

Haberleşme sistemlerin de, ana ihtiyaç bant sınırlı kanallarda düşük band genişliği kullanan ve alıcı tarafta optimal sembol kararını vererek sembol tahmini yapan modülasyon ve demodülasyon teknikleri kullanmaktır. Sayısal işaret işlemcilerin güçlü hesaplama özellikleri ve yüksek hızda geliştirme yapmaları ile birlikte bugün artık, sayısal haberleşme alanında artan bir ilgi görmektedir. Tamamı sayısal olan tasarımlarla modem uygulamaları, analog devrelerin azalmasına ve DSP 'nin hesaplama yükü ve analog-sayısal çeviricilerin yükü artmaktadır. Bu projede Gauss Minimum Anahtarlama tekniği kullanılarak, Dect sistemlerde ve GSM sisteminde kullanılan modülasyon ve demodülasyon işleminin sayısal işaret işlemci ile çözümü hedeflendi. Alıcıdaki algoritmalar, sembol eş zamanlamasını geriye elde etmek için örnek veri üzerinde çözümleme yaparlar. Bu proje de üzerinde TMS320C6711 DSP işlemci bulunan DSK kartı üzerinde gerçek zamanlı GMSK alıcı ve verici uygulaması yapılmıştır.

1. 1 Problem Tanımı

Optimal performans için, alınan haberleşme işaretleri, simgelerarası karışmanın (Inter Symbol Interference ISI) düşük olduğu anda örneklenmelidir. Bu işaretin bant genişliği azaltılması açısından istenerek yapılan ISI değeri GMSK için kritik bir değerdir. Bu ya analog yada sayısal işlemede A/D çeviricinin fazının ayarlanması için yapılması gereken genel bir işlemdir, buradan gelen işaretten uygun anda örnekleme yapılmalıdır. Asenkron olarak alternatif yaklaşım yapıldığında, A/D çeviriciden elde edilen sayısal veri, senkron örneklerin tahmini ile yeniden elde edilir. Alıcı ve verici mükemmel olarak senkron olması muhtemel olmadığından, alıcının sürekli saati takip etmesi ve işaretleşme hızı ile A/D çevirici saati arasındaki herhangi zamanlama farkını doğru alması gereklidir. Birçok sembol alındığı zaman, örnek zamanlama hatası hesaplanarak tahmin edilir. Bununla birlikte, sistemin veri hızı azalır.

1. 2 Sembol Eş Zamanlama

Alıcı da işaretin doğru olarak alınabilmesi için zamanlama bilgisi elde edilerek verici ile eş zamanlama sağlanmalı ve bu sayede her bir sembolün doğru zaman aralığında çözümlenmesi gerçekleştirilmelidir.

Alıcıda belirlenmesi gereken ilk parametre sembol veya örnekleme frekansıdır. Alıcıda sembol periyodunun kestirilmesi gerekmektedir. İletim hızı alıcıda bilinmesine rağmen sistemde kullanılan osilatör elemanlarının belirsizlikleri nedeniyle küçükde olsa sürüklenmeler meydana gelmektedir ve bu nedenle alıcıda küçük sapmalar oluşmaktadır. Sistemin çalışmasına etki etmemesi için alıcı ile verici arasındaki sembol oranı eş zamanlamasının sürekli olarak sağlanması gerekmektedir.

Eş zamanlama için alıcıda sembol fazınında belirlenmesi gerekir. Sembol fazında eş zamanlama sağlanarak sembol zaman dilimi içerisinde örneklemenin yapılacağı en iyi zaman anı belirlenmektedir. Bu nedenle sayısal haberleşmede, eş zamanlama için farklı yöntemler bulunmaktadır, Bunlar; taşıyıcı eş zamanlaması içeren, belirli bir kod kelimesi içeren eş zamanlama, çerçeve eş zamanlaması, saat eş zamanlaması, gibi… Bazı makalelerde, saat yada sembol eş zamanlamasına dayalı çalışmalar yayımlanmıştır ve zamanlamayı elde etme performansı için çeşitli algoritmalar amaçlanmıştır. Geçmişte bazı algoritmalar, örnekleme frekansını veya zamanlama hatasını azaltmak için geri beslemeli veya ileri beslemeli A/D çeviricinin fazını merkez almışlardır [2].

Bu çalışmada yaygın olarak kullanılan Gardner eş zamanlama yöntemi kullanımı hedeflenmiştir. Floyd Gardner, senkron BPSK / QPSK için zamanlama hatasını bulmak için basit bir algoritma önermiştir. Gardner'ın Algoritması, her bir sembol için iki örnek gerektirir, fakat algoritma zamanlama hatasını hesaplamak için yanlızca üç örneğin toplamını kullanır. İki örnekten birisi, senkronize örnek olarak varsayılır. Gardner eş zamanlama yaklaşımında zamanlama hatası kutuplu iletim için $e_n = (y_n - y_{n-2}).y_{n-1}$ olarak hesaplanmakta ve bu hataya göre örnekleme zamanları düzeltilmektedir. y_n ile y_{n-2} örnekleri arasında T_s (Örnekleme periyodu) kadar, y_n ile y_{n-1} örnekleri arasında $T_s/2$ kadar süre farkı bulunmaktadır. Gardner yöntemi

yaklaşık olarak %40 ile %60 arasında fazlalık bant genişliğine sahip Nyguist darbeleri kullanan doğrusal modülasyon sistemleri için kullanılabilmektedir. Gardner eş zamanlama yaklaşımı yüksek SNR iletimlerinde daha iyi başarım sağlamaktadır. Taşıyıcı modülasyonu kullanan sistemlerde, Gardner yaklaşımı ile sembol zamanlama bilgisinin geri elde edilmesi işlemi, taşıyıcı fazının elde edilmesinden bağımsız olarak gerçekleştirilebilmekte ve bu sayede taşıyıcı kaymalarına karşı dayanıklılık sağlamaktadır. Bu durumda sembol zamanlama döngüsü önce kilitlenerek taşıyıcının geri elde edilmesi işlemini kolaylaştırabilmektedir [4].

BÖLÜM 2. GAUSS MİNİMUM KAYDIRMALI ANAHTARLAMA (Gaussian Minimum Shift Keying – GMSK)

2. 1 GMSK

Mobil haberleşme açısından baktığımızda, komşu kanallar içindeki band dışı radyasyon gücü, genel olarak istenen kanal içinde 60 – 80 desibel bastırılmalıdır. Bu katı kurallara uymak için, çıkış işaretinin spektrumunu işlemek gereklidir. Düzgün bir dalga formu, az miktarda yüksek frekans bileşeni içerir. Böylece, mümkün olduğu kadar düzgün bir işaret elde etmek için, iyi bir spektrum izlenmelidir. Düzgün bir dalga şekli için sürekli değişen faz gereklidir [6].

Küçük çıkış güç spektrumu elde etmek için, ön modülasyon alçak geçiren filtrede, aşağıdaki özellikler takip edilmelidir.

1. Dar band genişliği ve keskin kesim .
2. Düşük aşma (overshoot) dürtü yanıtı .
3. Filtre çıkış darbe alanında faz kaydırma π/2'ye karşılık gelen alanda korunma

Durum;

1. Yüksek frekans bileşenleri bastırmaya,
2. Anlık frekans sapmalarına karşılık korumaya,
3. Basit MSK gibi uygulanabilir olan uyumlu sezmeye ihtiyaç duyulur.

GSM standartları, uyumlu sezme ile minimum kaydırmalı anahtarlama (MSK) ön modülasyon Gauss filtrelemeyi önerir. Geleneksel MSK ile benzer olmayan, GMSK modülasyonunda işaretler, öncelikle 3 dB bant genişliğine sahip alçak geçiren Gauss filtreden geçirilir ve daha sonra çıkış işareti faz modülasyonuna göre modüle edilir. Gauss filtrenin birim darbe cevabı oldukça düzgün olduğundan, böylece, modüleli işaretin fazı, sembol periyodu üzerinde sürekli değişir.

GMSK band dışı gücü küçüktür ve kablosuz haberleşme için istenen karakteristiği veren sabit bir zarfa sahiptir. Band dışı gücü, Gauss ön modülasyon filtresinin keskin kesim frekansı ile oldukça bastırılır.

2. 2 GMSK Altyapısı

GMSK, kablosuz ve mobil haberleşmede genel olarak kullanılan sayısal modülasyon tipidir. GMSK'da, taşıyıcının fazı Gauss filtre ile filtrelenmiş işarete göre sürekli değiştirilir. Minimum kaydırmalı anahtarlama (MSK) modülasyonun bir tipidir, GMSK 0.5 modülasyon indeksine sahiptir ve diferansiyel sezim (Differential Detection) kullanılarak demodüle edilebilir.

Gauss filtre, düşük band dışı gücün istenen karakteristiği için izin verilen enerjiyi bir yerde toplar. GMSK 'nın geniş anlamda avantajları vardır. Bunlar, oldukça dar band genişliği, sabit zarf modülasyonu, faz uyumlu ve faz uyumsuz deteksiyon içinde uygun olmasıdır. Sabit zarf GMSK'yı çevresel bozulmalardan daha az etkilenir olmasını sağlar ve bu bozulma genlik modülayonundakinden daha azdır. Bu avantajlar GMSK 'yı hücresel ve yer mobil radyo kanalları için GSM standardı için uygun hale getirir [1].

Bu bölümde GMSK modülasyon ve demodülasyonun bazı detaylarını inceleyeceğiz. GMSK modülasyonu için bir bit dizisi sıfıra dönüşsüz dizi olarak tasarlanmıştır ve Gauss birim cevabı ile alçak geçiren filtreden geçirilir. ISI'nın miktarı Gauss iletim filtresinin band genişliği – zaman çarpanı BTb 'ye bağlıdır. Filtrenin çıkışında IF taşıyıcı ile modüle edilir. GMSK işaretleşmesi için, demodülasyon düzenli MSK gibi diferansiyel sezim işleminden geçirilir.

2. 3 Farklı Teknikler ile Karşılaştırma

2. 3. 1 Dördün faz kaydırmalı anahtarlama (QPSK)

Dört adet işaret tanımlayalım, her biri birbirine 90 derece dik olsun, böylece dördün faz kaydırmalı anahtarlama elde etmiş oluruz. Giriş ikili bit dizisi {dk}, dk=0,1,2…

modülatör giriş hızı 1/T bit/sn ‘e ulaşır ve dI(t) ve dQ(t) olarak iki bit dizisinden sırasıyla çift ve tek bitleri gösterir .

$$dI(t) = d_0, d_2, d_{4,\ldots} \tag{2.1}$$
$$dQ(t) = d_1, d_3, d_{5,\ldots} \tag{2.2}$$

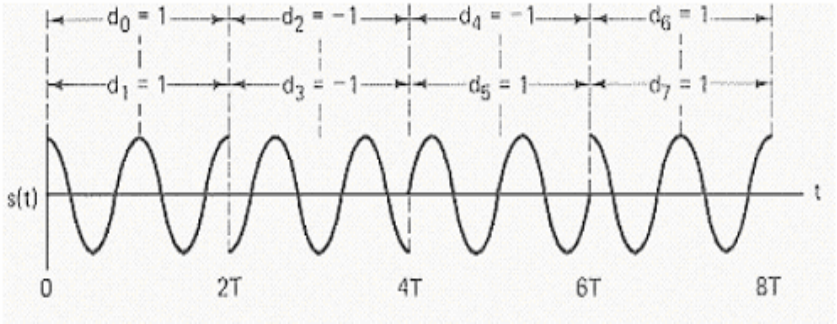
QPSK dalga şeklinin uygun ortogonal gerçeklemesi, s(t) in genlik faz modülasyonu ile elde edilir. Ve birbirne dik veri dizisi taşıyıcı dalganın kosinüs ve sinüs fonksiyonları üzerine (2.3) ‘deki gibi yerleştirilir.

$$s(t) = \frac{1}{2} dI(t) \cos 2\pi ft + \frac{p}{4} + \frac{1}{2} dQ(t) \sin(2\pi ft + \frac{p}{4}) \tag{2.3}$$

Trigonometrik özdeşlik kullanılarak yazılırsa aşağıdaki gibi elde edilir.

$$s(t) = A\cos[2\pi ft + \frac{p}{4} + q(t)] \tag{2.4}$$

Bununla birlikte her iki bileşenin de (dQ(t) ve dI(t)) işareti aynı anda değişirse QPSK taşıyıcısında 180^0 faz farkı meydana gelmektedir(Şekil 2.1). Bu durum yüksek frekanslı bileşenlere yol açar, haberleşme kanalı veya alıcıdaki bant sınırlı süzgeç etkileri ile birleştiğinde taşıyıcı genliğinde dalgalanmalara neden olarak alıcıda sembol hataları meydana gelmesine sebep olur.

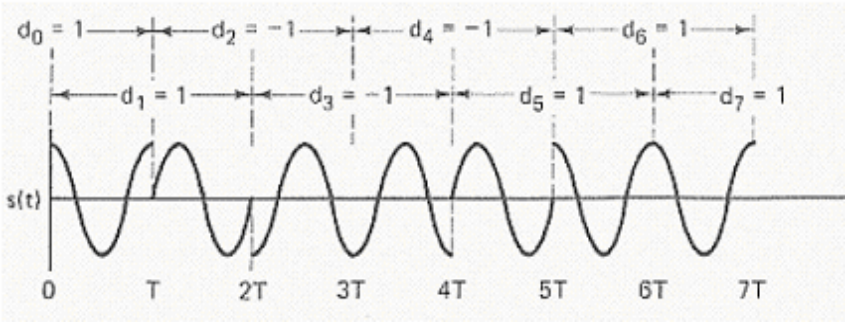


Şekil 2.1 : QPSK

QPSK modüleli işaretin spektral yan lobları azaltmak için filtreleme yapılırsa, sonuçta elde edilen dalga şekli sabit bir zarfa sahip olmayacaktır ve gerçekte, nadiren fazda 180 derece kayma geçici olarak zarfın sıfıra gitmesine sebep olacaktır .

2. 3. 2 Dengeli faz kaydırmalı anahtarlama (OQPSK)

İki bit dizisi I ve Q ½ bit aralığa göre öteleme değeri verilirse, o zaman genlik değişimleri asla 180 derece değişmeyeceği için minimum olur (Şekil 2.2). Ötelemeli faz kaydırmalı anahtarlama, Q kanalının yarım sembol peryodu kadar geciktirilmiş olduğu QPSK iletimine karşılık gelmektedir. QPSK ve OQPSK için bit hata oranı BPSK ile aynıdır.



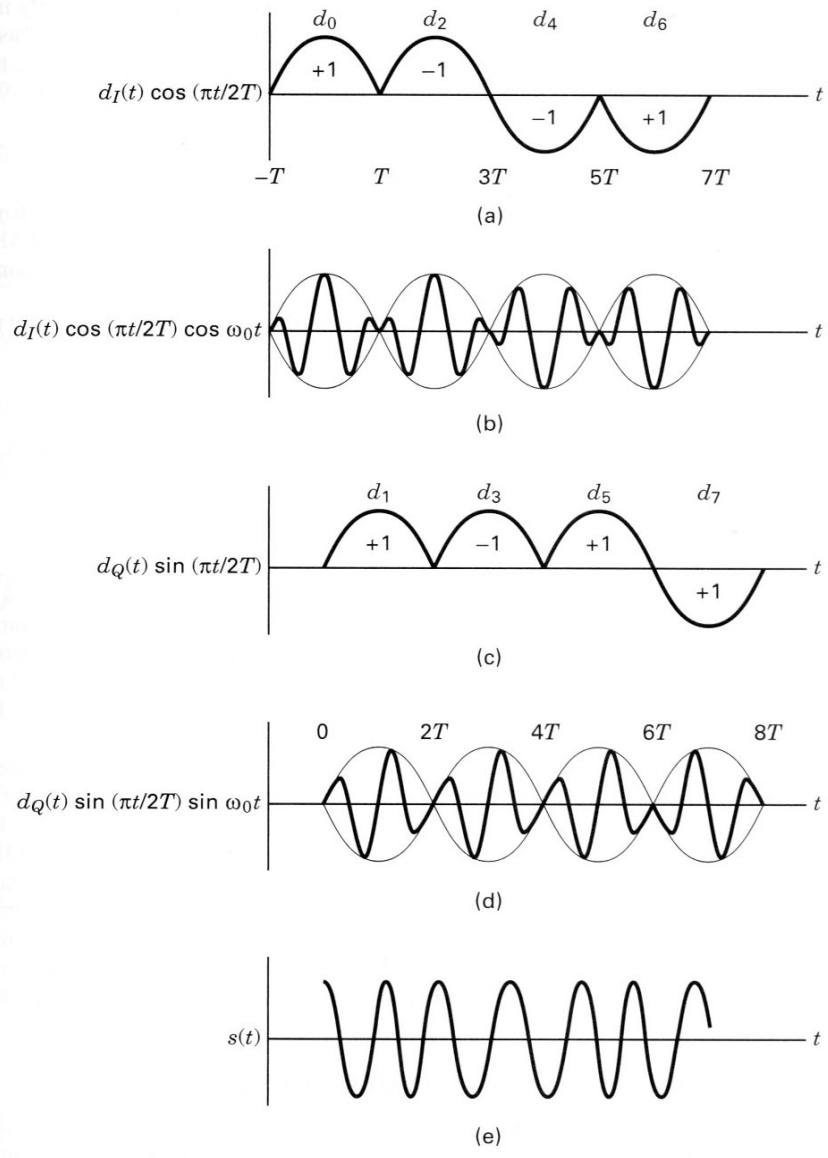
Şekil 2.2: OQPSK

2. 3. 3 Minimum kaydırmalı anahtarlama (MSK)

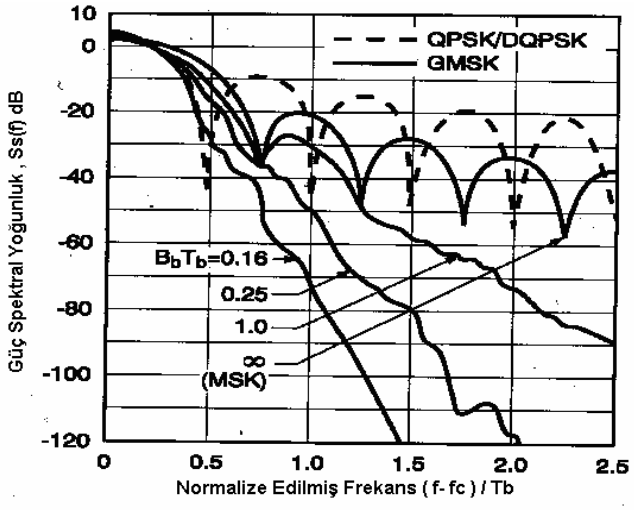
İletilecek işaretin seviye ve zarfında meydana gelen ani değişimlerin önlenmesi sonucu sistem performansında elde edilen iyileştirmeye ek olarak sembol geçişleri sırasında meydana gelecek faz devamsızlıklarının tamamen önlenmesi durumunda ek iyileştirme sağlanmaktadır. MSK sürekli faz modülasyon yöntemlerinden biridir. Burada faz sürekliliğini sağlayacak bir faz sabiti eklenmektedir. Fakat MSK modülasyonunda sinüssel darbe biçimlendirme kullanılarak işaret daha düzgün hale getirilmekte, fakat sembol geçişlerinde faz sürekli olmasına karşın frekanstaki süreksizlik devam etmektedir (Şekil 2.3).

2. 3. 4 Spektral güç yoğunlukları

Şekil2.4 ‘de QPSK, MSK ve GMSK için normalize edilmiş spektral yoğunluklar gösterilmektedir. GMSK için azaltılmış kenar enerjisine dikkat edilmelidir. Sonuçta, aynı bitişik kanal girişimi açısından MSK ile karşılaştırıldığında GMSK ‘da kanal aralığı daha dar olabilmektedir.



Şekil 2.3 : MSK



Şekil 2.4: Güç Spektral Yoğunluğu

2. 4 GMSK Modülasyonu

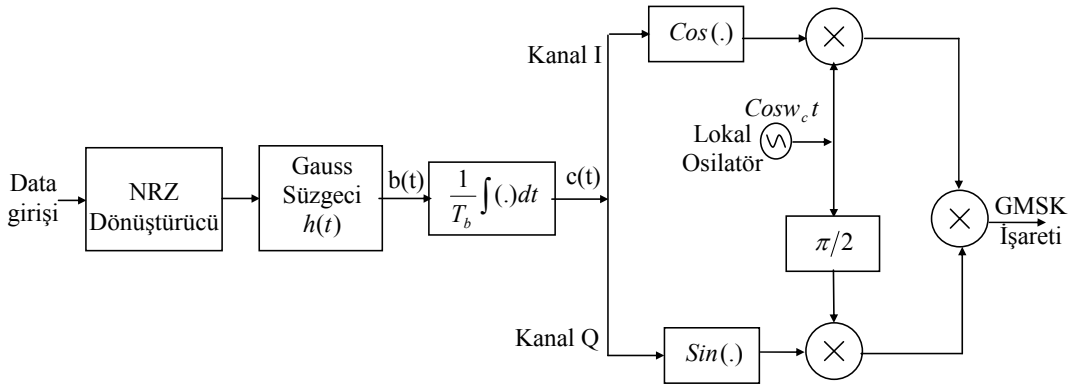
Şekil 2.5'de GMSK modülatörün blok diyagramı gösterilmektedir. Sürekli faz modülasyonunda iletilen işeret genel olarak ,

$$i(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi f_c t + \varphi(t, b_k)) \tag{2.6}$$

şeklinde ifade edilmektedir. Faz Modülasyonu,

$$\phi(t, b_k) = 2\pi h \int b_k g(\tau - kT_b) d\tau \tag{2.7}$$

şeklinde, modülasyon indisi h ve modülasyonda kullanılan darbe biçimi g(t) cinsinden ifade edilmektedir.



Şekil 2.5: GMSK Modülasyon Blok Yapısı

GMSK için Gauss darbe biçimlendirme süzgeci kullanılmaktadır. Gauss darbe biçimlendirme süzgecinin dürtü yanıtı

$$h(t) = \sqrt{\frac{2\pi}{\ln 2}} B e^{-\frac{2\pi^2 B^2}{\ln 2} t^2} \tag{2.8}$$

olmakta ve Gauss biçimlendirilmiş darbe biçimi

$$g(t) = Q(\pi B \frac{t + T_b/2}{\sqrt{\ln 2}}) - Q(2\pi B \frac{t + T_b/2}{\sqrt{\ln 2}}) \tag{2.9}$$

olarak elde edilmektedir. Burada T_b bit zaman dilimini,B ise Gauss süzgecinin 3dB bant genişliğini göstermektedir.

Genelde Gauss süzgecinin bant genişliği BT_b çarpımı cinsinden tanımlanmaktadır. Süzgecin bant genişliği $BT_b \geq 1$ olacak şekilde geniş tutulduğunda NRZ darbesinin çoğunluğu geçirilmekte ve Gauss darbe biçimlendirme etkisi çok düşük olduğundan iletilen işaret temelde MSK olmaktadır. Biçimlendirme süzgeci bant genişliği $BT_b < 1$ olacak şekilde belirlendiğinde ise darbe biçimlendirme meydana gelmektedir.

Gauss darbe biçimlendirmesinin sonucunda işaretin frekans spektrumu daraltılmakta, bunun sonucunda işaret zaman uzayında yayılmakta ve sonuçta simgelerarası karışma meydana gelmektedir. Özetle, GMSK modülasyonunda $B.T_b$ çarpımı sayesinde iletim için gerekli bant genişliği ile;bit hata olasılığı, bant dışı karışma, simgelerarası karışma ve iletim gücü arasında bir değiş tokuş yapılmaktadır.

Bu sürekli zaman algoritmasının ayrık zamanda gerçeklenmesi ile Matlab 'te GMSK örnekleri üretildi. Ek A'de Matlab kodu verilmektedir.

2. 5 GMSK Demodülasyonu

Her bir periyotta iletilen fazın değişimi bulunarak iletilen semboller bulunur. (2.10)'dan bu faz değişimi şöyle yazılabilir;

$$\Delta\varphi_b(t) = \varphi(t) - \varphi(t - T_b) = \pi / 2T_b \int_{t-T_b}^{t} d(t) * h(t)dt \qquad (2.10)$$

İletilen sembol a_k, faz değişiminin işaretine karşılık gelir. $\varphi(t)$ 'nin yeniden elde edilmesine göre, taşıyıcı birinci olarak kaldırılmalıdır.

Taşıyıcı frekansının bulunabilmesi için, IF işareti ana band işaretine çevrilir. Alınan işareti, temel band bileşenlerine ayırmak için işaret modülatöründen taşıyıcı ile

çarpılır. Böylece bir alçak geçiren filtre ile taşıyıcı bileşenleri olan I(t) ve Q(t) ‘ ye ayrılır. Temel band işaretinin demodülasyonu bir bit farksal sezim ile direk olarak başarılır. Eğer,

$$z(t) = I(t) + jQ(t) = A_r e^{j\varphi(t)} \tag{2.11}$$

Burada A_r, alınan vektörün büyüklüğü ise, böylece faz geçişi

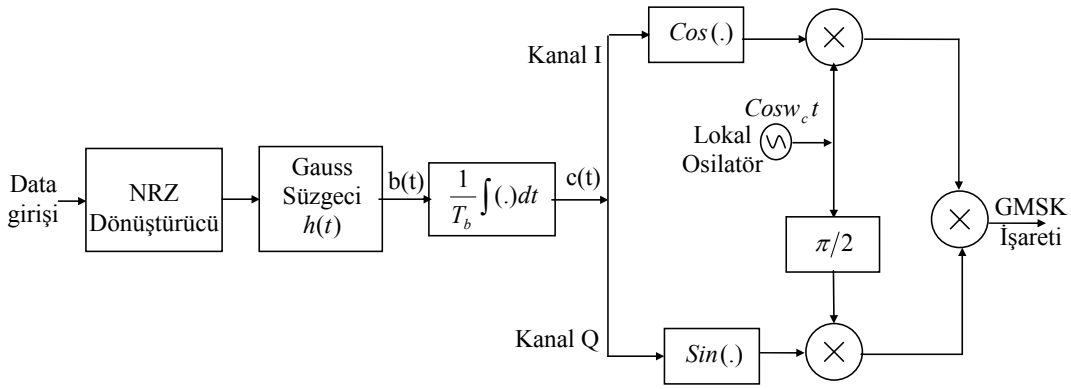
$$D(t) = \Delta\varphi_b(t) = img(z(t)z^*(t - T_b)) \tag{2.12}$$

olur. Burada img(.), demodüle edilen dalganın sanal kısmını gösterir ve D(t) ile gösterilir. Bir bit farksal sezim algoritması, alıcı da temel band I ve Q örnekleri kullanılarak DSP ile gerçeklenmiştir.

BÖLÜM 3. MATLAB SİMÜLASYONU

3. 1 Modülasyon

Matlab yazılımında Gauss minimum kaydırmalı anahtarlama çözümü ayrık zamanda yapıldı. Matlab'da GMSK verici uygulamasının blok diyagramı şekil 3.1'de gösterilmektedir ve Ek A'da çözüm için kullanılan program kodu verilmektedir. Şekilden de görüldüğü gibi, rastgele sayı üreteci, kullanıcının belirttiği genişliğe göre rastgele 1 ve 0 bitleri üretir.



Şekil 3.1: GMSK işaretinin üretilmesi

Burada bir örnek pratik olarak uygulamalarda iletimden kaynaklanabilecek problemlere karşı 10 kez gönderilmektedir. Bu işlem alıcı kısımda bitin doğru olarak sezilmesi için yapılmıştır.

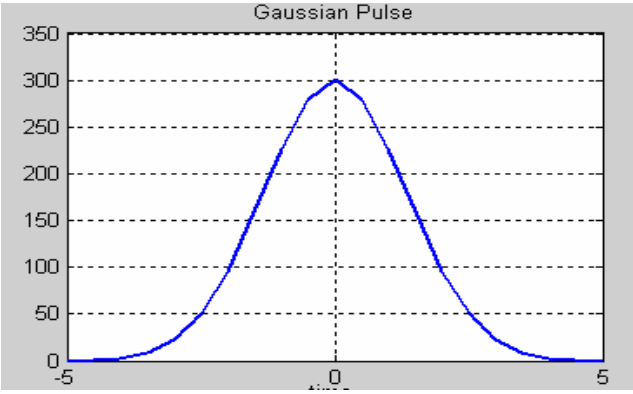
Verici tarafta gönderilecek bit 10 kez tekrarlanarak, gönderilir. Alıcıda belirli bir sayıda aynı bit alındığında alınan bit için karar verme işlemi yapılır.

Bitin tekrarlanması ile oluşan darbe işareti Gauss filtreden geçirilerek Gauss darbe biçimlendirmesi yapılır. Burada Gauss filtresinin $B.T_b$ değeri 0.5 'dir. Gauss filtrenin katsayıları (3.1) ve (3.2) eşitlikleri kullanılarak hesaplandı.

$$k = \pi \times sqrt(2/\log(2)) \tag{3.1}$$

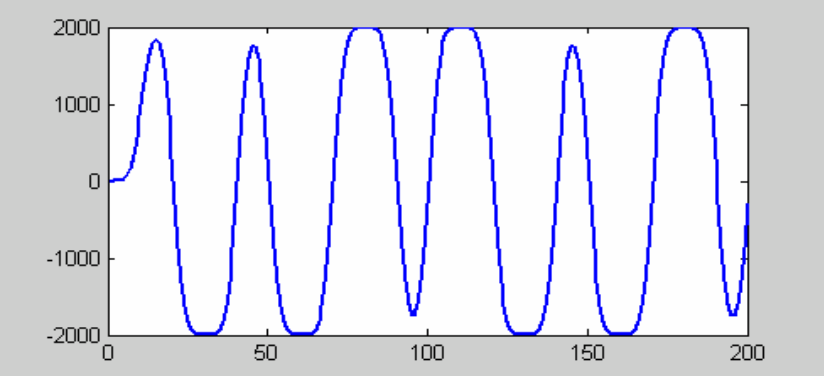
$$g(n) = (k \times B/sqrt(\pi)) \times \exp(-k^2 \times B^2 \times n^2) \quad (3.2)$$

Burada $n = -T_b : T_b / M : T_b$ olarak alındı. Şekil 3.2'de $B.T_b = 0.5$ olan Gauss darbesi gösterilmektedir.



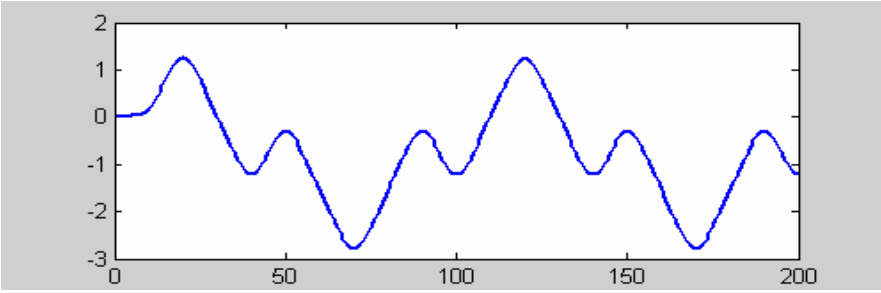
Şekil 3.2 : Gauss Darbesi

Şekil 3.3'te Gauss filtre ile filtrelenmiş veriler gösterilmektedir.



Şekil 3.3 : Darbe biçimlendirmeden geçirilmiş veri.

Gauss filtrenin çıkışı, Şekil. 3.4 'de gösterilmektedir.

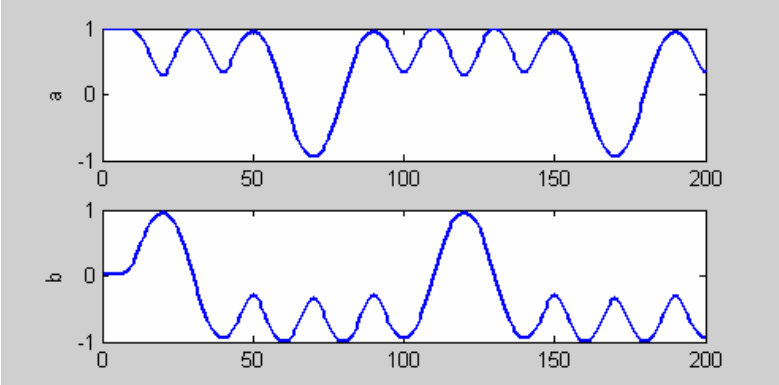


Şekil 3.4 : Faz Bileşeni

$$\phi(mT) = \pi/2 \times T_b \sum (d(mT) \times g(mT)) \quad (3.3)$$

İşaretin faz bileşeni, işaretin I ve Q bileşenlerini oluşturmak üzere kosinüs ve sinus ile modüle edilir(Şekil 3.5).

$$I(mT) = \cos(\phi(mT)) \quad (3.4)$$
$$Q(mT) = \sin(\phi(mT)) \quad (3.5)$$

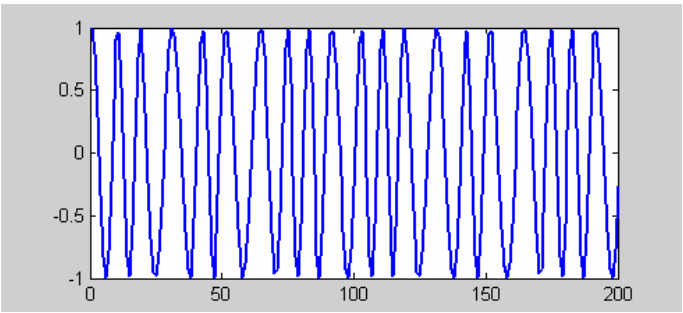


Şekil 3.5 : a) Eş-fazlı bileşen, b) Dik-Fazlı Bileşenleri

Bu iki dalga formu, toplandıktan sonra taşıyıcı ile modüle edilir ve iletilir .

$$x(mT) = I(mT)\cos(w_c mT) - Q(mT)\sin(w_c mT) \quad (3.6)$$

Şekil 3.6 'da 20 bitin taşıyıcı ile modüle edilmiş dalga şekli gösterilmektedir.

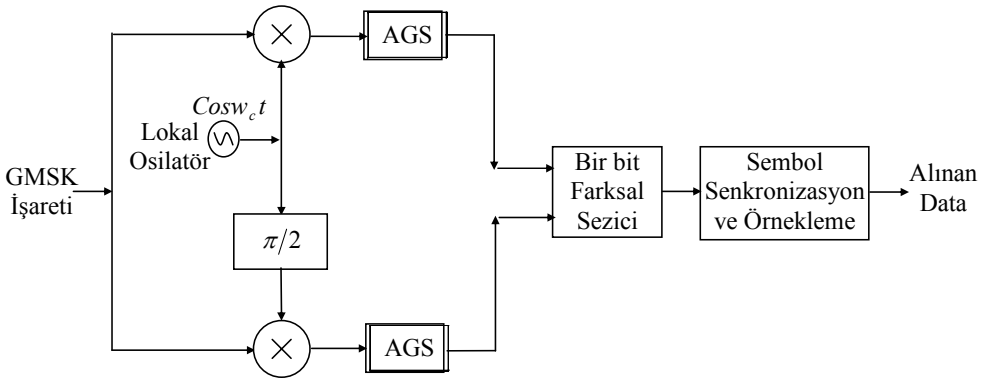


Şekil 3.6: Modüle edilmiş taşıyıcı

3. 2 Demodülasyon

Şekil 3.7'de blok diyagramda GMSK demodülasyonu için alıcı blok yapısı gösterilmektedir. Birinci olarak taşıyıcı, taşıyı ile veri çarpılarak kaldırılır ve alçak geçiren filtre ile filtrelenir.

$$r(t) = I(t)\cos(wt) - Q(t)\sin(wt) \tag{3.7}$$



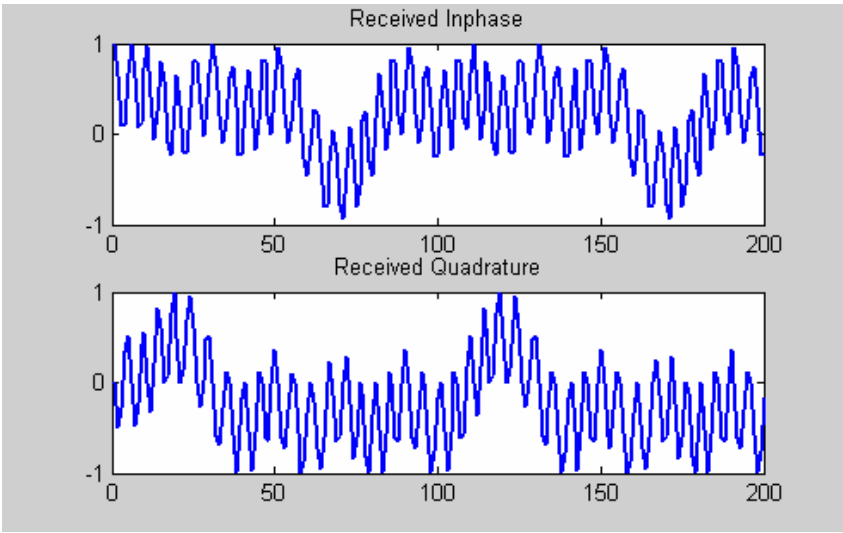
Şekil 3. 7 : GMSK Alıcı

$$r1(t) = [I(t)\cdot\cos(wt) - Q(t)\cdot\sin(wt)]\times 2\times\cos(wt) \tag{3.8}$$

$$r1(t) = I(t) + I(t)\cos(2wt) + Q(t)\cdot\sin(wt)] \tag{3.9}$$

Yüksek frekans bileşenleri alçak geçiren filtreden geçirilerek kaldırılır ve eşevreli bileşeni elde etmiş oluruz. Benzer şekilde alınan işareti sin(wt) ile çarparak dik evreli bileşeni de elde ederiz. Şekil 3.8'de yüksek frekans ve gürültülü işaret gösterilmektedir.

$$r2(t) = Q(t) \tag{3.10}$$

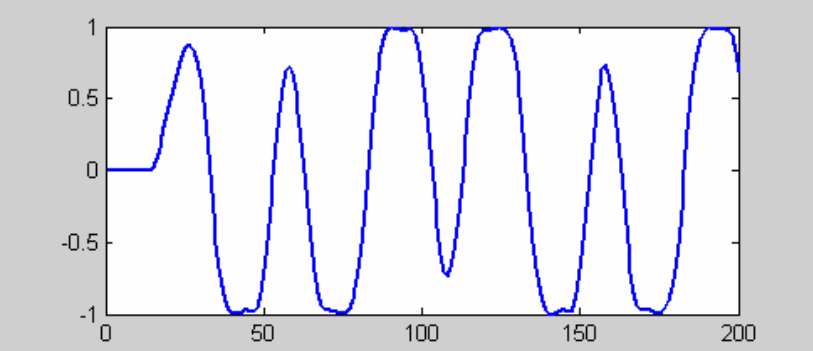


Şekil 3.8: Gürültülü olarak alınan IQ Bileşenleri

I ve Q değerlerini elde ettikten sonra, bir bit farksal sezim tekniği iletilen işaretten gerçek işareti elde etmeye çalışır. İletilen sembolleri elde etmek için, her bir bit periyodunda iletilen faz değişimlerini bulmak gereklidir. Faz farkı şöyle yazılabilir;

$$\Delta\phi(t) = \phi(t) - \phi(t - T_b) \tag{3.11}$$

Şekil 3.9 'da demodüle edilen dalga şekli gösterilmektedir.



Şekil 3.9: Demodüle Edilen İşaret

BÖLÜM 4. SAYISAL İŞARET İŞLEMCİ ve CODE COMPOSER STUDIO YAZILIMI

4. 1 Giriş

TMS320C6x ailesi sayısal işaret işlemciler hızlı özel amaçlı Texas Instruments firması tarafından üretilmiş işlemcilerdir. C6x gösterimi Texas Instruments firması tarafından tasarlanan sayısal işaret işlemcileri gösterir. C6x sayısal işaret işlemcisinin yapısı yoğun sayısal hesaplamalar için çok uygundur. C6x uzun komut kelimesi (very long instruction word - VLIW) sahip TI'nin en güçlü işlemcisidir. Sayısal işaret işlemciler haberleşmeden ses ve görüntü uygulamarına kadar geniş bir alanda uygulama alanına sahiptir. Modemler ve ses tanıma uygulamaları DSP teknikleri kullanılarak daha ucuza mal edilebilir. DSP 'nin kullanımı kolay,esnek ve ekonomiktir. Matematiksel işlemleri gerçekleştiren sayısal işaret işlemci, verileri ve program tanımlarını saklamak için hafıza, analog ve sayısal arasında dönüştürme yapabilmek için dönüştürücü modüllerini içerir.

DSP'de iki farklı mimari vardır, Von Neumann ve Harvard mimarisidir. Von Neumann mimarisi bilgisayar teknolojisindeki gelişmeler için bir standart olmuştur. Aslında mimari oldukça basittir. Program ve veri hafızada tanımlanmış alanda saklanır. Bu mimari genel amaçlı işlemci ihtiyaçları için taban oluşturur. Bu mimarinin dezavantajı, veri ve hafıza adreslerini paylaşan tek yol olmasıdır. Bu yüzden sadece veri alanı veya program alanına bir çevrimde bir zamanda ulaşılabilir.

Hızlı veri yönetiminin olduğu yerlerde program ve veri hafızasına tek bir çevrimde ulaşılabilmesi avantaj sağlar. Harvard mimarisi program ve hafıza alanlarını birbirinden ayırır. Her adres alanına hizmet veren iki yola sahip olmak, işlem hızını artıracak şekilde veri ve programa paralel olarak ulaşılmasını sağlar. Maalesef, işlemci gücü artışı yanında maliyet getirir. İki hafıza alanı, birçok adres için iki kat alana ve bu yüzden iki katı veri uçlarına ihtiyaç duyar. Fiyat ve performans bakımından en uygun çözüm modifiye edilmiş Harvard mimarisi sadece bir dış yola,

program ve veri için ayrı iki üç yola sahiptir. Texas firmasının sunduğu birçok DSP modifiye edilmiş Harvard mimarisini, kullanıcı için hızdan ödün vermeyerek maliyeti düşürecek şekilde destekler.

Programlanabilir DSP'ler matematiksel özelliklerine göre iki farklı gruba ayrılır; hareketli ve sabit noktalı. Her biri, bazı uygulamaları daha iyi çalıştıran ve bazılarında verimliliği düşen farklı mimariye sahiptir.

Sabit noktalı DSP'ler kesin ve sınırlı sayıda biti kapsayan bir alanda bir sayı sunar. Örneğin 6 bitlik bir işlemci $\pm 2^{15}$ alanını verir. İlk çıkan DSP'ler bu teknoloji üzerine oturtulmuştur ve bugünkü uygulamaların çoğunluğu için endüstri 6 bitlik sabit noktalı işlemcileri seçmektedir.

Kayan noktalı DSP'ler mantisa kullanılarak +1.0 ve -1.0 arasında sayılar verir. Ek olarak,sunumu üs olarak tanımlanan skalalı fonksiyonları içerir. Bu sunum metodu daha büyük bir dinamik alan verir ve bu şekilde işlem taşkınlığını azaltır. Değişken noktalı algoritmalar yüksek seviye dil derleyicilerini (MAC) optimize edecek şekilde uyarlar. 32 bitlik değişken noktalı işlemciler içindeki azaltılmış kuantalama hatası ses uygulamaları için idealdir.

4. 2 Dsk Destek Araçları

DSP uygulaması için şu araçlar gereklidir.

1. 1. TI DSP başlangıç kiti (DSP Starter Kit - DSK). DSK paketi şunları içerir;
   (a) Code Composer Studio (CCS), gerekli olan yazılım destek araçlarını içerir. C derleyici bileşenlerini içerir.
   (b) TMS320C6711 (C6711) kayan noktalı sayısal işaret işlemci giriş / çıkış için (I/O) 16 bit dönüştürücüye sahiptir .
   (c) DSK kartı ile PC arasındaki bağlantıyı sağlayan paralel kablo (DB25) vardır.
   (d) DSK kartı güç kaynağı içerir.
2. Paralel porta sahip PC

3. Osiloskop, işaret üreteci ve hoparlöre ihtiyaç vardır.

4. 3 Code Composer Studio (CCS) Yazılımı

Programlanabilir DSP'ler uygulama alanlarına iyimser çözümler getirirken, pazarlama için de kullanılan zamanı azaltan araçlarla, bugünün yazılım mühendislerine yol açmaktadır.

DSP'lerin gerçek güçlerinden yaralanabilmek için etkin yazılımlara ihityaç vardır. Aslında yazılım ürünün en kritik parçasıdır. Endüstriyel tahminlere göre yaklaşık olarak,çabaların %80 'i oturmuş sistemlerde yapılacak olan gelişmelerde ve sistem zorluklarının %80'ide yazılım da ortaya çıkmaktadır. Tasarımcılar karmaşık olaylar yerine, kullanılması kolay ve gelişmiş araçları olan işlemcileri sistemlerinde kullanmak istemektedirler. Bu tür araçlar günden güne pazarlama taleplerinde büyük bir etkiye sahiptir.

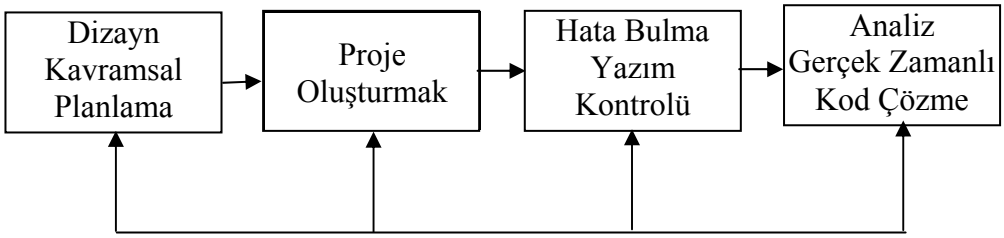
Geçmişteki tecrübelere göre, DSP araçları hem kod üretimini kolaylaştırmak için hem de kodların bölümlerini analiz etmek ve hata bulma işlemleri için uygun bir şekilde sınırlandırılmışlardır. Çoğu örnekte bu araçlar, geliştiricilerden istenen farklı uygulamalar arasındaki değişikliklerden yoksundur. Hata bulma gerçek zamanlı uygulamalar için bir engelleyici faktördür. Gerçek dünya koşulları altındaki uygulamaların davranışını anlamak için, geliştiriciler durmaksızın bir program akışını takip edebilmeyi isterler . Bunula beraber , tipik hata bulma sadece durma noktaları arasında tek tek , adım adım statik olarak hata bulunmasına izin verir . Bu programlar ardışık şekilde bilgi sağlayamazlar ya da kodun o anki uygulamasıyla , daha önceki olaylar arasında bir etkileşime sebep olabilirler .

Günümüz DSP araç setleri ; kod üretimi,basit hata bulma ve izleme gibi özellikler açısından ileri olmalıdır. DSP geliştirme sistemi için; ön sezgili, işlemciyi durdurmaksızın programın akışını izleme özelliği bulunması; anlaşılması zor zamanlarda problemlerini çözmek için gerçek zamanlı analizlerine sahip olması; bir yol içinde tasarımcıların sinyal verilerini incelemelerine izin veren grafiksel pencereler içermesi, bir editör yada derleyici kadar gereklidir.

Sonuç olarak tasarımcılar, bulundukları konumu muhafaza etmek ya da rakiplerine yetiştirmek için ürünlerine daha fazla özellik katmaya başlarken,DSP uygulamalarının karmaşıklılığı gittikçe artmaktadır.

Bu sonuçlar gösterir ki; eski günlerde DSP geliştiricileri sürekli gelişmiş sistemlere adapte olmaya zorlanmışken, şimdi ise kendi çalışacakları ortamı yaratacak ve gerekli olan araçları ekleme ihtiyacı içindedirler.

Tasarımcıların bu isteklerini karşılayan yazılımlar, pazarda çokbüyük paylar almaktadırlar. Texas firmasının ürettiği Code Composer Studio (CCS) bu ihtiyaçlara cevap vermektedir.
Code Composer Studio (CCS), hata bulma ve gerçek zaman analizleri gibi temel kod üretme araçlarına sahiptir. CCS şekil 4.1 'deki yazılım geliştrime döngüsündeki bütün evreleri sağlamaktadır.



Şekil 4.1: CCS Kod Geliştirme Evreleri

4. 4 Kod Üretme Araçları

Kod üretme araçları, CCS 'nin geliştirme ortamı için bir temel oluşturur. Şekil 4.2 'de tipik bir yazılım geliştirme akış diyagramı göstermektedir.

C Compiler, C kaynak kodunu alır ve assembly dilinde kaynak koduna çevirir.

Assembler, assembly dilindeki kaynak kodunu, makina dilindeki nesne dosyasına çevirir. Makina dili genel nesne dosya formatı (Common Object File Format – COFF) üzerine kuruludur.

Assembler Optimizer, kuyruk yapısıyla veya yazmaç tahsisi ile ilgilenmeden lineer assembly kod yazmamızı sağlar. Assembly optimizer yazmaç tahsisini yapar ve çevrim optimizasyonu yaparak lineer assembly'i, yüksek paralel assembly'e çevirir.

Linker, nesne dosyasını, çalışıtırılabilir tek nesne modüller haline getirir. Bu modülleri oluştururken, harici refaransları da çözüp yerleştirir. Linker giriş olarak COFF nesne dosyalarını ve nesne kütüphanelerini kabul eder.

Archiver, dosyaları kütüphane adı vererek bir arşiv altında toplar. Archiver aynı zamanda bu kütüphaneyi düzenlememizi sağlar.

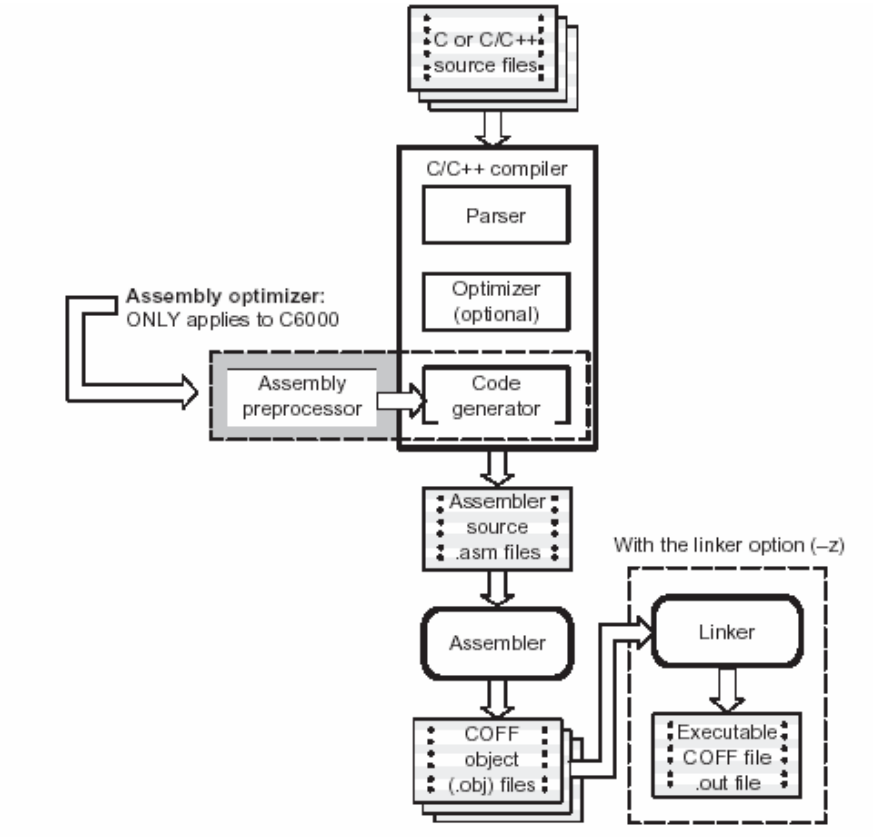
Library-build utility, kendi run-time destek kütüphanenizi oluşturmaya yarar.

Run-time Support Library, C derleyici tarafından desteklenen, ANSI standart run-time support fonksiyonları, compiler-utility fonksiyonları, kayan noktalı aritmetik fonksiyonlar ve I/O fonksiyonlarından oluşmaktadır.

Hex conversion utility, COFF nesne dosyasını TI-Tagged, ASCII Hex, Intel, Motorola-S yada Tektronix nesne formatına dönüştürür. Dönüştürülen dosya bir Eprom programlayıcısına yüklenebilir.

Cross-reference lister, nesne dosylarını, link edilmiş kaynak dosyasındaki cross-reference sembolleri, bunların tanımlamaları ve referansları için kullanılır.

Obsulate lister, link edilmiş nesne dosyalarını girdi alarak, çıktı olarak . abs dosyaları üretir.

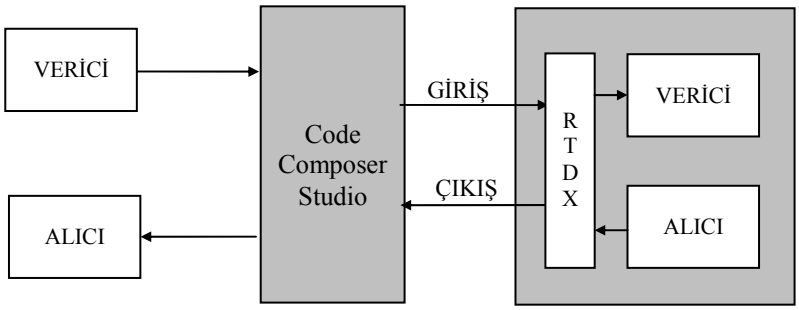


Şekil 4.2: CCS Akış Diyagramı

BÖLÜM 5. DSP ILE GMSK MODÜLASYON VE DEMODÜLASYON UYGULAMASI

Matlab'te simüle edilen algoritma, üzerinde değişiklikler yapılarak DSP işlemcide uygulaması yapıldı. Şekil 5.1'de GMSK modem uygulama blok yapısı gösterilmektedir.

Uygulama iki kısımdan oluşmaktadır. Birinci kısım DSK kartı üzerinde GMSK modülasyon ve demodülasyon işlemleri yapılmaktadır. İkinci kısım ise gerçek zamanlı veri alışveriş fonksiyonu ile DSK ile PC arasında modüle ve demodüle edilecek temel band işaretininin gönderilme ve alınma işlemlerinin yapılması, bu verilerin kullanıcı arayüzü ile grafik ekranda gösterilmesinden oluşmaktadır. Arayüz bölümünde ayrıca alınan ve gönderilen bitler arasında bit hata oranı hesaplanmaktadır.

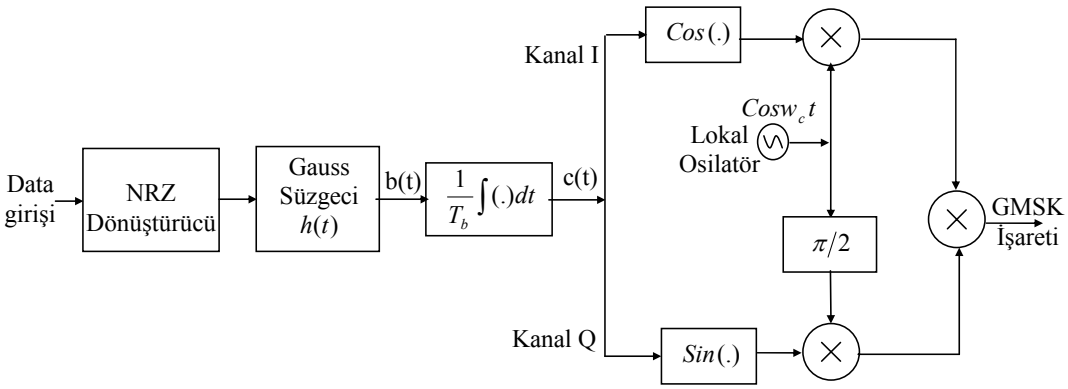


Şekil 5.1: RTDX ile GMSK uygulaması

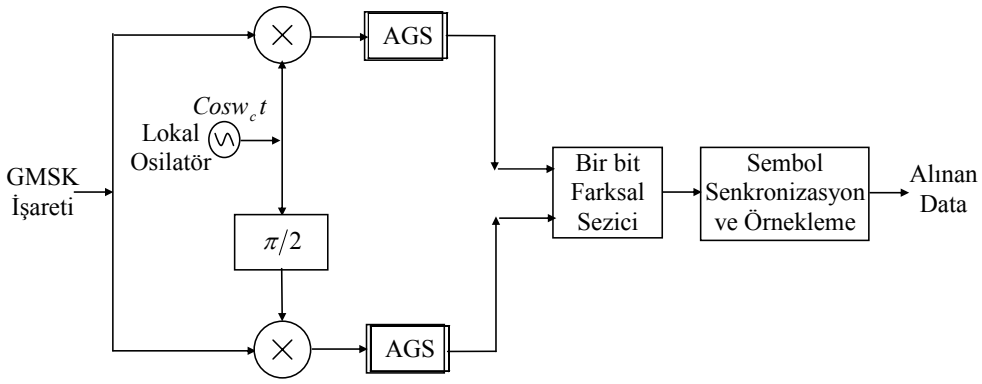
RTDX, DSP ile herhangi bir bilgisayar uygulaması ile haberleşme sağlayan bir arayüzdür. Bu arayüz ile DSP' den bilgisayara her türde (float, int , safearray v. b.) veri gönderilebilir veya veri alınabilir. Bu arayüz uluslararası standart olan JTAG ara yüzünü kullanır. JTAG arayüzü aslında her derleyicinin kod yüklerken ve debug yaparken kullandığı oldukça standart bir arayüzdür. Normalde 4 pin ile bu arayüz kurulabilir. RTDIN, RTDOUT, CE, Reset pinleri kullanılarak bu arayüzde veri aktarımı yapılır.

DSK kartı bilgisayara paralel port üzerinden bağlıdır. Burada veri gönderimi için paralel portun pinleri kullanılmaktadır. Code composer studio ve DSK kartı JTAG arayüzünü bu yapılar ile emüle etmektedir. Bu arayüzün pinleri veya kod yapısı kullanılarak bizden bağımsız sanki JTAG arayüzü varmış gibi Code Composer ve DSK kartı RTDX ara yüzünü kurmaktadır. RTDX, JTAG kullanır, DSK kartını hazırlayanlar karta fazladan JTAG konektörü koyarak %100 JTAG ara yüzü oluşturmak yerine paralel portun pinlerini kullanmışlardır.

C6000 cd' sini c:\ti altına kurulmalıdır. Kurulmadığı takdirde include ve kütüphane dosyalarının yerlerinin bulunamadığı hataları alınacaktır. Bu dosyalar ile ilgili düzenlemeler Project -> Build Options seçeneği altından düzeltilebilir.



Şekil 5.2: Eş-Fazlı ve Dik-Fazlı Taşıyıcılar ile GMSK Modülasyonu



Şekil 5.3: GMSK Demodülasyonu

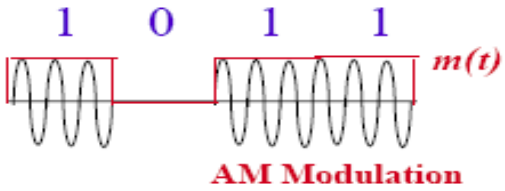
GMSK işleminin gerçeklenmesinde Şekil 5.2 ve Şekil 5.3 ‘deki blok yapıları göz önüne alınmıştır. GMSK DSP kodu içerisinde üç adet fonksiyonumuz vardır. Bu fonksiyonlar;

1. GMSK Modülasyon Fonksiyonu
2. GMSK Demodülasyon Fonksiyonu
3. Gerçek Zamanlı Veri Alışveriş Fonksiyonu
4. RTDX Fonksiyonu
5. Kullanıcı Arayüzü

Bu yazılımda “transmit” ve “receive” fonksiyonlarının her ikisi de bir tane short tipinde veriye bir tane short tipinde çıktı vermektedir. Örneğin 1=> biti için modülasyona işlemine başlandığında, bu 1 bitine tüm modülasyon işlemleri uygulanıp örneğin 340 gibi bir sayı elde edilir. Aynı şekilde “receive” tarafından tüm demodüle işlemleri sadece bir bit, örneğin alınan 340 sayısına karşılık 1 biti yeniden elde edilmesi işlemi yapılır. Modülasyon ve demodülasyon fonksiyonlarında her bir bit tek olarak modülasyon işlemine giriyor ve kanaldan demodülasyon işlemi sonucunda yeniden elde edilmektedir.

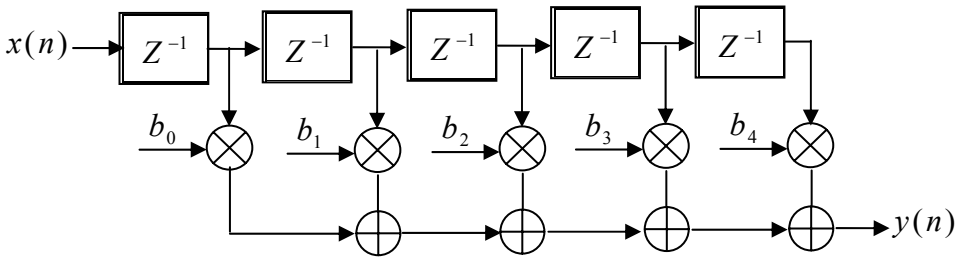
İkinci olarak yazılımda M kere aynı bit gönderiliyor. Pratikte gerçek uygulamalar da modüle edilen bir işaret kanala bir kez gönderilmez. Eğer kanala bir kez gönderildiğinde alıcıya ulaşması sırasında uğrayacağı bozulma (gürültü ve çevresel koşullar yüzünden oluşan bozulmalar v.b.) yüzünden alıcının algılama aralığına asla erişemez. Örneğin “1” bitine karşılık taşıyıcı ile elde edilen değer “340” gibi bir değer olsun ve bu “340” sayısını kanala gönderildiğinde karşı tarafa bu değer ulaşıncaya kadar çeşitli bozulmalardan dolayı değeri 100 ‘e düşebilir veya 1000’e çıkabilir. Bu durumları önlemek ve sinyalin ortalama değerini sabitleyebilmek için aynı sinyali belirli süreyle göndermek gerekir. Örneğin şekil 5.4’de basit ASK modülasyonu gösterilmiştir. Burada “1” sayısı için için üç adet sinüs işareti gönderilmiştir. Aynı işareti 3 kez gönderilmesinin sebebi işaretin ortalama gücünün sabit olarak kalmasını sağlamak içindir. Bu nedenle uygulamada M * Bit sayısı kadar “transmit ” fonksiyonunda modülasyon işlemi yapılmakta ve aynı bite karşılık

“receive ” fonksiyonunda demodüle işlemi yapılarak gönderilen bit yeniden elde edilmektedir.



Şekil 5.4: ASK Modülasyonu

Yazılımda kullanılan filtre tipi FIR filtre tipidir. Darbe biçimleyici ve gürültü gidermek için kullanılan filtreler için FIR filtre yapısı göz önüne alınmıştır. Şekil 5.5’de filtre için kullanılan FIR filtre yapısı gösterilmiştir.



Şekil 5.5: FIR Filtre Blok Diyagramı

Taşıyıcı işaret, sinus ve kosinüs tablosu kullanılarak üretilmiştir. Bu tabloyu kullanmak işareti üretmek için etkili bir yoldur. Bu tabloda taşıyıcının tek bir çevrim değeri kaydedilir ve bu tekrar ettirilerek sürekli taşıyıcı işareti oluşturulur.

5. 1 GMSK Modülasyon Fonksiyonu

Aldığı data üzerinde GMSK modülasyon işlemi yapıp sonucu “transmitted” değişkeni içinde saklar. Program kodu Ek B’de verilmiştir.

Yapılan hesaplamalarda kullanılan kosinüs değerleri için daha önceden hazırlanmış tablo değerleri kullanılmıştır. “cos” fonksiyonu ise DSP başlık dosyası “math. h” içindeki fonksiyondur. Değişkenler short olarak saklandığında 2 byte,float olarak saklandığında 4 byte yer tutmasından dolayı bazı işlemlerde “float” olan sayı tipi yerine “short” sayı tipi kullanılmıştır. Ayrıca “short” sayı tipi “float” sayı tipine

göre daha kolay işlenir ve saklanır. ”float” sayı tipi kullanmak yerine katsayılar 1000 ile genişletilerek “short” sayı tipi olarak işlenmiştir.

Tx_signal, O anki GMSK modülatör çıkış değeridir. Modülatör çıkışı;

$$\cos(2\pi f_c t) \times \cos(c(t)) + \sin(2\pi f_c t) \times \sin(c(t)) \quad (5.1)$$

olmalıdır(5.1). Aynı şekilde “costable“ ve “sintable“ tablolarıda 1000 sayısı ile genişletilerek hazırlandığından işlem sonunda tekrar float sayıya dönebilmek için 1000‘e bölünmüştür. Modülasyon sonucu elde edilen sonuç bilgisayara gönderilir.

5. 2 GMSK Demodülasyon Fonksiyonu

Parametresi ile belirtilmiş data modüle edilmiş data üzerinde GMSK demodülasyon işlemi yapıp sonucu bilgisayara gönderilmektedir. Alınan işaretten önce taşıyıcı bilgisinin çıkartılması işlemi yapılmaktadır.

Filtre çıkışında elde edilen değerler işlem kolaylığı için belli bir oranda küçültülür. DSP‘de bölme işlemi tek bir komut çevriminden büyük olduğu için bölme işlemi yerine sağa kaydırma işlemi kullanılır. Tek bir kaydırma işlemi 2’ye bölme işlemidir.

Alınan bite karar verme işlemi yapılmaktadır. Alınan işaretin ortalama gücü ve M/2 süreyle alınıp alınmadığına bakılarak bite karar verme işlemi yapılır. Ek B ,de demodülasyon fonksiyonu verilmiştir.

5. 3 Rtdx Fonksiyonu

RTDX yapısını kullanabilmek için öncelikle Code Composer Studio yazılımı içinde RTDX seçeneği etkin hale (Tools->RTDX->Configuration Tool) getirilmelidir. Bu işlem yapıldıktan sonra RTDX yapısına ait özellikler kullanılır hale gelecektir. RTDX, DSP işlemci ile Code Composer Studio yazılımı arasında bir yapıdır. DSP işlemci üzerinde yapıması gerekli işlemler vardır. Bu işlemler “GMSK.c” dosyasında yapılmıştır. DSP işlemci tarafında, üç adet “.c” kodumuz ve bir adet “.h” kodumuz

vardır. Bunların daha anlaşılır ve efektif olması için ayrı ayrı dosyalar olarak oluşturulmuştur .

Globals.h; Tüm global değişkenler bu başlık dosyası içinde extern olarak tanımlanmıştır. Bu sayede aynı değişkenler tüm ".c" uzantılı dosyalarda da tanımlı hale gelmiştir. Tüm fonksiyonların tanımlamaları bu dosyadadır. Tüm include edilen başlık dosyaları bu dosyadadır.

Globals.c; Extern edilmiş tüm değişkenler burada tanımlanmıştır. Bir global değişkenin projedeki tüm ".c" dosyalarında kullanılabilmesi için extern ile bir header dosyada tanımlanmalı ama bir adet projedeki herhangi bir "c" dosyasında tanımlamaları yapılmalıdır.

Drivers.c; Projede kullanılan tüm fonksiyonların bulunduğu yerdir.

Main.c; RTDX arayüzünü açıp kapatan veri alan ana döngünün olduğu ana işlevin yapıldığı yerdir.

RTDX' i yüklemek için öncelikle, RTDX için oluşan olayların yani veri gönderme alma yönteminin kesme kullanılarak mı yoksa fonksiyon içinde sürekli kontrol ederek mi kullanılıp kullanılmayacağının belirtilmesidir. Burada sürekli kontrol edilerek bir yapı oluşturulmuştur.

RTDX 'in aktif olarak başlayabilmesi için; Tools->RTDX -> Configuration Control penceresini açıp « Enable RTDX » denilmesi gerekmektedir. Aksi taktirde RTDX, herhangi bir yazılımla haberleşemez. Bu el ile yapılan bir rutindir. Otomatik olarak yapılan bir işlem değildir, code composer studio yazılım yapısından kaynaklanmaktadır.

Global.c dosyasında sadece global değişkenler tutulur. Create fonksiyonları "global.c" 'de çağrılır. Bunun nedeni kanal yaratan bu işlemlerin bir fonksiyon olarak değil bir bir makro olarak yazılmasıdır ve bu makro aslında bizim yapmamız gereken işi yani "input channel/output channel" türünden değişken tanımlayıp onu "create" etme işlemini yapmaktadır.

Yani CreateInputChannel'ı çağırdığımızda aslında biz bir fonksiyon çağırmıyoruz RTDX_inputChannel türünde bir değişken tanımlıyoruz. Bu bir değişken tanımı olduğu bunu "Global.c"'ye, yani tüm global değişkenlerin olduğu yerde olması faydalı olmaktadır.

RTDX' te kullanılan kanal isimleri çok önemlidir. Gerçekten bu kanal isimleri ile kanallar yaratılır. Örneğin bir uygulama bu kanallara erişmek isterse yine bu tanımlanan isimleri kullanmak zorundadır. Aksi taktirde tanımsız bir kanala erişmeye çalışır.

Bu noktada eğer Code composer ile debug yaparak kanalların yaratılıp yaratılmadığını görmek için derleyicide şu işlem yapılmalıdır. Tools-RTDX-Channel viewer Control penceresi açılır. Açılan bu pencere kanal durumlarını gösterir.

RTDX fonksiyonu ilk parametresi ile verilen kanaldan okuduğu/okuyacağı bilgiyi üçüncü parametresi ile verilen uzunlukta, ikinci parametresindeki değişkene yerleştirir.

Eğer kanalda bir veri yoksa burada bekler. Okuduğu anda program koşmaya devam eder. Yani program bu noktadan okuma yapmadan geçemez.

Ayrıca DSK kartındaki DSP' nin geçici belleği küçüktür. Eğer 400 byte' lık bir data yollanırsa ve sürekli sonucu almaya çalışırsak 400 * 15 = > 6000 = 4 Kb yer gerekir, bu da oldukça fazla bir değerdir. Böyle durumlarda sonuçları kart üzerinde harici olarak bulunan sdram' e yazmak gerekir, fakat bu işlemde artı iş yükü getirir.

5. 4 PC Uygulama Yazılımı

Burada RTDX ile haberleşme, haberleşme için Pc tarafında yapılan işlemler ele alınmaktadır. Öncelikle bilinmesi gereken bilgisayar tarafında gerçek zamanlı veriyi DSP 'nin belleğinde bir alandan okuyarak almadığıdır. Elbette yine bilgisayarda bir belleği okuyarak alması gerektiğidir. İşte böyle durumlarda yani bilgisayar gibi bir yapının DSK gibi bir yapıdan veri alması veya göndermesi için Windows ile gelen bu tip durumlar için haberleşme arayüzü (COM Interface) kullanılır.

Bu COM ara yüzü ortak bir bellek alanını, dosyayı kullanır, veriler COM ara yüze ait genel belleklerde veya dosyalarda saklanır, veriler okunacağı zaman bu dosyayı veya belleği okur, veriler yazılacağı zaman yine bu ara yüze ait yerlere yazılır. Burada bir senkronizasyondan söz etmek mümkün değildir. Biz gidip varolan verileri alınmış eski veya yeni verileri COM ara yüze ait yerlerden alınır veya bu alanlara yazılır. Aslında RTDX ile COM ara yüzü direkt haberleşmez. Code Composer , RTDX ile DSP'den veri çeker ve COM arayüze ait belleklere veriyi yazar veya buralardan okuyup RTDX ile veriyi DSP' ye gönderir. Yine Bilgisayarda aynı işlemi yapar.

RTDX ile Uygulama arayüzü kurulurken yapılması gerekli işlemler;

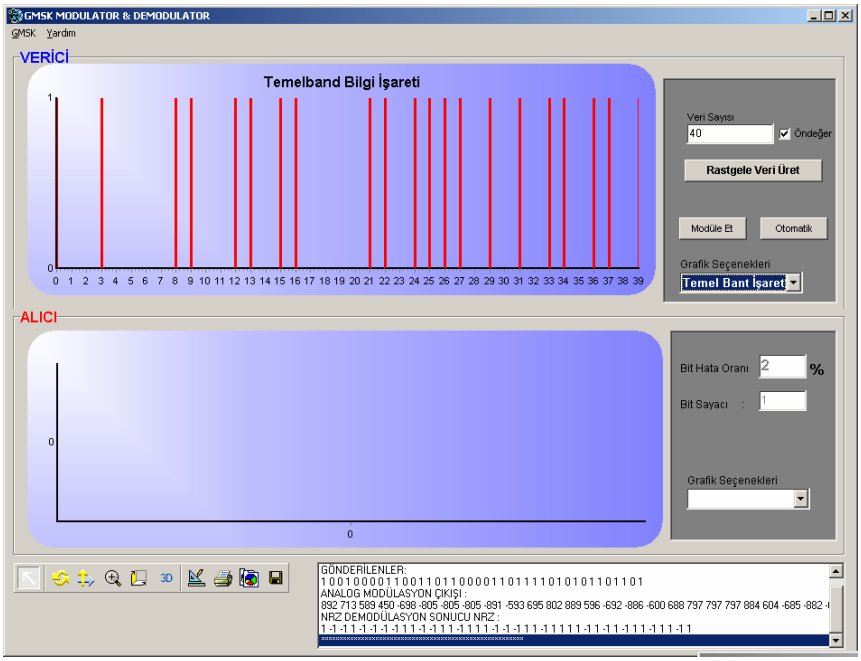
1. RTDX'e ait library projeye eklenmelidir (Borland' da Project -> Import Type Library -> eklenerek "rtdx. lib" kütüphane dosyası dosyası eklenir ve yükleme tuşuna basılarak sisteme yüklenmesi sağlanır.
2. Aşağıdaki hazırlıklar sırayla kod içerisinde yapılması gerekir.

Daha sonra giriş kanalı açılır. Aynı şekilde DSP tarafında olduğu gibi kanalın açılması gerekir. Artık RTDX değişkeni ile, RTDX ile kurulmuş tüm kanalları açılıp kapatılabilir.

Bilgisayar tarafında RTDX arayüzü kurulunca okuma ve yazmalar DSK'dan bağımsız olarak COM arayüze ait bellekten yapılır. Yani eğer DSK buraya yazma yapmışsa yazdığı veriler, yazma yapmamışsa başka veriler alınır. Bahsi geçen bellekte herhangi bir kartın veya yazılımın yazdığı tüm veriler tutulur. Yani geçmiş verilerede ulaşılabilir. İstenirse her seferinde kanal açıldıktan o ana kadar olan tüm verilere erişilebilir.

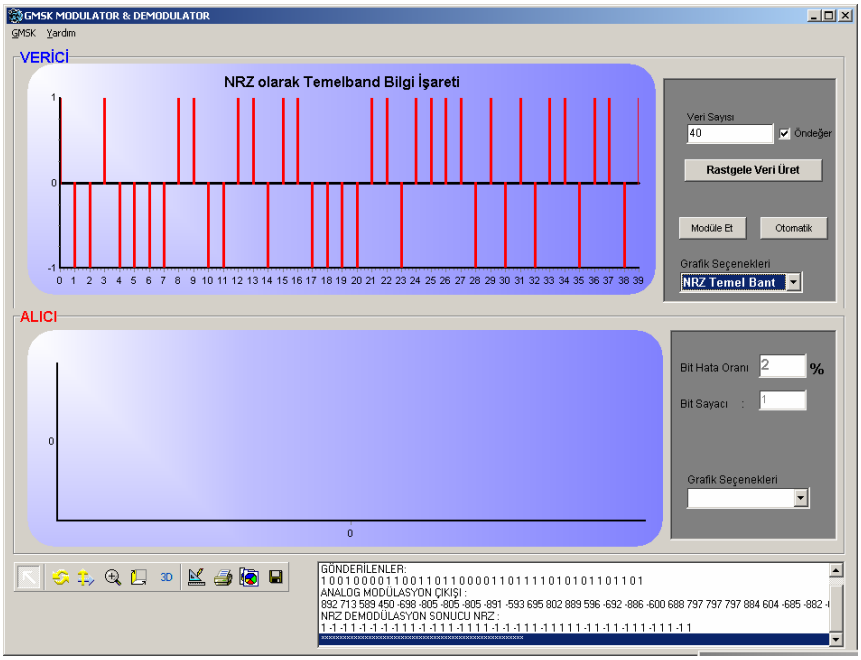
5. 5 Kullanıcı Arayüzü

İlk önce Code Composer Studio çalıştırılarak, "GMSK_fix. out" dosyası DSP yüklenir. RTDX seçeneği ile RTDX -> Enable seçilir. DSP kodu çalıştırılır. Bu işlemler yapılarak DSP veri almaya ve göndermeye hazır hale getirilir. Bu işlemlerden sonra PC tarafında "GMSK. exe" yazılımı çalıştırılır. Bu yazılım çalıştırıldığında aşağıdaki kullanıcı arayüzü ekrana gelir. (Şekil 5.6)



Şekil 5.6: Kullanıcı Arayüzü

Program açıldıktan sonra “Veri Sayısı” girilerek “Rastgele Veri Üret ” seçeneği tıklanarak tek kutuplu veri üretilir. Daha sonra üretilen tek kutuplu veri çift kutuplu sıfıra dönüşsüz hale çevrilir. Bunun sebebi, çift kutuplu verilerde bant genişliği, eşdeğer özellikteki tek kutuplu veriye göre daha düşük olduğu için daha verimli bir iletim sağlanmasıdır. Ayrıca tek bitlik bir hata algılama imkanına sahiptir. Tek kutuplu işaretler, çift kutuplu işaretlere göre iki kat bant genişliğine sahiptir.



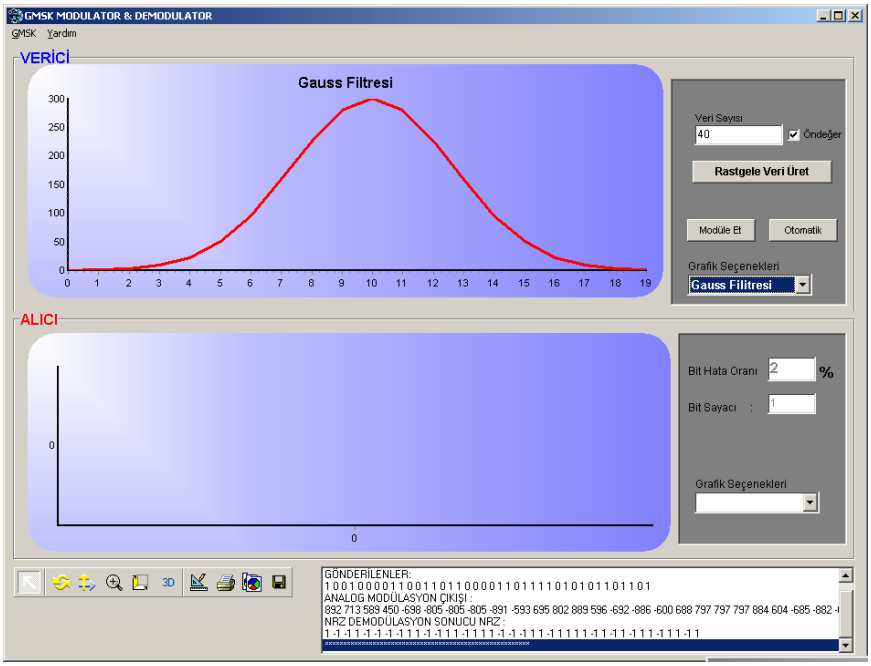
Şekil 5.7:NRZ Formda Temelbant Bilgi İşareti

Veri sayısı, öndeğer seçeneği işaretli ise başlangıç değeri olarak 40 değerine ayarlanmıştır. Öndeğer seçeneği seçilmeyerek istenilen sayı girilebilir. Maksimum değeri 450 bit'dir.

Grafik seçeneklerinden NRZ olarak temelband bilgi işareti seçildiğinde Şekil 5.7'daki ekran gelir. Burada rasgele üretilen veri NRZ formda çift kutuplu yani -1 ile +1 olarak gösterilir ve bu şekilde DSP 'ye gönderilir.

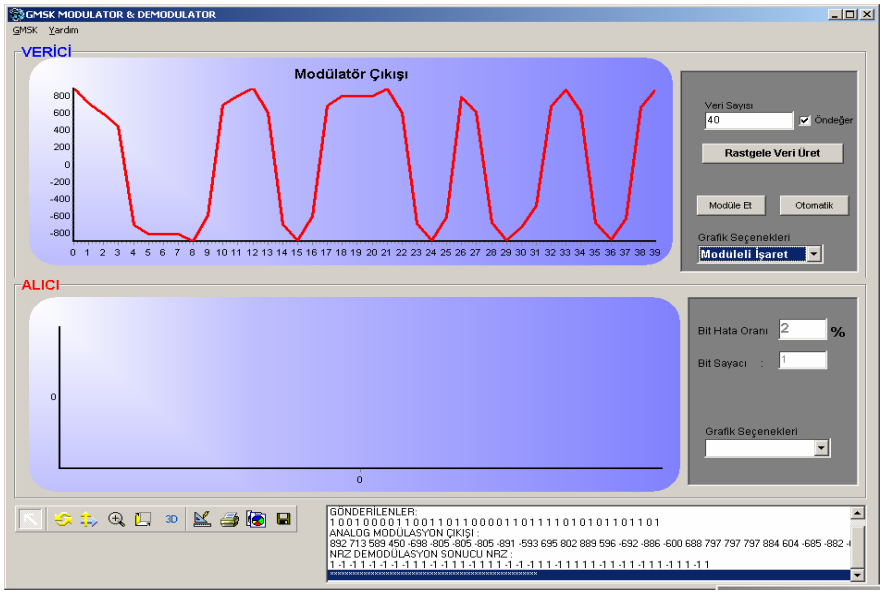
Rasgele veri üretildikten sonra "Modüle Et " butonuna tıklanarak, DSP'ye veriler gönderilir ve DSP'de işlenen veriler alınmaya başlar. COM arayüzü paralel port üzerinden olduğu için veri sayısı ne kadar yüksekse o kadar yavaş çalışacaktır .

Ayrıca grafik seçeneklerinde "Gauss Filtresi" seçildiğinde Şekil 5.8'deki Gauss Darbesi görülebilir.



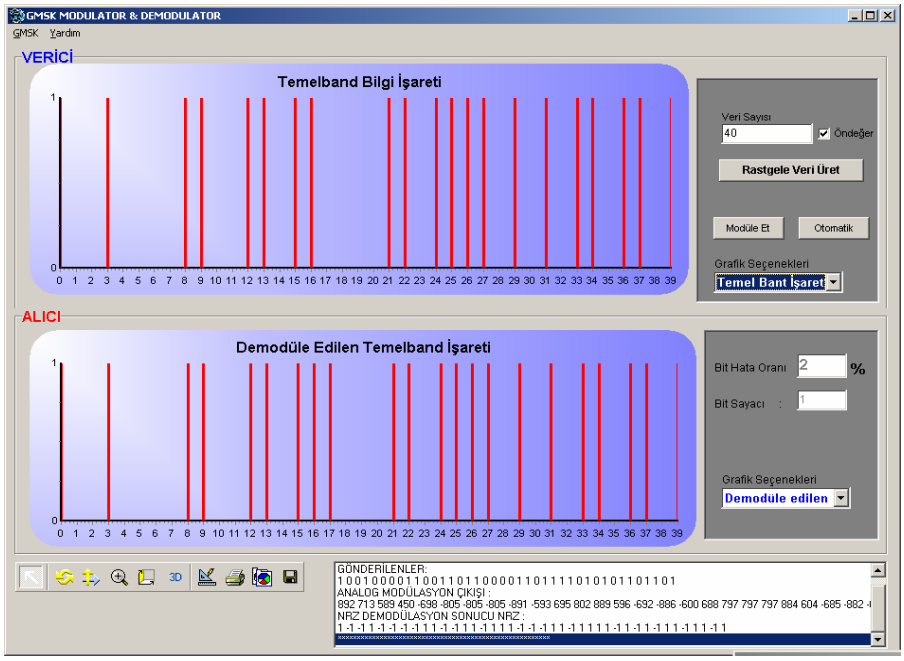
Şekil 5.8: Gauss Filtresi

Yine grafik seçeneklerinde "Modüleli İşaret" seçildiğinde de modülatör çıkışı görülecektir. (Şekil 5.9)



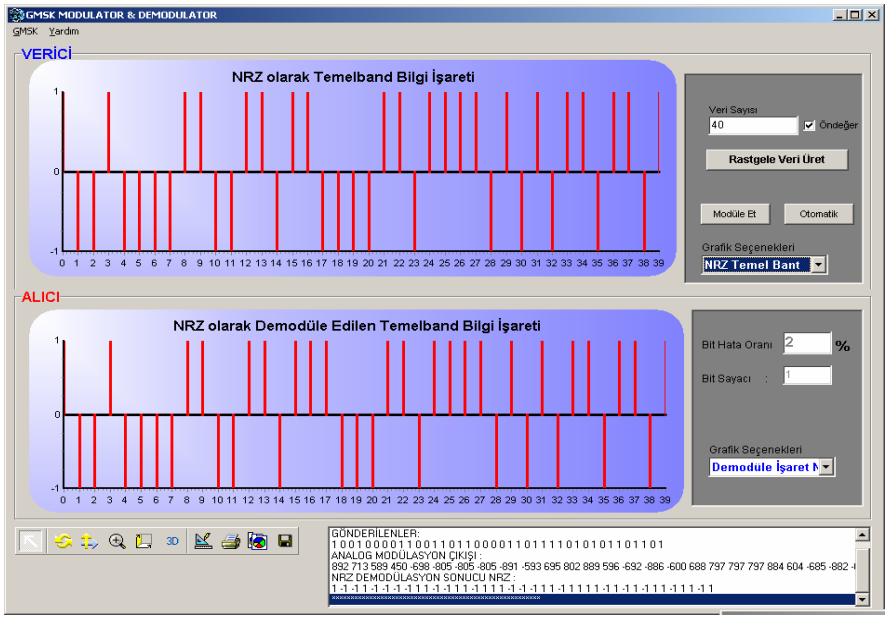
Şekil 5.9: Modülatör Çıkışı

Grafik seçeneklerinden "Alıcı" kısmında "Demodüle Edilen Temelbant İşareti" seçilerek DSP'den alınan verilere göre demodüle edilen temelbant işareti çizilir (Şekil 5.10).



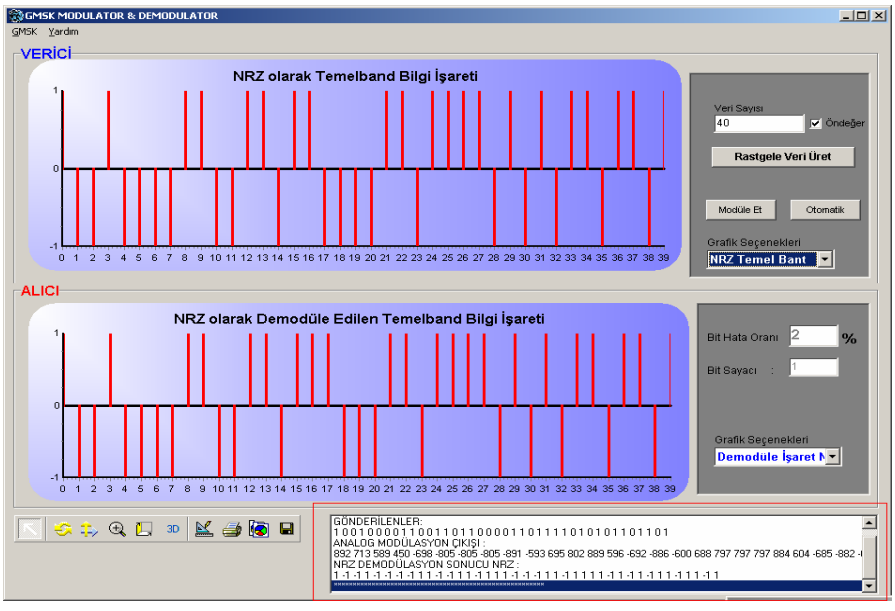
Şekil 5.10: Demodüle Edilen Temelbant işareti

Grafik seçeneklerinde "NRZ formda Demodüle Edilen Temelbant İşareti" seçilerek çift kutuplu olarak demodüle edilen işaret görülür. (Şekil 5.11)



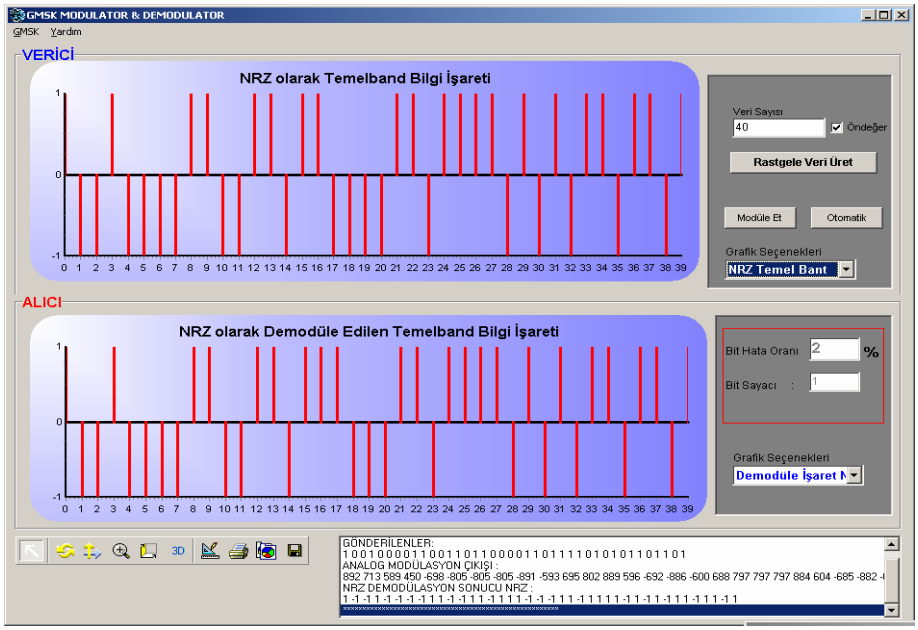
Şekil 5.11: NRZ olarak demodüle edilen temelband bilgi işareti

Ayrıca Şekil. 5.12'dan görüleceği üzere yazılım çalışırken yapılan işlem adımları, gönderilen bitler, analog modülasyon çıkış değerleri ve NRZ formda demodülasyon bitleri de görülebilmektedir.



Şekil 5.12: Gönderilen ve alınan bitlerin gösterilmesi.

Ayrıca alınan bitler ile gönderilen bitler karşılaştırılarak Şekil 5.13'de belirtilen "Bit Hata Oranı" yüzde olarak hesaplanır.



Şekil 5.13: Hesaplanan Bit Hata Oranının Gösterilmesi

BÖLÜM 6. PERFORMANS TESTİ

Modülatör çıkışına gürültü eklenerek işaret gürültü oranı parametresi değiştirilmesi ile performans değişiminin gözlenmesi amaçlanmıştır. Çeşitli işaret gürültü oranı değerlerine karşılık gelen toplamsal beyaz gauss gürültü değerleri elde edilmiştir. Bu değerler MATLAB yazılımındaki "y=wgn(m,n)" fonksiyonu ile elde edilmiş, daha sonra bu değerler DSP'de modülasyon sonucu elde edilen değerlere eklenerek sistemin çeşitli işaret gürültü oranlarında bit hata oranı hesaplanarak Tablo1'deki değerler elde edilmiştir. Bu değerler Matlab yazılımına girilerek şekil 6. 1'deki grafik elde edilmiştir.

Beyaz gauss gürültüsü özelliği taşıyan rasgele sayıları üretmenin birkaç yolu vardır. Bunlardan biri kolayca gerçekleştirilebilecek olan rasgele sayı üreteç algoritmalarının ürettiği sayıların dağılımının gauss yapılabilmesi için gerekli eklemelerin yapılmasıdır. Başka bir yol ise bu tip dağılımlı sayıların üretiminin çok kolay olduğu bazı programlar ile üretip, daha sonra bu üretilmiş olan sayıları çalışma sırasında kullanmaktır.

Yapılan pratik çalışmada, ilk olarak MATLAB programı kullanılarak, programın içerdiği "wgn" fonksiyonu ile istenen uzunlukta dağılımı gauss olan sayılar üretilmiştir. Daha sonra bu sayılar belli bir değere göre normalize edilmiş olup ayrıca DSP sabit noktalı aritmetiğine uygunlaştırılmak için kesirli sayı olarak ifade edilmiştir.

İşaret gürültü oranı işaret gücünün gürültü gücüne oranıdır ve genellikler desibel cinsinden gösterilir. Eşitlik 6. 1'de gösterilmiştir. Burada, P_{isaret} işaretin ortalama gücü, P_{isaret} gürültünün ortalama gücünü göstermektedir.

$$SNR(dB) = 10\log_{10}\frac{P_{isaret}}{P_{gürültü}} \tag{6.1}$$

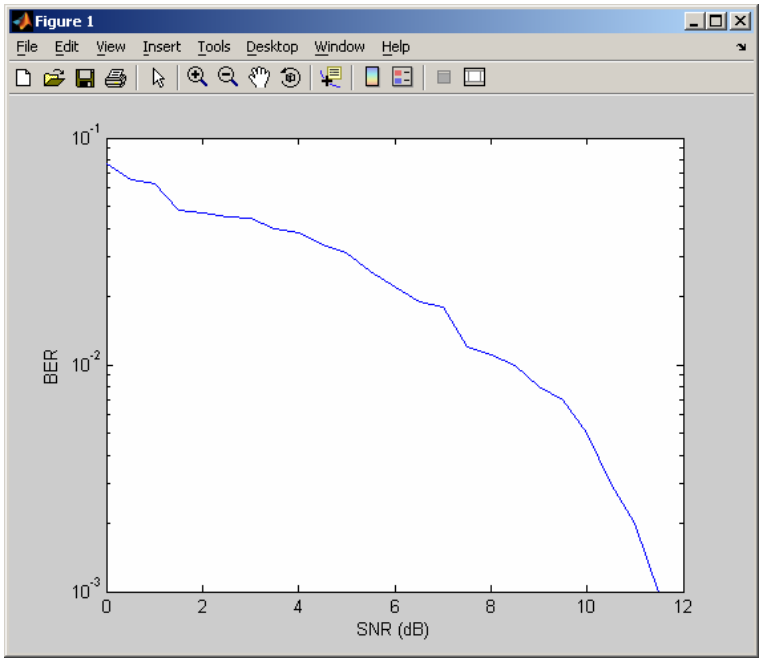
Bit hata oranı hesaplaması eşitlik 6. 2'de gösterilmektedir.

$$BER = \frac{HataliBitSayisi}{ToplamBitSayisi} \tag{6.2}$$

Çeşitli işaret gürültü oranlarına göre hesaplanan bit hata değerleri Tablo 1'de verilerek, ayrıca grafik olarak şekil 6.1 'de çizdirilmiştir.

Tablo 6.1: BER, SNR Değerleri

Örnek No	SNR (dB)	BER	Örnek No	SNR (dB)	BER	Örnek No	SNR (dB)	BER
1	0	0. 077	15	7	0. 018	29	14	0
2	0. 5	0. 065	16	7. 5	0. 012	30	14. 5	0
3	1	0. 063	17	8	0. 011	31	15	0
4	1. 5	0. 048	18	8. 5	0. 010	32	15. 5	0
5	2	0. 047	19	9	0. 008	33	16	0
6	2. 5	0. 045	20	9. 5	0. 007	34	16. 5	0
7	3	0. 044	21	10	0. 005	35	17	0
8	3. 5	0. 040	22	10. 5	0. 003	36	17. 5	0
9	4	0. 038	23	11	0. 002	37	18	0
10	4. 5	0. 034	24	11. 5	0. 001	38	18. 5	0
11	5	0. 031	25	12	0	39	19	0
12	5. 5	0. 026	26	12. 5	0	40	19. 5	0
13	6	0. 022	27	13	0	41	20	0
14	6. 5	0. 019	28	13. 5	0			



Şekil 6. 1: BER/SNR Grafiği

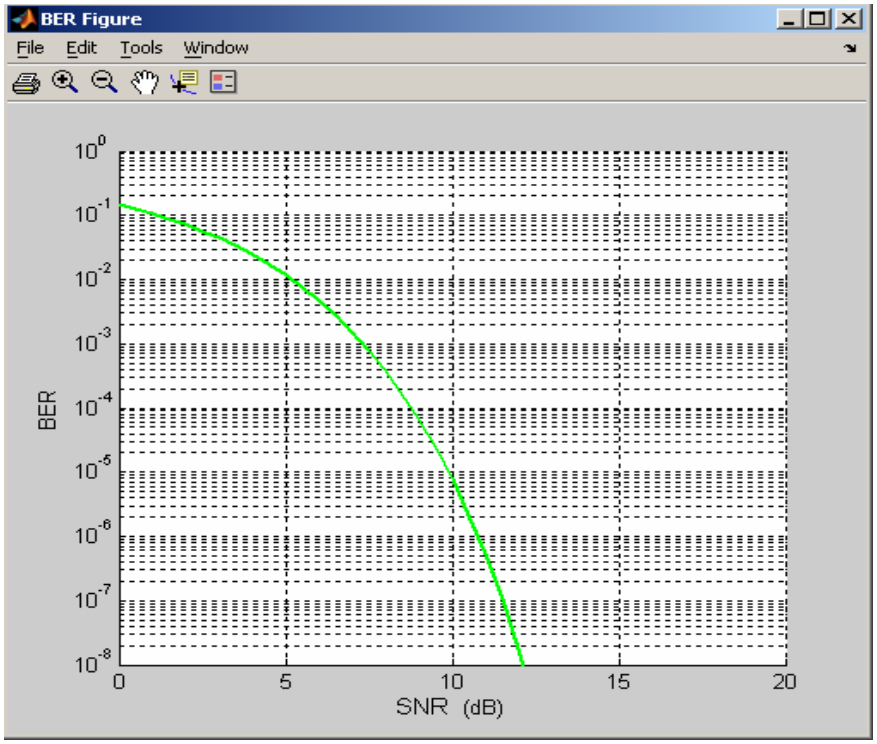
BÖLÜM 7. SONUÇ

Bu uygulamada GMSK modülasyon ve demodülasyon işlemleri sayısal bir işlemci olan TMS320C6711 işlemcisi kullanılarak tüm işlemler sayısal olarak gerçekleştirilmiştir. Sayısal teknikler ile, hata deteksiyonu ve düzeltmeler yapılarak son derece düşük hata oranları ile aslına uygun işaret elde edildiği gözlemlenmiştir.

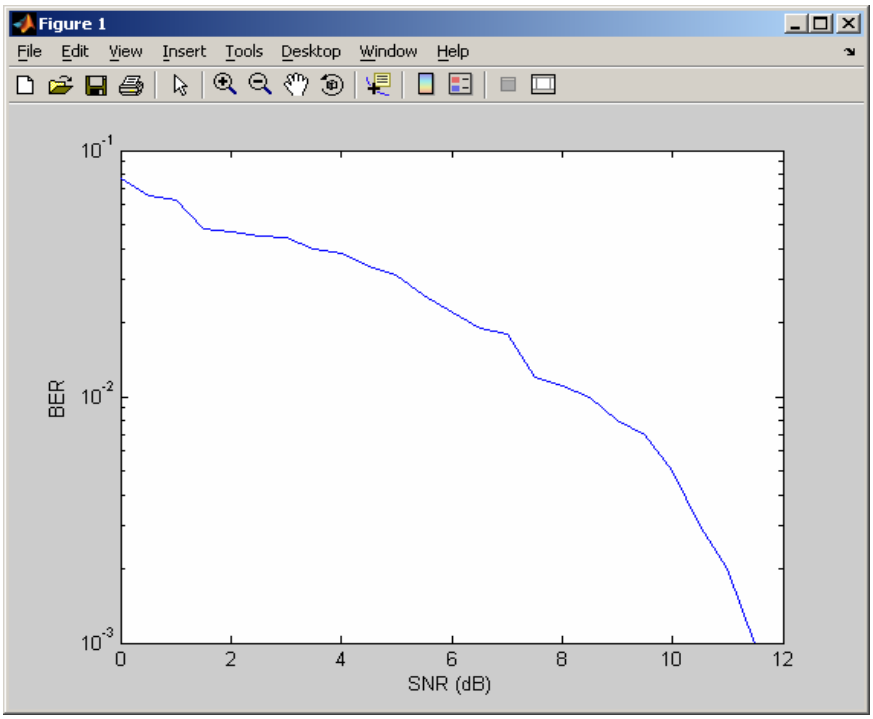
Sonuç olarak elde edilen işaret gürültü oranı değerlerine bakıldığında işaret gürültü oranı arttıkça hata oranın düştüğü görülmektedir. Uygulama sonucunda elde edilen değerlerin teorik sonuçlarla uyumlu olduğu ve sistemin başarılı çalıştığı gözlemlenmiştir.

Tezde elde edilen grafik, kaynak [3]'de verilen BER/SNR grafiği ile örtüşmektedir. Ayrıca bu kaynakta farklı SNR altında hesaplanan BER değerleride teorik olarak elde edilmesi gerekli BER değerleri ile yaklaşık olarak aynıdır. Ayrıca elde edilen sonuçlar ile [3]'deki çalışmanın sonuçları birbiri ile uyumlu bulunmuştur. (Şekil 7.1 ve Şekil 7.2).

Bu uygulama daha hızlı işlemcilerde uygulanarak daha iyi performans elde edilebilir. Ayrıca uygulama yapılan DSP program yazılımı içinde performansı artırmak amacı ile bazı kısımlar alt düzey programlama diline çevrilebilir.



Şekil 7.1 : Referans Alınan BER/SNR Grafiği



Şekil 7.2 : Elde Edilen BER/SNR Grafiği

KAYNAKLAR

[1] H. Harada, R. Prasad, "Simulation and Software Radio for Mobile Communications", ***Artech House***, May 2002, ISBN 1580530443

[2] Richard Paul Lambert, "A Real-Time DSP GMSK Modem with All-Digital Symbol Synchronization", M. S. Thesis, ***Texas A&M University***, May 1998

[3] Jeffery D. Laster, "Robust GMSK Demodulation Using Demodulator Diversity and BER Estimation" , Virginia Polytechnic Institute and State University, March 1997, Blacksburg,Virginia

[4] Floyd M. Gardner "A BPSK/QPSK Timing Error Detector for Sampled Receivers" ***, IEEE Transactions on Communications*** ,vol. 34. pp. 423-429, May , 1986

[5] Renfors, M. Babic, D. "Syncronisation in Digital Receivers" , ***ICE Tutorial*** , 83050E/284

[6] Digital Communications Fundamentals and Applications (Second Edition) By Benard Sklar. ***Prentice Hall,*** 2nd Edition ,2002

[7] M. P. Chen. "Detection improvement methods for a GMSK signal". ***Technical report,Mobile & Portable Radio Research Group***, Virginia Tech, 1996.

[8] Ertürk,S. ,"Sayısal Haberleşme",***Birsen Yayınevi***,2005

EKLER

EK A . Matlab Simülasyon Program Kodu

```
%*******************************************************
%Genel Tanımlamalar
%*******************************************************
%Gürültüyü kaldırmak için tasarlanan filtre
N = 16;
h = fir1(N,0. 16);
Rb = 200;          %Bit Rate (Bits/sec)
Tb = 1/Rb;         %Bit devam süresi
M = 10;            %Her bir bitde alınan örnekler
Ts = Tb/M;
%*******************************************************
%Gauss Filtre Dürtü Yanıtı
%*******************************************************
BT=0. 5;
B = 0. 5/Tb;        %Gauss Filtre Band Genişliği
n = -Tb:Tb/M:Tb;
k = pi*sqrt(2/log(2));
gauss = (k*B/sqrt(pi)) * (exp(-(k^2)*(B^2)*(n. ^2)));
gauss = round(gauss);
figure,plot(n,gauss),grid;
title('Gauss Darbesi');

%*******************************************************
%       GMSK Modem için İletim
%*******************************************************
num_bits = 20;
NRZ = ((rand(1,num_bits) > . 5) - . 5)*2 ;
for i=1:num_bits
   Bits((i-1)*M+1:(i)*M) = NRZ(i);
end
% Gauss Filtresinden geçirilen NRZ verisi
m = filter(gauss,1,Bits);
figure;
subplot(211),stairs(Bits);
title('NRZ data');
subplot(212),plot(m);
title('Gauss Filtrelenmiş Veri');
%I ve Q bulunarak verinin birleştirilmesi
int_m(1) = m(i);
for i=2:length(m)
   int_m(i) = int_m(i-1)+m(i);
end
```

```
int_m = (pi*Ts)/(2*M) * int_m;
figure, plot(int_m),title('Faz');

%I ve Q bileşenlerinin Hesaplanması
dI = cos(int_m);
dQ = sin(int_m);
figure;
subplot(211),plot(dI);
title('I Bileşeni');
subplot(212),plot(dQ);
title('Q Bileşeni ');
%I ve Q Bileşenlerine göre taşyıcının modüle edilmesi
fc = 8000;
ts = 1/(10*fc);
t = 0:ts:ts*(length(dI)-1);
tx_signal = cos(2*pi*fc*t). *dI - sin(2*pi*fc*t). *dQ;
figure,plot(tx_signal);
title('Modüle Edilen İşaret');

%*********************************************************%
%GMSK Alıcı
%*********************************************************
fc = 8000;
ts = 1/(10*fc);
t = 0:ts:ts*(length(tx_signal)-1);
I = cos(2*pi*fc*t). *tx_signal;
Q = -(sin(2*pi*fc*t)). *tx_signal;
%Alınan işarete AWGN gürültüsü eklenmesi
SNR = 50;
I = awgn(I,SNR,'ölçülen',4321,'dB');
Q = awgn(Q,SNR,'ölçülen',4321,'dB');

%Gürültünün kaldırılması için Filtreleme işlemi
I1 = filter(h,1,I);
Q1 = filter(h,1,Q);
figure;
subplot(211),plot(I1/max(abs(I1))),title('Alınan I ');
subplot(212),plot(Q1/max(abs(Q1))),title('Alınan Q ');

%İletilen işaretin detekte edilmesi
Z = I1 + Q1*j;
demod(1:length(Z)) = imag(Z(1:length(Z)) . * conj([zeros(1,M), Z(1:length(Z)-
M)]));
demod = demod/max(abs(demod));
figure;
subplot(211),plot(m);
title('Modüle edilen İşaret');
subplot(212),plot(demod);
title('Demodüle edilen işaret');
```

```
%Sembol Eş zamanlaması için Değişkenler
num = 1;
time = 2*M+5;
mu = 0. 04;
shift = 0;
%Sembol Eş zamanlaması ve orjinal örneklenmiş işaret
while(time <=length(demod))
   samples(num) = (demod(time) >0)*2-1;
   error = (sign(demod(fix(time)))-sign(demod(fix(time-M))))*(demod(time-M/2));
   shift = error*M*mu;
   steps = fix(shift);
   time = time + M-steps;
   num = num+1;
end
figure;
subplot(211),stem(NRZ),title('İletilen Veri '),
subplot(212),stem(samples),title('Alınan Veri ');
```

EK B . GMSK Modem DSP Program Kodu

```
////////////////////////////////////////////////////////////////////////
//
//        Ana Döngü
////////////////////////////////////////////////////////////////////////

#include "Globals. h"
void main()
{
        short wBitCount = 0;
        long status = 0;
        TARGET_INITIALIZE();                    // Enable RTDX interrupt
        RTDX_enableOutput(&D2A_channel);  // Enable the output channel
        RTDX_enableInput(&A2D_channel);    // Enable the output channel
        // SUPER LOOP
        while(1){
           // Receive PACKET_SIZE from host
           status = RTDX_read(&A2D_channel, &wBitCount, sizeof(wBitCount));
           if(status != sizeof(wBitCount)){
                        printf("ERROR: PACKET_SIZE : RTDX_read failed!\n");
                        exit(-1);
                }
        // Packet size received. Allocate memory for coming data from the host
                printf("Packet size receieved %d\n",wBitCount);
                //wBitCount = wBitCount+2;
                pData  =  (short *) malloc((wBitCount + 2)* sizeof(short));
                pTransmitted  =  (short *) malloc((wBitCount + 2) * sizeof(short));
                pOut      =  (short *) malloc((wBitCount + 2) * sizeof(short));
        if(!(pData && pOut && pTransmitted )){printf("ERROR:Allocation\n");
                exit(-1);
                }
                RTDX_write(&D2A_channel, &CMD_PACKET, sizeof(short));
                if(status == 0){
                        puts("ERROR: CMD_PACKET : RTDX_write failed!\n");
                        exit(-1);
                }
                CMD_PACKET = CMD_PACKET + 1;
                // Wait till writing complete
                while(RTDX_writing != NULL){
                        #if RTDX_POLLING_IMPLEMENTATION
                                RTDX_Poll();
                        #endif
                }
        // Receive data from host
           status = RTDX_read(&A2D_channel, pData, sizeof(short) * wBitCount);
           if(status != (sizeof(short) * wBitCount)){
```

```
                printf("ERROR: DATA : RTDX_read failed!\n");
                exit(-1); }

        //Data has just received. Use it in GMSK
        printf("Data Received\n");
        GMSK(pData, wBitCount + 2);
        //Say to host that we finisned the GMSK
        RTDX_write(&D2A_channel, &GMSK_END, sizeof(short));
        if(status == 0){
                puts("ERROR: GMSK_END : RTDX_write failed!\n");
                exit(-1);
        }
        GMSK_END = GMSK_END + 1;
        // Wait till writing complete
        while(RTDX_writing != NULL){
                #if RTDX_POLLING_IMPLEMENTATION
                        RTDX_Poll();
                #endif
        }
        // Send Demodulation result
        RTDX_write(&D2A_channel, pOut, sizeof(short) * wBitCount);
        if(status == 0){
                puts("ERROR: samples : RTDX_write failed!\n");
                exit(-1);
        }
        // Wait till writing complete
        while(RTDX_writing != NULL){
                #if RTDX_POLLING_IMPLEMENTATION
                        RTDX_Poll();
                #endif
        }
// Send Modulation result
RTDX_write(&D2A_channel, pTransmitted, sizeof(short) * wBitCount);
        if(status == 0){
                puts("ERROR: mod_result : RTDX_write failed!\n");
                exit(-1);
        }
        // Wait till writing complete
        while(RTDX_writing != NULL){
                #if RTDX_POLLING_IMPLEMENTATION
                        RTDX_Poll();
                #endif
        }
        // Free all used items
        free(pData);
        free(pTransmitted);
        free(pOut);
        wBitCount = 0;
        status = 0;
```

```
                ResetGlobals();
                puts("RX-TX Ok\n");
    }   }
////////////////////////////////////////////////////////////////////////////////
//
//      GMSK Alıcı ve Verici
//
////////////////////////////////////////////////////////////////////////////////

#include "Globals. h"
inline short sign(int _number)
{
        if (_number<0) return -1;
        else if(_number>0) return 1;
        return 0;
}
short Transmitter(short _data)
{
        int m = 0;
        short modulated = 0;
        int gaus_filtered_data = 0;
        float phase = 0. 0;
        short dI = 0;
        short dQ = 0;
        int i = 0;
        for (i = GAUSS_SIZE - 1;i>0;i--)
        {
                shift_data_gauss[i] = shift_data_gauss[i-1];
                // no need scaling Q12*Q12*21(~Q29)
                gaus_filtered_data = gaus_filtered_data +
gauss[i]*shift_data_gauss[i];
        }
        shift_data_gauss[0] = _data;
        gaus_filtered_data = gaus_filtered_data + gauss[0]*shift_data_gauss[0];

        m = m_prev + gaus_filtered_data;
        m_prev = m;

        phase = (pi*m)/(2. 0*M*fs);

        dI = Q12(cos(phase));
        dQ = Q12(sin(phase));

        modulated = (cos_table[bit_index] * dI - sin_table[bit_index] *dQ )>>12;

        pTransmitted[transmit_index] = modulated;

        return modulated;}
void Receiver(short _recv)
```

```
{
        int demod = 0;
        short error = 0;
        int shift = 0;
        int i = 0;

        short I = (cos_table[bit_index]*_recv)>>12;
        short Q = -(sin_table[bit_index]*_recv)>>12;

        int filtered_I = 0. 0;
        int filtered_Q = 0. 0;

        for (i = FILTER_SIZE - 1;i>0;i--)
        {
                shift_I[i] = shift_I[i-1];
                shift_Q[i] = shift_Q[i-1];

                // no need scaling Q12*Q12*17(~Q28)
                filtered_I = filtered_I + shift_I[i]*h[i];
                filtered_Q = filtered_Q + shift_Q[i]*h[i];
        }

        shift_I[0] = I;
        shift_Q[0] = Q;

        filtered_I = filtered_I + shift_I[0]*h[0];
        filtered_Q = filtered_Q + shift_Q[0]*h[0];

// filtere_I and Q is max Q28 before scaling. After scaling Q15 and demod will be
max Q30
        filtered_I = filtered_I>>13;
        filtered_Q = filtered_Q>>13;

        demod = ((filtered_Q * shift_filtered_I[0]) - (filtered_I*shift_filtered_Q[0]));

        for (i = 0;i<M-1;i++)
        {
                shift_filtered_I[i] = shift_filtered_I[i+1];
                shift_filtered_Q[i] = shift_filtered_Q[i+1];
        }
        shift_filtered_I[M-1] = filtered_I;
        shift_filtered_Q[M-1] = filtered_Q;

        pDemod[demod_index] = demod>>18;// demod is maximum Q30.

        if(d_time == t)
        {
                pOut[num] = (pDemod[demod_index] > 0);
```

```
                error = ((sign(pDemod[demod_index]) - sign(pDemod[demod_index -
M])) * (pDemod[demod_index - M / 2]));
                shift = (error * M * mu)>>24;
                d_time = d_time + M - shift;

                for(i = 0;i<shift+1;i++)
                        pDemod[i] = pDemod[demod_index - shift + 1];

                demod_index = shift;
                num++;
        }
        demod_index++;
}

void GMSK(short *_pInput, short _wBitCount)
{
        int i = 0;
        int j = 0;
        short wData = 0;
        short wTransmitted = 0;
        for (i = 0;i<_wBitCount;i++)
        {
                // 0 sa -1 yap
                wData = ((2*_pInput[i])-1);
                wTransmitted = 0;
                transmit_index = i;
                for (j = 0;j<M;j++)
                {
                        bit_index = j;
                        wTransmitted = Transmitter(wData);
                        Receiver(wTransmitted);
                        t++;
                }
        }
}
```

EK C . PC Program Kodu

```
/*****************************************************************
*/
/*                                                               */
/*    MAIN . CPP                                                 */
/*                                                               */
/*    GMSK  -Host Main Routines                                  */
/*                                                               */
/*    by Sevil 01/06/2007                                        */
/*****************************************************************/
#include <vcl. h>
#pragma hdrstop
#include "Main. h"
#include "Splash. h"
#pragma package(smart_init)
#pragma link "CandleCh"
#pragma link "RTDXINTLib_OCX"
#pragma resource "*. dfm"
TMainFrm *MainFrm;
#define SUCCESS                 0x00
#define FAILURE                 0x80004005
#define NODATAAVAIL             0x8003001E
#define ENDOFLOGFILE    0x80030002          // End of log file
#define DEFAULT_PACKET_SIZE 40
#define GAUSS_SIZE 20
#define CMD_DATA 0x0A
enum {GMSK_end = 'gm',cmd_packet = 'cm'};
short GMSK_END = GMSK_end;
short CMD_PACKET = cmd_packet;
// GLOBALS
short PACKET_SIZE;                          // Message length
short *data_TX;                             // Holds baseband info signal
datas
short *data_RX;                             // Holds demodulated signal datas
short *mod_result;                          // Holds modulated signal datas
short commander;
bool Modulation_Done = false;
char InputChannel[] = "A2D_channel";
char OutputChannel[] = "D2A_channel";
short Gaussian[] = 0,1,3,9,23,51,96,159,226,280,301,280,226,159,96,51,23,9,3,1,0};
class AutoThread : public TThread
{
private:
protected:
```

```
   void __fastcall Execute();
public:
   __fastcall AutoThread(bool CreateSuspended);
   void __fastcall PushTheButton();};
AutoThread *MsgThread = new AutoThread(true); // create and run the thread
//---------------------------------------------------------------------------
__fastcall TMainFrm::TMainFrm(TComponent* Owner)
   : TForm(Owner)
{}
//---------------------------------------------------------------------------
void __fastcall TMainFrm::FormCreate(TObject *Sender)
{
   if(Check_Default->Checked){
      Edit_DataNumber->Enabled = false;
      PACKET_SIZE = DEFAULT_PACKET_SIZE;
      Edit_DataNumber->Text = IntToStr(PACKET_SIZE);
   }
   // Generate immediately random array and show it on the screen
   Button_GenerateData->Click();
   ComboBox_ModGraphSelection->ItemIndex = 0;
}
void __fastcall TMainFrm::FormCloseQuery(TObject *Sender, bool &CanClose)
{
   CanClose = false;
   if(MessageDlg("Çıkmak istediğinizden emin misiniz!", mtWarning,
TMsgDlgButtons()  << mbYes << mbNo, 0) == mrYes){
      CanClose =  true;
      SplashFrm->Close();
   }
}
//---------------------------------------------------------------------------
void __fastcall TMainFrm::eXT1Click(TObject *Sender)
{
   Close();
}
//---------------------------------------------------------------------------
void __fastcall TMainFrm::About1Click(TObject *Sender)
{
   SplashFrm->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMainFrm::Check_DefaultClick(TObject *Sender)
{
   if(Check_Default->Checked){
      Edit_DataNumber->Enabled = false;
      PACKET_SIZE = DEFAULT_PACKET_SIZE;
      Edit_DataNumber->Text = IntToStr(PACKET_SIZE);
   }
   else{
```

```
      Edit_DataNumber->Enabled = true;
      Edit_DataNumber->Focused();
   }
}
/*****************************************************************
*MODULATION & DEMODULATION--
*****************************************************************/
// Generate Random datas to send
void __fastcall TMainFrm::Button_GenerateDataClick(TObject *Sender)
{
   if(Edit_DataNumber->Text == ""){
      ShowInfoMsg("Please enter a number !");
      return;    }
   delete []data_TX;
   delete []data_RX;
   delete []mod_result;
   PACKET_SIZE = (short)StrToInt(Edit_DataNumber->Text);
   data_TX = new short[PACKET_SIZE];
   data_RX = new short[PACKET_SIZE];
   mod_result = new short[PACKET_SIZE];
   srand(time(NULL));
   for(int i = 0; i < PACKET_SIZE; i++){
      data_TX[i] = rand() % 2;
      mod_result[i] = 0;
      data_RX[i] = 0;   }
   Button_Modulate->Hint = "Send generated datas";
   Modulation_Done = false;
   ComboBox_ModGraphSelection->ItemIndex = 0;
   ComboBox_ModGraphSelectionChange(Sender);
}
// Send & receive datas via RTDX
void __fastcall TMainFrm::Button_ModulateClick(TObject *Sender)
{
   Edit_DataNumber->Enabled = false;
   SendReceive();
   Edit_DataNumber->Enabled = true;
   Modulation_Done = true;
   ComboBox_ModGraphSelectionChange(Sender);
   ComboBox_DemodGraphSelectChange(Sender);
   ShowInfoMsg("*****************************************");
   // Calculate the Error rates and etc.
   int bit_errorcounter = 0;
   for(int i = 0; i < PACKET_SIZE; i++)
      if(data_RX[i] != data_TX[i])
         bit_errorcounter++;
   float Error_Rate = 100 * bit_errorcounter / PACKET_SIZE;
   Edit_ErrorRate->Text = FloatToStr(Error_Rate);
   Edit_BitCounter-> Text = IntToStr(bit_errorcounter);
         return ;
```

```
}
// Send & receive datas
void __fastcall TMainFrm::SendReceive(void)
{
   VARIANT sa,command;                    // Pointer to SAFEARRAY
   long i;                                // SAFEARRAY incrementer
   long status = 0;                       // Status of RTDX COM API calls

SAFEARRAYBOUND rgsabound[1];// Set the dimension of the SAFEARRAY
long bufferstate;                         // The state of the host's write buffer

   IRtdxExpPtr rtdx;                      // Pointer to IRtdxExp interface
   HRESULT hr;                           // Status of generic COM API calls
   String S;
        // PREPARE ALL STUFFS
::CoInitialize(NULL);                     // Initialize COM interface
   ::VariantInit(&sa);                    // Initialize VARIANT
   ::VariantInit(&command);               // Initialize VARIANT
hr = rtdx. CreateInstance(L"RTDX" );   // Instantiate the RTDX COM Object
   if(FAILED(hr)){
      ShowInfoMsg("Error: Instantiation failed!");
      return;
   }
//*************** PAKET BOYUTUNU GÖNDER   (WRITE) ************
// Open input channel for writing the packet size
        status = rtdx->Open(AnsiToOLESTR(InputChannel),
AnsiToOLESTR("W"));
        if(status != SUCCESS){
      ShowInfoMsg("Error: opening of input channel failed!");
                return; }
// Send packet size from host and check if receiving is OK
   rtdx->WriteI2(PACKET_SIZE, &bufferstate);
//******** PAKET BOYUTUNU ALDIĞINI SÖYLEMESİNİ BEKLE   (READ)*
status = rtdx->Open(AnsiToOLESTR(OutputChannel), AnsiToOLESTR("R"));
        if(status != SUCCESS){
             ShowInfoMsg("Opening output channel failed");
                return; }
commander = 1;
        while(commander != CMD_PACKET)
        {
                status = rtdx->ReadSAI2(&command);
                if (status == SUCCESS)
                {
                        long i = 0;
                        SafeArrayGetElement(command. parray,&i,&commander);
                }
        }
        CMD_PACKET++;
   ShowInfoMsg("DSK RECEIVED PACKET_SIZE");
```

```
//*************** PAKETİ GÖNDER    (WRITE) ********
status = rtdx->Open(AnsiToOLESTR(InputChannel), AnsiToOLESTR("W"));
        if(status != SUCCESS){
      ShowInfoMsg("Error: opening of input channel failed!");
                return; }
sa. vt = VT_ARRAY | VT_I2;
// Set the VARIANT to be a SAFEARRAY of 32-bit longs
        rgsabound[0]. lLbound = 0;   // Set the lower bound of the SAFEARRAY
        rgsabound[0]. cElements = PACKET_SIZE;
        sa. parray = SafeArrayCreate(VT_I2, 1, rgsabound);
        for(i = 0; i < (signed)sa. parray->rgsabound[0]. cElements; i++)
           hr = ::SafeArrayPutElement(sa. parray, &i, (short *)&data_TX[i]);
        status = rtdx->Write(sa, &bufferstate);              // Send data to the target
        if(status != SUCCESS){
      ShowInfoMsg("Error: Write failed");
           return;    }
   ShowInfoMsg("SEND ITEMS:");
   S = "";
   for(i = 0; i < PACKET_SIZE; i++)
           S += IntToStr(data_TX[i]) + " ";
   ShowInfoMsg(S);
//*************** SAMPLE' I AL *********************************
   Sleep(1000);
        status = rtdx->Open(AnsiToOLESTR(OutputChannel),
AnsiToOLESTR("R"));
        if(status != SUCCESS){
      ShowInfoMsg("Opening output channel failed");
                return; }
        commander = 1;
   while(commander != GMSK_END)
        {
                status = rtdx->ReadSAI2(&command);
                if (status == SUCCESS)
                {
                        long i = 0;
                        SafeArrayGetElement(command. parray,&i,&commander);
                }
        }
        GMSK_END++;
status = NODATAAVAIL;// read modulated results
        while(status == NODATAAVAIL)
                status = rtdx->ReadSAI2(&sa);
        if(status == (signed)FAILURE){
        ShowInfoMsg("Error: ReadSAI4 returned failure!");
                return; }
ShowInfoMsg("MODULATED RESULT:");
        S = "";
        for(i = 0; i < (signed)sa. parray->rgsabound[0]. cElements; i++){
                hr = ::SafeArrayGetElement(sa. parray, &i, (short*)&data_RX[i]);
```

```
        S += IntToStr(data_RX[i]) + " ";
        }
        ShowInfoMsg(S);

        status = NODATAAVAIL;
        while(status == NODATAAVAIL)
                status = rtdx->ReadSAI2(&sa);
        if(status == (signed)FAILURE){
        ShowInfoMsg("Error: ReadSAI4 returned failure!");
                return; }
ShowInfoMsg("DEMODULATED RESULT:");
   S = "";
        for(i = 0; i < (signed)sa. parray->rgsabound[0]. cElements; i++){
                hr = ::SafeArrayGetElement(sa. parray, &i, (short*)&mod_result[i]);
      S += IntToStr(mod_result[i]) + " ";
        }
   ShowInfoMsg(S);
        status = rtdx->Close();// Close the channel
    rtdx. Release();              // Release the reference to the COM object
   ::VariantClear( &sa );       // Clear Variant
   ::VariantClear( &command ); // Clear Variant
        ::CoUninitialize();     // Uninitialize COM
return ;}
void __fastcall TMainFrm::ComboBox_ModGraphSelectionChange(TObject
*Sender)
{
   switch(ComboBox_ModGraphSelection->ItemIndex)
   {
      // BASEBAND SIGNAL
      case  0:
         Chart_TX->Series[0]->Visible = true; // show FFT like graphic
         Chart_TX->Series[1]->Visible = false; // hide line graphic
         Chart_TX->Title->Caption = "BASEBAND INFORMATION SIGNAL";
         Chart_TX->Series[0]->Clear();
         Chart_TX->LeftAxis->Grid->Visible = false;
         Chart_TX->LeftAxis->Automatic = true;
         Chart_TX->LeftAxis->Increment = 1;
         // draw the baseband signal
         for(int i = 0; i < PACKET_SIZE; i++)
            Chart_TX->Series[0]->Add(data_TX[i], IntToStr(i), clRed);
      break;
      case 1:
         Chart_TX->Series[0]->Visible = true; // show FFT like graphic
         Chart_TX->Series[1]->Visible = false; // hide line graphic
  Chart_TX->Title->Caption = "BASEBAND INFORMATION SIGNAL AS NRZ";
         Chart_TX->Series[0]->Clear();

         Chart_TX->LeftAxis->Grid->Visible = true;
         Chart_TX->LeftAxis->Automatic = false;
```

```
            Chart_TX->LeftAxis->Maximum = 1;
            Chart_TX->LeftAxis->Minimum = -1;

            // draw the baseband signal as NRZ
            for(int i = 0; i < PACKET_SIZE; i++){
               if(data_TX[i] == 0)
                  Chart_TX->Series[0]->Add(-1, IntToStr(i), clRed);
               else
                  Chart_TX->Series[0]->Add(1, IntToStr(i), clRed);
            }
         break;
         // GAUSS Filtresi
         case 2:
            Chart_TX->Series[0]->Visible = false; // hide FFT like graphic
            Chart_TX->Series[1]->Visible = true; // show line graphic
            // prepare the graph to show baseband like signal ( just 1-0 )
            Chart_TX->Title->Caption = "GAUSS FILTER";
            Chart_TX->Series[1]->Clear();
            Chart_TX->LeftAxis->Grid->Visible = false;
            Chart_TX->LeftAxis->Automatic = true;
            // draw the gauss filter
            for(int i = 0; i < GAUSS_SIZE; i++)
               Chart_TX->Series[1]->Add(Gaussian[i], IntToStr(i), clRed);
         break;
         // MODULATED SIGNAL
         case 3:
            Chart_TX->Series[0]->Visible = false; // hide FFT like graphic
            Chart_TX->Series[1]->Visible = true; // show line graphic
            Chart_TX->Title->Caption = "MODULATOR OUTPUT";
            Chart_TX->Series[1]->Clear();
            Chart_TX->LeftAxis->Grid->Visible = false;
            Chart_TX->LeftAxis->Automatic = true;
            for(int i = 0; i < PACKET_SIZE; i++)
               Chart_TX->Series[1]->Add(mod_result[i], IntToStr(i), clRed);
         break;
      }
}
void __fastcall TMainFrm::ComboBox_DemodGraphSelectChange(TObject
*Sender){
   switch(ComboBox_DemodGraphSelect->ItemIndex)
   {
      case  0:
         Chart_RX->Series[0]->Visible = true; // show FFT like graphic
// prepare the graph to show baseband like signal ( just 1-0 )
         Chart_RX->Title->Caption = "DEMODULATED BASEBAND SIGNAL";
         Chart_RX->Series[0]->Clear();
         Chart_RX->LeftAxis->Grid->Visible = false;
         Chart_RX->LeftAxis->Automatic = true;
         Chart_RX->LeftAxis->Increment = 1;
```

```
            // draw the baseband signal as
            for(int i = 0; i < PACKET_SIZE; i++){
               if(data_RX[i] == 1)
                  Chart_RX->Series[0]->Add(1, IntToStr(i), clRed);
               else
                  Chart_RX->Series[0]->Add(0, IntToStr(i), clRed);
            }
         break;
         case 1:
            Chart_RX->Series[0]->Visible = true; // show FFT like graphic
Chart_RX->Title->Caption = "DEMODULATED BASEBAND INFORMATION
SIGNAL AS NRZ";
            Chart_RX->Series[0]->Clear();
            Chart_RX->LeftAxis->Grid->Visible = true;
            Chart_RX->LeftAxis->Automatic = false;
            Chart_RX->LeftAxis->Maximum = 1;
            Chart_RX->LeftAxis->Minimum = -1;
              for(int i = 0; i < PACKET_SIZE; i++){
               if(data_RX[i] == 1)
                  Chart_RX->Series[0]->Add(1, IntToStr(i), clRed);
               else
                  Chart_RX->Series[0]->Add(-1, IntToStr(i), clRed);
            }
         break;
         case 2:
         break;
      }
}
void __fastcall TMainFrm::ShowInfoMsg(String S)
{
   Memo_Msg->Items->Add(S);
   Memo_Msg->Selected[Memo_Msg->Items->Count - 1] = true;
}
void __fastcall TMainFrm::Button_AutoSendClick(TObject *Sender)
{
   if(Button_AutoSend->Caption == "Auto Send"){
      Button_AutoSend->Caption = "Stop Auto";
      MsgThread->Resume();
   }
   else{
      Button_AutoSend->Caption = "Auto Send";
      MsgThread->Suspend();
   }
}
// Auto Send random datas to DSK
void __fastcall AutoThread::Execute()
{
   bool TerminateMode;
   TerminateMode = Terminated;
```

```
    while(!TerminateMode){
        Synchronize(PushTheButton);
        Sleep(1);
        if(Terminated)
            TerminateMode = True;
    }
}
__fastcall AutoThread::AutoThread(bool CreateSuspended)
    : TThread(CreateSuspended)
{
    Priority = tpLowest;
}
void __fastcall AutoThread::PushTheButton(void)
{
    MainFrm->Button_GenerateData->Click();
    MainFrm->Button_Modulate->Click();
}
```

ÖZGEÇMİŞ

10. 04. 1979 tarihinde Kars'ta doğdu. Orta öğrenimini 1996 yılında Ümraniye Teknik Lisesi Bilgisayar Yazılım Bölümünde tamamladı. 2001 yılında Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği Bölümünden mezun oldu. 2001 tarihinden itibaren Tübitak-Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü'nde araştırmacı olarak çalışmaktadır. Evli ve bir çocuk annesidir.