EEM 437-SAYISAL İŞARET İŞLEME LABORATUARI
LAB 1: CCS VE TMS320C6713 DSK

1 Code Composer Studio

Code Composer Studio (CCS), gerçek-zaman sayısal işaret işleme uygulamaları için C proğramlama diline dayalı tümleşik bir geliştirme ortamıdır. Bünyesinde bir C derleyicisi, bir makina dili dönüştürücüsü ve ve bir bağlayıcı içermektedir. Grafiksel yetenekleri vardır ve gerçek-zamanda hata düzeltmeyi desteklemektedir.

C derleyicisi, `.c` uzantılı bir kaynak C proğramını derleyerek `.asm` uzantılı bir assembly kaynak dosyası oluşturur. Makina dili dönüştürücüsü, `.obj` uzantılı bir makina dili nesne dosyası oluşturmak için bir `.asm` kaynak dosyasını işler. Bağlayıcı, `.out` uzantılı çalıştırılabilir bir dosya oluşturmak için nesne dosyalarını ve nesne kütüphanelerini bağlar. Çalıştırılabilir bu dosya, Unix tabanlı sistemlerde yagın olan ve bazı sayısal işaret işlemci üreticilerinin kullandığı bir bağlı ortak nesne dosya formatını (COFF) temsil eder. Çalıştırılabilir bu dosya, sayısal işaret işlemciye yüklenebilir ve işlemci üzerinde doğrudan çalıştırılabilir. Bir başka laboratuarda C ve assembly proğramlarının kesişimi olan `.sa` uzantılı doğrusal assembly kaynak dosyası tanıtılacaktır. Doğrusal bir en iyileştirici, `.asm` uzantılı bir assembly dosyası oluşturmak için `.sa` uzantılı bu kaynak dosyasını (C derleyicisinin yaptığına benzer bir şekilde) en iyiler.

Bir Code Composer Studio projesi, çalıştırılabilir bir dosya oluşturmak için gerekli tüm dosyaları (veya tüm dosyalara bağları) içerir. CCS, farklı türden dosyaların bir projeye eklenmesine veya projeden silinmesine imkan veren çeşitli seçenekler sunar. Ayrıca, bir Code Composer Studio projesi çalıştırılabilir bir dosya oluşturmak amacıyla dosyaların tam olarak nasıl kullanılacağı bilgisini içerir. Derleyici/bağlayıcı tercihleri belirtilebilir. Proğram durdurma noktaları oluşturma ve değişkenleri izleme, hafıza, kütükler ve karışık C ve assembly proğramı görüntüleme, sonuçları çizdirme ve proğram çalışma süresini izlemeyi içeren çeşitli hata düzeltme özellikleri mevcuttur. Bir proğram çeşitli şekillerde çalıştırılabilir.

CCS'nin gerçek-zaman veri alışveriş (RTDX) birimi kullanılarak gerçek-zaman analiz yapılabilir. RTDX, sayısal işaret işlemciyi durdurmadan bir bilgisayar ve üzerinde işlemciyi bulunduran geliştirme kartı (DSK) arasında veri alışverişine imkan verir. RTDX'in kullanımı başka bir laboratuarda anlatılacaktır.

Bundan sonraki tartışmalarda, CCS dosyalarının `c:\CCStudio_v3.1` klasöründe olduğu varsayılacaktır. Farklı uzantılı çok sayıda dosyalarla çalışacağız. Bu dosyalar aşağıda listelenmişir:

1. `dosya.pjt`: dosya adında bir proje
2. `dosya.c`: C kaynak dosyası
3. `dosya.asm`: assembly kaynak proğramı
4. `dosaya.sa`: doğrusal assembly kaynak proğramı
5. `dosya.h`: başlık destek dosyası
6. `dosya.lib`: kütüphane dosyası
7. `dosya.cmd`: kısımları hafızaya dönüştüren bağlayıcı komut dosyası
8. `dosya.obj`: nesne dosyası
9. `dosya.out`: çalıştırılabilir dosya
10. `dosya.cdb`: DSP/BIOS kullanırken konfigürasyon dosyası

2 DSK'nin Hızlı Testi (Güç Verildiğinde ve CCS Kullanılırken)

Güç verildiğinde, kart üzerindeki flash bellekte saklı bir güç-açık testi (POST) DSK'yı test etmek için kart destek kütüphanesinden (BSL) rutinler kullanır. Bu proğram için `post.c` adındaki kaynak dosyası, `c:\CCStudio_v3.1\examples\dsk6713\bsl\post` klasöründe saklıdır. Proğram dahili, harici ve flash belleği, iki adet çok-kanallı tamponlanmış seri portu (McBSP), doğrudan bellek erişimini (DMA), kart üzerindeki AD/DA dönüştürücüleri ve LED'leri test eder. Tüm testler başarılıysa, dört LED'in hepsi üç kez yanıp södükten sonra tüm LED'ler yanık olarak durur. AD/DA dönüştürücülerin testi esnasında 1 saniye süresince 1 kHz'lik sinüzoidal bir işaret üretilir.

CCS proğramını çalıştırdıktan sonra, *Debug* → *Connect* seçiniz. CCS penceresinin sol alt köşesinde (birkaç saniyeleğine) "The target is now connected" gözükür. *GEL* → *Check DSK* → *Quick Test* seçiniz. Hızlı (Quick) test doğru yükleme ve çalışmanın teyit edilmesinde kullanılabilir. CCS içerisinde yeni bir pencerede aşağıda verilen şekilde bilgi gözükmelidir:

Switches:15 Board Revision:2 CPLD Revision:2

Switches etiketinden sonra gözüken rakam, DSK'nın kenarındaki dört DIP anahtarın konumunu yansıtır. 15 değeri, tüm anahtarların yukarı konumda olduğu anlamına gelir. Anahtarları ikili taban gösterilimiyle $(1110)_2$ konumuna, yani ilk üç anahtarı (0,1,2) yukarı, dördüncü anahtarı (3) aşağı konuma getirin. Tekrar *GEL* → *Check DSK* → *Quick Test* seçiniz ve bu kez rakamın 7 ("Switches:7") olduğunu doğrulayınız. Dört anahtarla temsil edilen değeri 0 ile 15 arasında değiştirebilirsiniz. DSK üzerinde çalışan proğramlar DIP anahtarların durumunu test edebilir ve duruma göre davranabilir. Board Revision ve CDLD Revision etiketlerinden sonraki rakamlar, DSK'nın türüne ve sürüm sayısına bağlıdır.

3 DSK Araçlarının Testi İçin Proğramlama Örnekleri

CCS ve DSK'nın bazı özelliklerini göstermek amacıyla üç proğramlama örneği tanıtılacaktır. Bu örneklerin amacı, öğrencinin laboratuar boyunca kullanılacak yazılım ve donanım araçlarını tanımasını sağlamaktır. Diğer deneyleri yapmadan önce, bu üç örneği tamamlamanız tavsiye edilir.

Örnek 1: DIP Anahtar Kontrolüyle Sekiz Nokta Kullanarak Sinüs Üretilmesi (sine8_LED)

Bu örnek, tablodan okuma yöntemiyle sinüzoidal bir analog çıkış dalgaşekli üretmektedir. Daha önemlisi, kaynak dosyalarının değiştirilmesi, bir proje oluşturulması, proğram geliştirme araçlarına erişilmesi ve bir proğramın C6713 işlemcisi üzerinde çalıştırılması amacıyla CCS'nin bazı özelliklerini tanıtacaktır. C kaynak dosyası `sine8_LED.c`, Şekil 1'de verilmiştir.

Proğramın Açıklanması

Proğramının çalışması şöyledir. 8 adet 16-bit işaretli tamsayılardan oluşan `sine_table` adında bir dizi oluşturulur. Sinüzoidal işaretin bir periyodundaki sekiz örneği içerecek şekilde başlangıç değeri verilir. $i = 1, 2, \ldots, 7$ olmak üzere, `sine_table[i]` değeri $100sin(2\pi i/8)$'e eşittir. *main()* fonksiyonunun içinde *comm_poll()*, *DSK6713_LED_init()* ve *DSK6713_DIP_init()* fonksiyonlarına yapılan çağrılar DSK'yı, DSK üzerindeki AIC23 AD/DA dönüştürücüyü ve C6713 işlemcisindeki iki adet çok-kanallı tamponlanmış seri portu (McBSPs) başlangıç konumuna getirir. *comm_poll()* fonksiyonu `c6713dskinit.c` dosyasında tanımlıdır, *DSK6713_LED_init()* ve *DSK6713_DIP_init()* fonksiyonları, `dsk6713bsl.lib` kart destek kütüphane dosyasında (BSL) mevcuttur.

main() fonksiyonu içindeki `while(1)` proğram satırı, sonsuz bir döngü oluşturur. Döngü içinde 0 nolu DIP anahtarın konumu test edilir ve konumu aşağıya doğruysa 0 nolu LED yanar ve tablodan bir örnek dışarıya gönderilir. Aksi halde, 0 nolu LED söndürülür. 0 nolu DIP anahtarın konumu aşağıya doğru olduğu sürece, `sine_table` dizisinden okunan örnek değerleri dışarıya gönderilir. AIC23 ADC/DAC'nin sol kanalı, LINE OUT ve HEADPHONE soketleri aracılığıyla sinüzoidal bir analog çıkış dalgaşekli üretilecektir. Her defasında `sine_table` dizisinden bir örnek okunup `gain` değişkeni ile çarpıldıktan ve ADC/DAC'ye yazıldıktan sonra, dizinin hangi elemanıyla çalışıldığını belirten `loopindex` değişkeni bir arttırılır ve değişkenin değeri, dizi için izin verilen aralığı (LOOPLENGTH-1) geçtiğinde sıfıra eşitlenir.

c6713dskinit.c kaynak dosyasında tanımlı `output_left_sample()` fonksiyonu, dışarıya bir değer göndermek amacıyla her çağrıldığında, 8 kHz hızında dışarıya örnek gödermek amacıyla `comm_poll()` fonksiyonuyla ilk değerlendirilen ADC/DAC bir sonraki örnek için hazır oluncaya kadar bekler. Bu yolla, 0 nolu DIP anahtarın konumu aşağıyken anahtar 8 kHz hızında test edilecektir. ADC/DAC'nin örnekleme

```
//sine8_LED.c  sine generation with DIP switch control

#include "dsk6713_aic23.h"          //codec support
Uint32 fs = DSK6713_AIC23_FREQ_8KHZ;   //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC;//select input
#define LOOPLENGTH 8
short loopindex = 0;                //table index
short gain = 10;           //gain factor
short sine_table[LOOPLENGTH]={0,707,1000,707,0,-707,-1000,-707}; //sine values

void main()
{
  comm_poll();                      //init DSK,codec,McBSP
  DSK6713_LED_init();       //init LED from BSL
  DSK6713_DIP_init();       //init DIP from BSL
  while(1)                //infinite loop
  {
    if(DSK6713_DIP_get(0)==0)    //=0 if DIP switch #0 pressed
    {
      DSK6713_LED_on();           //turn LED #0 ON
      output_left_sample(sine_table[loopindex++]*gain); //output sample
      if (loopindex >= LOOPLENGTH) loopindex = 0; //reset table index
    }
    else DSK6713_LED_off(0);    //turn LED off if not pressed
  }                                   //end of while(1) infinite loop
}                   //end of main
```

Şekil 1: DIP anahtar kontrolüyle sekiz nokta kullanılarak sinüs dalgası üretme proğramı (`sine8_LED.c`).

frekansı aşağıda verilen proğram satırıyla ayarlanır:

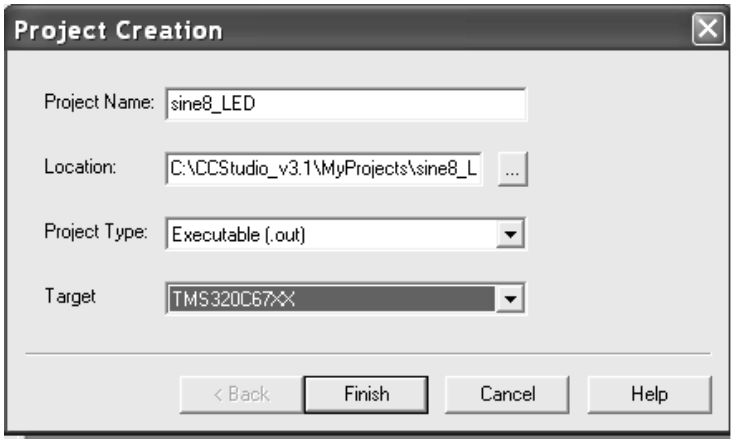
```
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
```

Sinüzoidal analog çıkış dalgaşeklindeki bir çevrim, 8 çıkış örneğine karşılık gelir. Dolayısıyla, sinüzoidal analog çıkış dalgaşeklinin frekansı ADC/DAC örnekleme frekansının (8 kHz) sekizde biri, yani 1 kHz'dir.

Bir Proje Oluşturma

Bu kısımda, bir projenin nasıl oluşturulacağı ve çalıştırılabilir bir dosya üretmek için gerekli dosyaların nasıl ekleneceği açıklanacaktır. Laboratuardaki bilgisayarlarda `sine8_LED` klasöründe `sine8_LED.pjt` isminde uygun bir proje vardır. Ancak, CCS'ye aşina olmak amacıyla, bu projenin sıfırdan nasıl oluşturulacağı anlatılacaktır.

1. `c:\CCStudio_v3.1\myprojects\sine8_LED` klasöründeki `sine8_LED.pjt` proje dosyasını siliniz. Bu işlemi CCS'nin dışında yapınız.
2. CCS proğramını başlatınız.
3. *Project* → *New* seçiniz. Çıkan pencerede, Şekil-2'de gösterildiği gibi proje ismi olarak `sine8_LED` yazarak `sine8_LED.pjt` isminde yeni bir proje dosyası oluşturunuz. *Finish* üzerine tıklamadan önce Target alanını TMS320C67XX seçeneğine ayarlayınız. Yeni proje dosyası, `sine8_LED` klasöründe saklanacaktır. `.pjt` dosyası, derleme seçenekleri, kaynak dosya isimleri ve bağlılıklar hakkında proje bilgisini saklamaktadır. Bir projenin kullandığı dosyaların isimleri, Code Composer penceresinin sol tarafında gözüken *Project View* penceresinde gösterilmektedir.
4. `sine8_LED.c` kaynak dosyasını projeye ekleyiniz. `sine8_LED.c`, `main()` fonksiyonunun tanımını içeren en üst seviye C kaynak dosyasıdır. Bu kaynak dosyası `sine8_LED` klasöründe saklıdır ve çalıştırılabilir `sine8_LED.out` dosyasını oluşturmada kullanılacaksa projeye eklenmesi gereklidir.



Şekil 2: sine8_LED için CCS proje oluşturma penceresi.

Project → *Add Files to Project* seçiniz ve çıkan pencereden `*.c`, `*.ccc` uzantılı C kaynak dosyalarını aratınız. Çıkan pencereden `sine8_LED.c` üzerine iki kez tıklayınız. `sine8_LED.c` kaynak dosyası, *Project View* penceresinde *Source* klasörünün içinde gözükmelidir.

5. `c6713dskinit.c` dosyasını projeye ekleyiniz. Bu dosya, `c:\CCStudio_v3.1\myprojects\Support` klasöründe saklıdır ve *comm_poll()* ve *output_left_sample()* fonksiyonlarını içeren düşük seviyeli işlemler için bazı fonksiyon tanımlarını içermektedir. *Project* → *Add Files to Project* seçiniz ve çıkan pencereden `*.c`, `*.ccc` uzantılı C kaynak dosyalarını aratınız. Çıkan pencereden `c6713dskinit.c` üzerine iki kez tıkladığınızda `c6713dskinit.c` kaynak dosyası, *Project View* penceresinde *Source* klasörünün içinde gözükmelidir.

6. `vectors_poll.asm` kaynak dosayasını projeye ekleyiniz. `vectors_poll.asm`, C6713 işlemcisi için kesme servis tablosunu içermektedir ve `c:\CCStudio_v3.1\myprojects\Support` klasöründe saklıdır. *Project* → *Add Files to Project* seçiniz ve çıkan pencereden `*.a*` uzantılı assembly kaynak dosyalarını aratınız. Çıkan pencereden `vectors_poll.asm` üzerine iki kez tıklayınız. `vectors_poll.asm` kaynak dosyası, *Project View* penceresinde *Source* klasörünün içinde gözükmelidir.

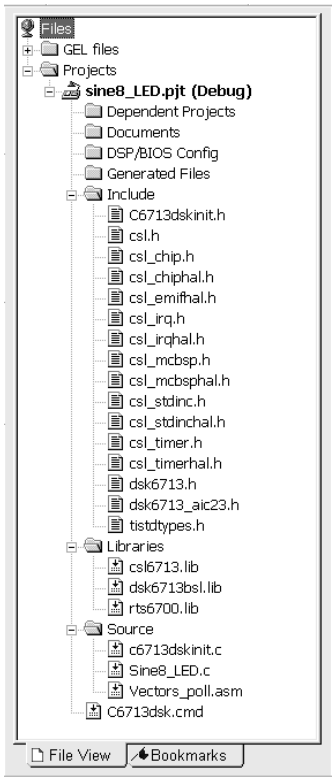
7. `rts6700.lib`, `dsk6713bsl.lib` ve `csl6713.lib` destek dosyalarını, *Project* →*Add Files to Project* işlemini üç kez tekrarlayarak ve çıkan pencereden `*.o*`, `*.l*` uzantılı nesne ve kütüphane kaynak dosyalarını aratarak projeye ekleyiniz. Dosyalar, sırasıyla `c:\CCStudio_v3.1\c6000\cgtools\lib`, `c:\CCStudio_v3.1\c6000\dsk6713\lib` ve `c:\CCStudio_v3.1\c6000\csl\lib` klasörlerinde mevcuttur. Bu dosyalar, (C67x mimarisi için) çalışma esnasındaki destek, (C6713 DSK için) kart desteği ve (C6713 işlemcisi için) işlemci desteği için kütüphane dosyalarıdır.

8. `c:\CCStudio_v3.1\myprojects\Support` klasöründe saklı `c6713dsk.cmd` bağlayıcı komut dosyasını projeye ekleyiniz. *Project* → *Add Files to Project* seçiniz ve çıkan pencereden `*.cmd`,`*.lcf` uzantılı dosyaları aratınız. Çıkan pencereden `c6713dsk.cmd` üzerine iki kez tıklayınız. `c6713dsk.cmd` kaynak dosyası, *Project View* penceresinde *Source* klasörünün içinde gözükmelidir.

9. Bu aşamada, *Project View* penceresinde hiçbir başlık dosyası gözükmemektedir. *Project* → *Scan All File Dependincies* seçilmesi bunu düzeltecektir. Bu işlemden sonra, *Project View* penceresinde `c6713dskinit.h`, `dsk6713.h` ve `dsk6713_aic23` başlık dosyalarını görebiliyor olmalısınız.

10. Adımları doğru yapmanız halinde, CCS'de *Project View* penceresi Şekil-3'deki gibi gözükmelidir. `dsk6713.gel` GEL dosyası, projeyi oluşturduğunuzda otomatik olarak eklenmektedir. GEL dosyası,CPU saatini 225 MHz'e ayarlamak için PLL'yi kullanan kart destek kütüphanesini çalıştırarak C6713 DSK'yı başlangıç konumuna getirir. *Project View* penceresinde listelenen herhangi bir dosyanın (kütüphane dosyaları hariç) isminin üzerine iki kez tıklanarak içeriğine bakılabilir (ve içeriği değiştirilebilir). Projeye başlık dosyası veya `include` komutu ile dosya eklememelisiniz. *Project* → *Scan All File Dependincies* seçtiğinizde dosyalar projeye otomatik olarak eklenir (projeyi derlediğinizde de dosyalar eklenir).

Project View penceresini inceleyerek proje (`.pjt`), bağlayıcı komut (`.cmd`), üç kütüphane (`.lib`), iki C kaynak (`.c`) ve assembly (`.asm`) dosyalarının projeye eklendiğinden emin olunuz.



Şekil 3: Onuncu adım sonunda eklenen dosyaları gösteren *Project View* penceresi.

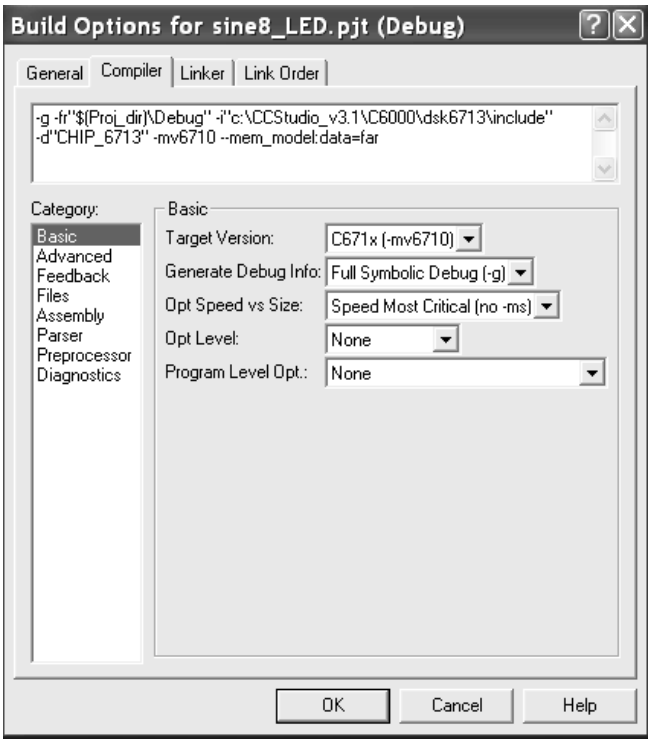
Proğram Oluşturma ve Üretme Seçenekleri
CCS'deki proğram oluşturma araçlarının (C derleyicisi, makina dili dönüştürücüsü ve bağlayıcı) herbirine ilişkin çeşitli seçenekler vardır. Çalıştırılabilir bir proğram oluşturma teşebbüsünden önce bu tercihler uygun olarak ayarlanmalıdır. Bu tercihler bir kez ayarlandıktan sonra proje dosyasında saklanacaktır.

Derleyici Tercihlerinin Ayarlanması
Project → *Build Options* seçiniz ve çıkan pencereden *Compiler* üzerine tıklayınız. Şekil 4,5 ve 6'da gösterildiği gibi aşağıdaki tercihleri ayarlayınız. *Basic* kategorisinde *Target Version* kısmını *C671x (-mv6710)*'ye ayarlayınız. *Advanced* kategorisinde *Memory Models* kısmını *Far (-mem-model:data=far)*'a ayarlayınız. *Preprocessor* kategorisinde *Pre-Define Symbol* kısmını *CHIP-6713*'e ve *Include Search Path (-i)* kısmını `c:\CCStudio_v3.1\c6000\dsk6713\include` klasörüne ayarlayınız. Ayarlardan sonra *OK* üzerine tıklayınız.

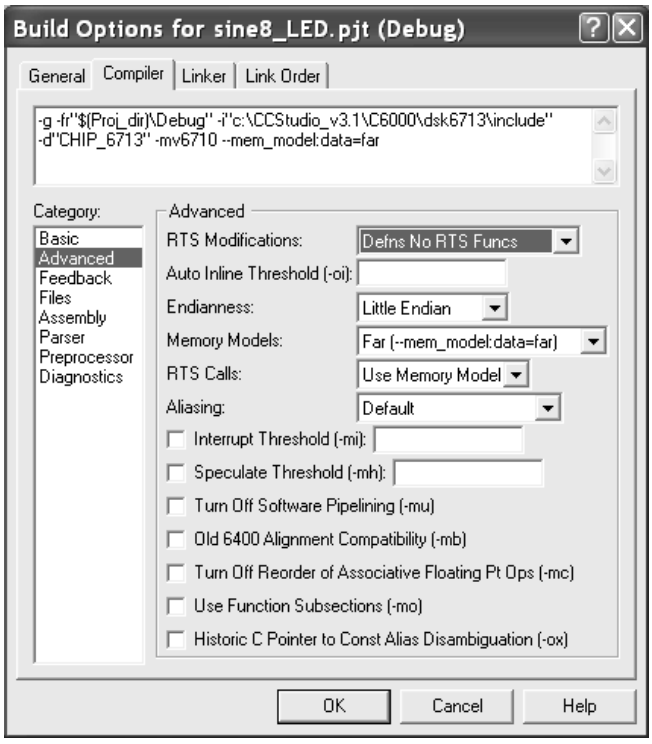
Bağlayıcı Tercihlerinin Ayarlanması
Project→ *Build Options* seçildiğinde çıkan *Build Options* penceresinde Şekil 7'de gösterildiği gibi *Linker* üzerine tıklayınız. *Output Filename* kısmı proje dosyasının ismine dayalı olarak `.\Debug\sine8_LED.out` değerine ayarlanmalıdır ve *Autoinit Model* kısmında *Run-Time Autoinitialization* yazılmalıdır. Ayrıca, aşağıdaki ayarları yapınız. *Library Search Path* kısmını `c:\CCStudio_v3.1\c6000\dsk6713\lib` klasörüne ayarlayınız ve *Include Libraries* kısmında *rts6700.lib, dsk6713bsl.lib, csl6713.lib* yazınız (tümü *Basic* kategorisinde). Map dosyası proğram kontrolünde (fonksiyonların hafızadaki konumları vb) faydalı bilgi sağlayabilir. Değişkenlere çalışma esnasında ilk değer atamak için -c tercihi kullanılır ve -o tercihi çalıştırılabilir çıkış dosyasını (`sine8-LED.out`) isimlendirmek içindir. Ayarlardan sonra *OK* üzerine tıklayınız.

Çalıştırılabilir Proğramın Oluşturulması, İşlemciye Yüklenmesi ve Çalıştırılması
Yukarıda anlatılan işlemlerden sonra `sine8_LED` projesi oluşturulabilir ve `sine8_LED.out` çalıştırılabilir dosyası DSK'ya yüklenebilir ve çalıştırılabilir.

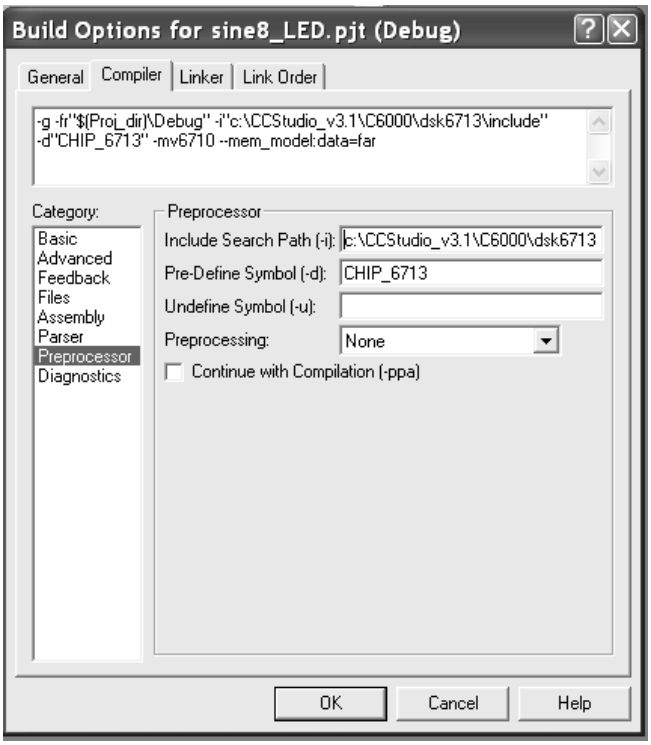
1. Projeyi `sine8_LED` olarak oluşturunuz. *Project* → *Rebuild All* seçiniz (veya aşağıya doğru üç ok sembollü bütona tıklayınız). Bu işlem sonucunda, `c16x` kullanılarak tüm C dosyaları derlenir ve makine diline dönüştürülür; `asm6x` kullanılarak `vectors_poll.asm` assembly dosyaysı makina di-



Şekil 4: CCS derleme tercihleri: Temel (basic) derleyici ayarları.



Şekil 5: CCS derleme tercihleri: İleri (advanced) derleyici ayarları.



Şekil 6: CCS derleme tercihleri: Önişlemci (preprocessor) derleyici ayarları.

line dönüştürülür. Sonuçlanan nesne dosyaları, `lnk6x` kullanılarak kütüphane dosyalarına bağlanır. Bu işlemler sonucunda C6713 işlemcisine yüklenebilen ve çalıştırılabilen `sine8_LED.out` dosyası oluşturulur. Derleme, makine diline dönüştürme ve bağlama komutlarının Build tercihiyle yapıldığına dikkat ediniz. Derleyici seçenekleriyle birlikte derlenen ve makine diline dönüştürülen dosyalarını gösteren bir `cc_build_Debug.log` kayıt dosyası oluşturulur. Kayıt dosyası, kullanılan destek fonksiyonlarını da listeler. Proje oluşturma işlemi, (dosya bağımlılıklarının taratılmasının unutulması durumunda) bağımlı tüm dosyaların dahil edilmesine neden olur. CCS penceresinin sol alt köşesinde "Build Complete, 0 Errors, 0 Warnings, 0 Remarks" ile son bulan bir takım uyarılar göreceksiniz. Stack boyutu hakkında bir uyarı gözükebilir. Bu uyarı ihmal edilebilir veya *Linker Build Options*'ın *Advanced* kategorisindeki *Warn About Output Sections* tercihi seçilmeyerek uyarının gözükmesi engellenebilir. Uyarının gözükmesini engellemenin diğer bir yolu, *Linker Build Options*'ın *Advanced* kategorisindeki *Stack Size* tercihini uygun bir değere (örneğin `0x1000`) ayarlamaktır.

Debug → *Connect* seçerek DSK'ya bağlanın ve CCS penceresinin sol alt köşesindeki sembolün DSK'ya bağlı olduğunuzu belirttiğinden emin olun.

2. `sine8_LED.out` çalıştırılabilir proğramını işlemciye yüklemek için *File* → *Load Program* seçiniz. Proğram, `c:\CCStudio_v3.1\myprojects\sine8_LED\Debug` klasöründe saklıdır. *Debug* → *Run* seçiniz. DSK üzerindeki LINE OUT ve HEADPHONE çıkışlarında 1 kHz frekanslı sinüzoidal bir çıkış dalgaşeklinin olduğunu teyit etmek için 0 nolu DIP anahtarın yönü aşağı doğruyken LINE OUT çıkışına bağlı bir osiloskop ve HEADPHONE çıkışına bağlı bir ses kablosu kullanın.

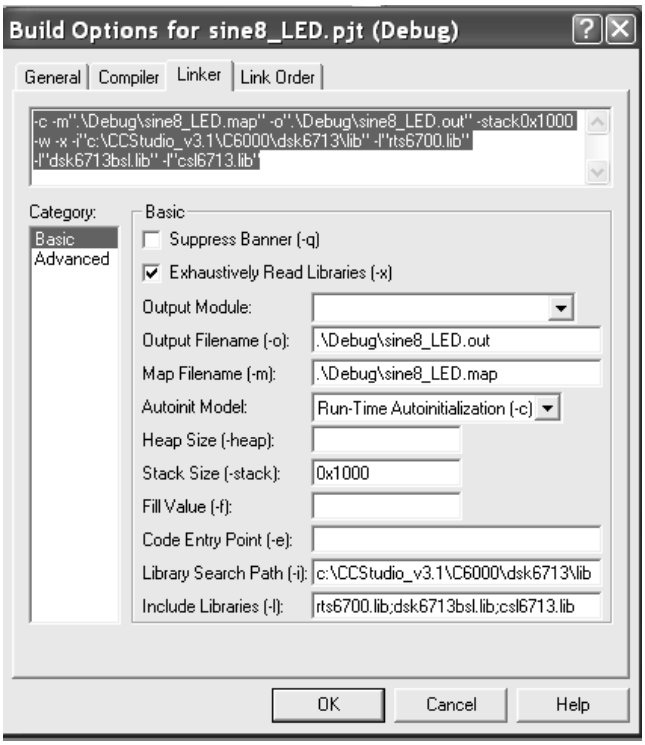
CCS Kullanarak Kaynak Dosyalarını Düzenlenme

Kaynak dosyalarını düzenlemeyi öğrenmek amacıyla aşağıdaki işlemleri gerçekleştiriniz.

1. *Debug* → *Halt* seçerek proğramın çalışmasını durdurun.
2. *Project View* penceresinde `sine8_LED.c` üzerine iki kez tıklayınız. Bu işlem sonucunda CCS içinde kaynak dosyasının gösterildiği yeni bir pencere açılacaktır.
3. Proğram içinde

   ```
   short gain = 10;
   ```

   satırındaki noktalı virgülü silin.



Şekil 7: CCS derleme tercihleri: Temel (basic) bağlayıcı ayarları.

4. Kısmi derleme yapmak için *Debug* → *Build* seçiniz veya aşağı doğru iki oklu bütonun üzerine tıklayın. Sadece `sine8_LED.c` dosyası değiştiğinden kısmi derleme yapmak mantıklıdır. Rebuild tercihinin (aşağıya doğru üç oklu sekme) kullanılması durumunda, önceden derlenmiş ve makine diline dönüştürülmüş dosyalar gereksiz bir şekilde yeniden işlenirler.

5. CCS'nin solt alt köşesindeki *Build* penceresinde aşağıda verilen kırmızı renkte iki hata gözükmelidir:

   "Sine8-LED.c", Line 11: error: expected a ";"
   "Sine8-LED.c", Line 23: error: identifier "sine-table" is undefined

   Bu hataları daha iyi görmek için *Build* penceresinde yukarıya doğru gitmek zorunda olabilirsiniz. Birinci hata satırının üzerine iki kez tıklayınız. Bu, körsırı proğramda hata olan satıra getirir. Uygun düzeltmeyi yaptıktan sonra projeyi yeniden derleyin, işlemciye yükleyin, çalıştırın ve önceden elde ettiğiniz sonuçları elde edin.

Watch (Gözlem) Penceresinin İzlenmesi

İşemcinin çalıştığından (ve 0 nolu DIP anahtarın yönünün aşağıya doğru olduğundan) emin olun. CCS'nin sol alt köşesinde "RUNNING" mesajı gözükmelidir. *Watch* penceresi, bir parametrenin değerini değiştirmeye veya bir değişkeni izlemeye imkan verir.

1. *View* → *Quick Watch* seçiniz. *gain* yazınız ve daha sonra *Add to Watch* üzerine tıklayınız. Proğramda 10 yapılan kazanç değeri Watch penceresinde gözükmelidir.

2. *Watch* penceresinde *gain* değerini 10'dan 30'a değiştirin. Üretilen sinüzoidal işaretin genliğinin arttığını doğrulayınız (işlemci çalışmaktadır ve 0 nolu DIP anahtarın yönü aşaığıdır). Sinüs dalgasının genliği tepeden-tepeye 0.9V değerinden 2.5V değerine yükselmelidir.

GEL Fonksiyonu Kullanarak Kazancı Kontrol Etme

Genel Genişletme Dili (GEL) C'ye benzer bir dildir. İşlemci çalışırken bir değişkenin değerini değiştirmenize imkan verir.

1. *File* → *Load GEL* seçiniz ve `sine8_LED` klasöründeki `gain.gel` dosyasını yükleyiniz. *Project View* penceresinde `gain.gel` dosya ismi üzerine iki kez tıklayarak dosyanın CCS içinde bir pencerede açılmasını sağlayınız. GEL dosyası, Şekil 8'de verilmiştir. GEL fonksiyonunun formatı

```
/*gain.gel GEL slider to vary amplitude of sinewave*/
/*generated by program sine8_LED.c*/

menuitem "Sine Gain"

slider Gain(0,30,4,1,gain_parameter) /*incr by 4, up to 30*/
{
        gain = gain_parameter;       /*vary gain of sine*/
}
```

Şekil 8: `gain.gel` GEL dosyası.

```
slider param_definition(minVal,maxVal,icrement,pageIncrement,paramName)
{
  ifadeler
}
```

şeklindedir. `param_definition` kayma çubuğunu tanımlar ve kayma penceresinin ismi olarak gözükür; `minVal` kayma çubuğu en alt seviyedeyken `paramName` GEL değişkenine atanan değerdir, `maxVal` kayma çubuğu en alt seviyedeyken `paramName` GEL değişkenine atanan değerdir, `increment` GEL değişkenine yukarı ve aşağı oklar kullanılarak yapılan değişimin miktarını ve `pageincrement` GEL değişkenine kayma penceresine tıklanarak yapılan değişimin miktarını belirtir. `gain.gel` durumunda

```
gain = gain_parameter
```

ifadesi `gain_parameter` GEL değişkeninin değerini `sine8_LED.c` proğramındaki `gain` değişkenine atar.

```
menuitem "Sine Gain"
```

satırı, `gain.gel` yüklendiğinde CCS'deki *GEL* menüsünde bir tercih olarak gözükecek metni ayarlar.

2. *GEL* → *Sine Gain* → *Gain* seçiniz. Bu işlem sonucunda, Şekil-9'da gösterilen kayma penceresi gözükecek ve kazanç en küçük değer 0'a ayarlanacaktır.

3. Yukarı ok tuşuna üç kez tıklayarak kazanç değerini 0'dan 12'ye yükseltiniz. Üretilen sinüs dalgasının tepeden-tepeye (t-t) değerinin yaklaşık 1.05V olduğunu doğrulayınız. Kayma çubuğunu her seferinde 4 birim arttırmaya devam etmek için yukarı ok tuşuna tıklayınız. `gain` değeri 30 olduğunda sinüs dalgasının t-t genliği 2.5V civarında olmalıdır. Kayma penceresinde, okun mevcut konumunun altında (üstünde) bir yerde tıklamanız durumunda değişkenin değeri 1 azalacaktır (artacaktır). Kayma çubuğu kullanılarak gain değişkeninin değerinde yapılan değişmeler *Watch* penceresine yansıyacaktır.

Üretilen Sinüsün Frekansını Değiştirme

`sine8_LED.c` proğramıyla üretilen sinüzoidal işaretin frekansını değiştirmenin birkaç yolu vardır.

1. AIC'nin örnekleme frekansını, `sine8_LED.c` proğramındaki `"Uint fs=DSK6713-AIC23-FREQ-8KHZ;"` satırını `"Uint fs=DSK6713-AIC23-FREQ-16KHZ;"` şeklinde yeniden düzenleyerek 8 kHz'den 16 kHz'e yükseltiniz. Projeyi kısmi derleyin, işlemciye yükleyin, çalıştırılabilir yeni dosyayı çalıştırın ve üretilen sinüsün frekansının 2 kHz olduğunu doğrulayın. AIC ADC/DAC'nin desteklediği örnekleme frekansları 8,16,24,32,44.1,48 ve 96 kHz'dir.

2. Okuma tablosundaki örnek sayısını dört yapın. `sine8_LED.c` proğramında

```
define LOOPLENGTH 8
short sine_table[LOOPLENGTH]={0,707,1000,707,0,-707,0,-1000,-707}
```

ile verilen satırları

Şekil 9: `sine8_LED` proğramındaki kazanç değerini değiştirmek için kullanılan GEL çubuğu.

```
define LOOPLENGTH 4
short sine_table[LOOPLENGTH]={0,1000,0,-1000}
```

şeklinde değiştiriniz. Üretilen sinüsün frekansının 2 Khz olduğunu doğrulayınız. 0 nolu DIP anahtarın yönü aşağı doğru değilse sinüs işaretinin üretilmeyeceğini hatırlayınız. *DSK6713-DIP-get(), DSK6713-LED-on()* ve *DSK6713-LED-off()* fonksiyonlarına geçirilen parametrenin değerini değiştirerek bir sinüsün üretilip üretimeyeceğini belirlemek amacıyla farklı bir DIP anahtar kullanılabilir. Uygun değerler 0,1,2 ve 3'dür.

Kazancı ve frekansı değiştirmek amacıyla iki çubuk kullanılabilir. C proğramı içinde `loopindex` değişkeninin değerine uygulanan artış miktarı değiştirilerek (örneğin okuma tablosundaki iki örnekten biri atlanarak) farklı frekanslı bir işaret üretilebilir. Bir proje üretildikten sonra CCS'yi kapattığınızda projeye yapılan tüm değişiklikler saklanabilir. Daha sonra, projeye bıraktığınız yerden devam edebilirsiniz. Örneğin, CCS'yi başlattıktan sonra `sine8_LED.pjt` gibi var olan bir projeyi açmak için *Project* → *Open* seçiniz.

Örnek 2: Sinüsün Üretilmesi ve CCS'yle Çizilmesi (sine8_ buf)

Bu örnekte, önceden hesaplanmış ve saklanmış sekiz örnek değer kullanarak sinüzoidal bir analog çıkış işareti üretilmesinin nasıl yapılacağı gösterilecektir. Ancak, çalışması kesme kullanımına dayalı olduğundan `sine8_LED` projesinden bütünüyle farklıdır. Ayrıca, `BUFFERLENGTH` adet en son çıkış örneğini saklamak için bir kayıt dosyası kullanmaktadır. Örnek, verileri hem zaman hem de frekans uzaylarında çizmede CCS'nin yeteneklerini göstermek amacıyla verilmiştir.

Çalıştırılabilir `sine8_buf.out` dosyası oluşturmak için gerekli tüm dosyalar `sine8_buf` klasöründedir. `sine8_buf.c` proğram dosyası Şekil-10'de verilmiştir. `sine8_buf.pjt` proje dosyası mevcut olduğundan yeni bir proje dosyası oluşturmaya ve dosyalar eklemeye veya derleyici ve bağlayıcı seçeneklerini değiştirmeye gerek yoktur. `sine8_buf.c` proğramını derlemek, işlemciye yüklemek ve çalıştırmak için aşağıdaki adımları gerçekleştiriniz:

1. CCS'de açık tüm projeleri kapatınız.

2. *Project* → *Open* seçerek ve `sine8_buf` klasöründeki `sine8_buf.pjt` dosyasının üzerine iki kez tıklayarak textttsine8_ buf.pjt projesini açınız. Bu proğram, yoklama yerine kesmeyle belirlenen giriş/çıkş kullandığından `vectors_poll.asm` yerine `vectors_intr.asm` kullanılır. `vectors_intr.asm` dosyasında belirtilen kesme servis tablosu (IST) `c_int11()` kesme servis işlemini (ISR), her örnekleme anında DSK üzerindeki AIC23 tarafından gönderilen INT11 donanım kesmesiyle ilişkilendirir.

`main()` fonksiyonunun içinde `comm_poll()` yerine `comm_intr()` fonksiyonu kullanılmıştır. Bu fonksiyon, `c6713dskinit.c` dosyasında tanımlıdır ve ikinci laboratuarda detaylı açıklanacaktır. `comm_intr()` fonksiyonu temel olarak, AIC23 örnekleme frekansı `fs` değişkeninin değerine göre ayarlanacak ve her örnekleme anında AIC23 işlemciye kesme gönderecek şekilde DSK donanımını başlangıç değerine getirir. `main()` fonksiyonundaki `while(1)` ifadesi, işlemcinin kesme beklediği sonsuz bir çevrim oluşturur. Kesme oluştuğunda proğram kontrolü, `sine_table` dizisinden yeni bir örnek okuyan ve örneği hem `out_buffer`'a hem de `output_left_sample()` fonksiyonu kullanarak DAC'ye yazan kesme servis proğramı (ISR) `c_int11()`'ye geçer. Kesmeler detaylı olarak üçüncü laboratuarda tartışılacaktır.

```
//sine8_buf.c sine generation with output stored in buffer
#include "DSK6713_AIC23.h"                  //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;          //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input
#define LOOPLENGTH 8
#define BUFFERLENGTH 256
int loopindex = 0;    //table index
int bufindex = 0;                            //buffer index
short sine_table[LOOPLENGTH]={0,707,1000,707,0,-707,-1000,-707};
int out_buffer[BUFFERLENGTH];                //output buffer
short gain = 10;
interrupt void c_int11()                     //interrupt service routine
{
  short out_sample;
  out_sample = sine_table[loopindex++]*gain;
  output_left_sample(out_sample);        //output sample value
  out_buffer[bufindex++] = out_sample;     //store in buffer
  if (loopindex >= LOOPLENGTH) loopindex = 0; //check for end of table
  if (bufindex >= BUFFERLENGTH) bufindex = 0; //check for end of buffer
  return;                //return from interrupt
}
void main()
{
  comm_intr(); //initialise DSK
  while(1);     //infinite loop
}
```

Şekil 10: `sine8_buf.c` prooğramı.

Bu projeyi `sine_buf` olarak derleyin. Çalıştırılabilir `sine8_buf.out` dosyasını işlemciye yükleyerek çalıştırın ve birinci örnekte olduğu gibi LINE OUT ve HEADPHONE çıkışlarında 1 kHz sinüzoidal bir işaret olduğunu doğrulayın.

CCS'de Grafik Çizimi

En son `BUFFERLENGTH` sayıda çıkış örneğini saklamak için `out_buffer` dizisi kullanılır. Proğramın çalışması durdurulduktan sonra, `out_buffer`'da saklı veri CCS'de grafik olarak gösterilebilir.

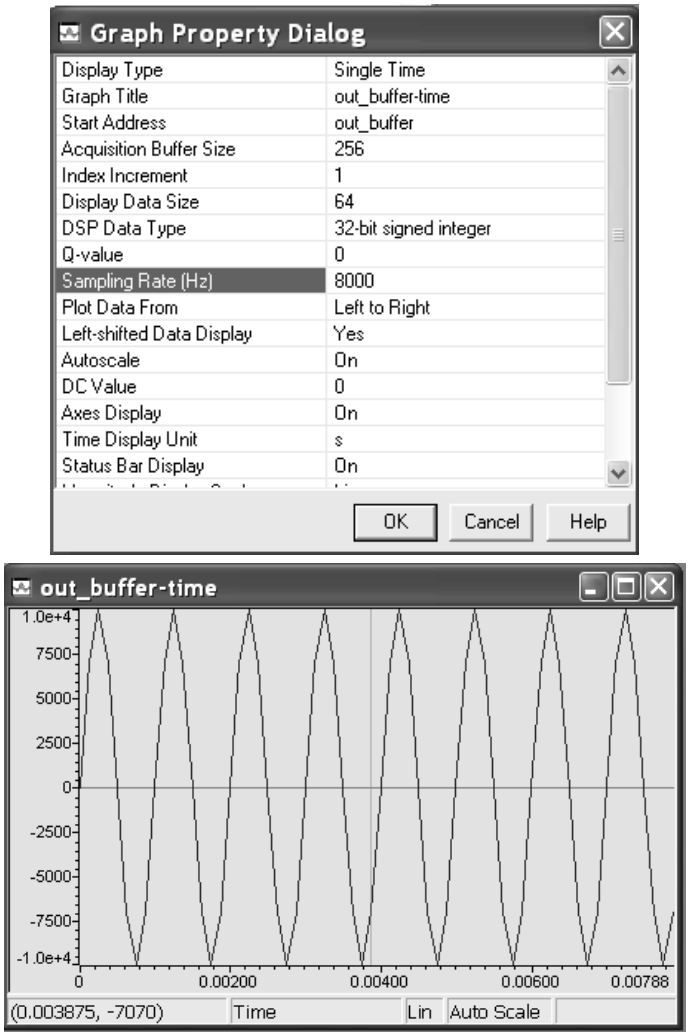
1. *View* → *Graph* → *Time/Frequency* seçiniz ve *Graph Property Dialog* özelliklerini Şekil 11(a)'da gösterildiği gibi ayarlayınız. Şekil 11(b)'de görülen eğriyi elde etmelisiniz.

2. Şekil 12(b)'de gösterilen `out_buffer` içeriğinin frekans uzayı gösterilimine karşılık gelen *Graph Property Dialog* penceresi Şekil 12(a)'da gösterilmiştir. 1 kHz'deki impuls, `sine8_buf.c` proğramıyla üretilen sinüzoidal işaretin frekansını belirtmektedir.

Hafızadaki Verinin Gösterilmesi ve Dosyaya Kaydedilmesi

`out_buffer` içeriğini görmek için *View* → *Memory* seçiniz. Şekil 13(a)'da gösterildiği gibi *Address* alanına `out_buffer` yazınız ve *Format* alanında *32-bit Signed Integer* tercihini yazınız. OK tuşuna tıkladıktan sonra sonuçlanan hafıza penceresi Şekil 13(b)'de verilmiştir.

`out_buffer` içeriğini bir dosyaya kaydedmek için *File* → *Data* → *Save* seçiniz. Dosyayı, `sine8_buf` klasöründe veri türünü *Integer* seçerek `sine8_buf.dat` olarak saklayınız. *Storing Memory Into File* penceresinde *Address* kısmına `out_buffer` yazınız ve *Length* alanına 256 giriniz. Sonuçlanan dosya bir metin dosyasıdır ve diğer proğramlar (örneğin MATLAB) kullanılarak çizdirilebilir. `sine_table` dizisinde saklanan ve `output_left_sample` fonksiyonuna gönderilen değerler 16-bit işaretli tamsayılar olmasına rağmen, `sine8_buf.c` proğramında `out_buffer` dizisinin 32-bit işaretli tamsayı olarak temsil edilmesinin nedeni CCS'de *Save* → *Data* seçildiğinde 16-bit işaretli tamsayı tercihinin olmamasıdır.



Şekil 11: `out_buffer`'da saklı veri için (üst) Graph Property pencresi, (alt) zaman grafiği.

Örnek 3: İki Dizinin İç Çarpımı (dotpt)

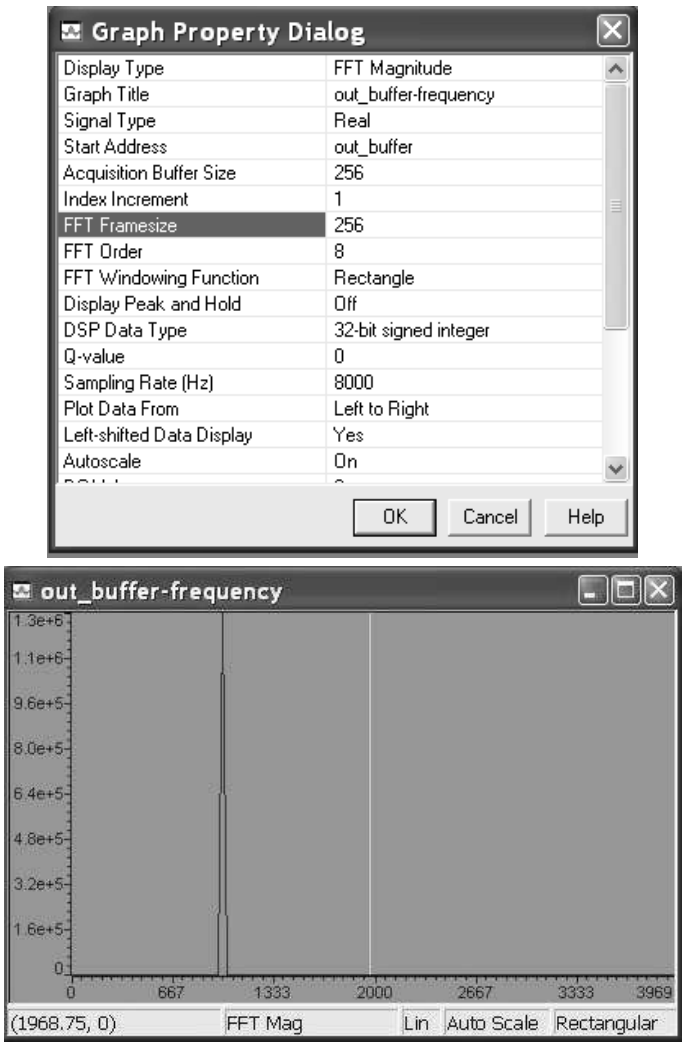
Bu örnekte, CCS'de durma noktalarının oluşturulması ve proğramda adım adım ilerleme anlatılacaktır. Ayrıca, bir proğramın belirli bir kısmını çalıştırmak için geçen süreyi tahmin etmek amacıyla CCS'nin *Profile Clock* özelliği tanıtılacaktır.

Çarpma/toplama, sayısal işaret işlemede çok önemli bir işlemdir. Çarpma/toplama, sayısal filtreleme, korelasyon ve hızlı Fourier dönüşümü algoritmlarında kullanılan temel bir işlemdir. Çarpma işlemi sık yapıldığından ve çoğu işaret işleme algoritmları için önemli olduğundan bir komut süresinde yapılması önemlidir. C6713 işlemcisi bir komut süresinde iki çarpma/toplama gerçekleştirebilir.

Şekil 14'de verilen `dotp4.c` C kaynak dosyası, tamsayılardan oluşan iki dizinin iç çarpımını hesaplamaktadır. 1,2,3 ve 4 değerleri kullanılarak ilk diziye; 0,2 ve 4 değerleri kullanılarak da ikinci diziye başlangıç değeri veilir. İki dizinin iç çarpımı $(1 \times 0) + (2 \times 2) + (3 \times 4) + (4 \times 6) = 40$'dır. Uzunluğu dörtten fazla dizilerinn iç çarpımını hesaplamak için proğramda gerekli değişiklikler kolaylıkla yapılabilir. Bu örnekte gerçek-zaman giriş veya çıkış kullanılmadığından `c6713dskinit.c` ve `vectors_intr.asm` gerçek-zaman destek dosyasları gerekli değildir.

Aşağıda listelenen dosyaların projeye dahil edildiğinden emin olduktan sonra bu projeyi `dotp4` olarak derleyiniz.

1. `dotp4.c`: C kaynak dosyası
2. `c6713dsk.cmd`: genel bağlayıcı komut dosyası
3. `rts6700.lib`: kütüphane dosyası



Şekil 12: `out_buffer`'da saklı veri için (üst) Graph Property pencresi, (alt) frekans grafiği.

Bu işlemlerden sonra *Project view* penceresi Şekil 15'deki gibi olmalıdır.

Bir Değişkenin Gözlemlenmesi

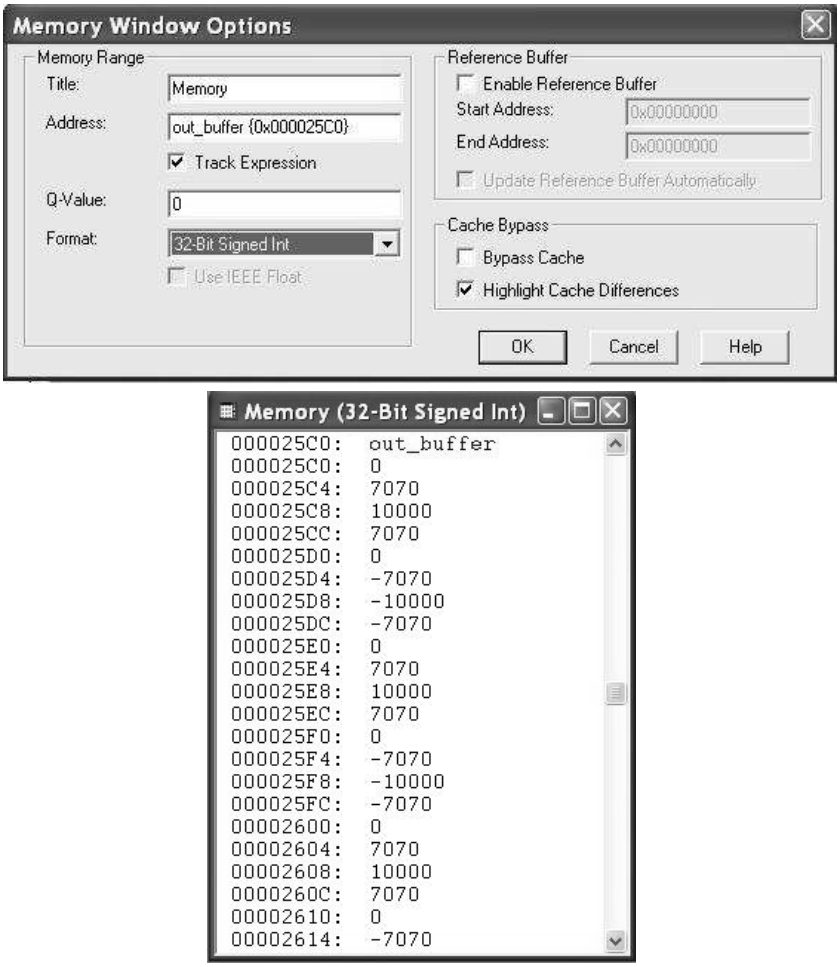
1. *Project* → *Build Options* seçiniz ve *Basic Compiler* ayarlarının Şekil 16'daki gibi olduğundan emin olunuz. Bu örnekte, optimizasyonun seçili olmaması (*Opt Level None*) önemlidir.

2. Aşağı doğru üç oklu bütona tıklayarak (veya menüden *Project* → *Build*) seçerek projeyi derleyiniz ve çalıştırılabilir `dotp4.out` dosyasını işlemciye yükleyiniz.

3. *View* → *Quick Watch* seçiniz. Çıkan pencerede, `sum` değişkenini gözlemlemek için *sum* yazınız ve *Add to Watch* üzerine tıklayınız. *Watch* penceresinde "identifier not found:sum" uyarısı gözükmelidir. `sum` değişkeni `dotp()` fonksiyonunda tanımlanmıştır ve bu fonksiyon çağrılmadığı sürece mevcut değildir.

4. `dotp4.c` kaynak dosyasında, `sum += a[i] * b[i];` yazan satırın üzerine tıklayarak ve *Toogle Breakpoint* sekmesine tıklayarak bir durma noktası oluşturun. Proğram satırının solunda kırmızı bir nokta gözükmelidir.

5. *Debug* → *Run* seçiniz. Proğram, durma noktasının oluşturuluduğu satıra kadar çalışacaktır. Durma noktasının oluşturuluduğu satırın solunda sarı bir ok gözükecektir. Bu aşamada, *Watch* penceresinde `sum` değişkeni için 0 değeri gözükmelidir. `sum`, sadece `dotp()` fonksiyonu için tanımlı bir değişkendir. `dotp()` fonksiyonu çalışmakta olduğundan değişken mevcutur ve değeri gösterilebilir.



Şekil 13: `out_buffer`'da saklı veri için (üst) Memory pencresi ayarları, (alt) veriler.

6. *Debug* → *Step Into* seçerek veya F11 fonksiyon tuşuna basarak proğramı kaldığı yerden çalıştırmaya devam edin. Her döngü sonunda ve *Watch* penceresinde `sum` değişkeninin 0, 4, 16 ve 40 şeklinde değiştiğinini gözlemleyiniz (Şekil 17'ye bakınız).

7. `sum` değişkeninin değeri 40'a ulaştıktan sonra, proğramın çalışmasını durdurmak için *Debug* → *Run* seçiniz `dotp()` fonksiyonu ile oluşturulan değerin *Stdout* penceresinde `result = 40 (decimal)` şeklinde gözüktüğünü doğrulayınız. Bu aşamada, `dotp()` fonksiyonunun çalışmasının bittiğini `sum` yerel değişkeninin mevcut olmadığını belirten "identifier not found:sum" uyarısı *Watch* penceresinde tekrar gözükmelidir.

`printf()` fonksiyonu, hataların tespiti ve düzeltilmesi amacıyla faydalıdır ancak 6000 komut süresinden fazla süre aldığından gerçek-zaman proğramlarda kullanılmasından kaçınılmalıdır.

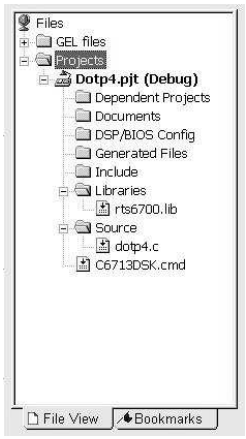
Animasyon

1. Çalıştırılabilir `dotp4.out` dosyasını yüklemek için *File* → *Reload Proğram* seçiniz. Çalıştırılabilir dosya yüklendikten sonra proğram sayıcısı `c_int00` adresine ayarlanır. *Disaasembly* penceresine bakılarak bu doğrulanabilir.

2. Önceden oluşturulan durma noktası proğramın iligi satırında duruyor olmalıdır. *Debug* → *Animate* seçiniz ve `sum` değişkeninin değerinin *Watch* penceresinde değiştiğini gözlemleyiniz. Animasyonun hızı *Option* → *Customize* → *Animate Speed* seçilerek ayarlanabilir.

```
//dotp4.c dot product of two vectors
int dotp(short *a, short *b, int ncount); //function prototype
#include <stdio.h>              //for printf
#define count 4                 //# of data in each array
short x[count] = {1,2,3,4};     //declaration of 1st array
short y[count] = {0,2,4,6};     //declaration of 2nd array
main()
{
  int result = 0;     //result sum of products

  result = dotp(x, y, count); //call dotp function
  printf("result = %d (decimal) \n", result); //print result
}
int dotp(short *a, short *b, int ncount) //dot product function
{
  int i;
  int sum = 0;
  for (i = 0; i < ncount; i++)
     sum += a[i] * b[i];        //sum of products
  return(sum);                  //return sum as result
}
```

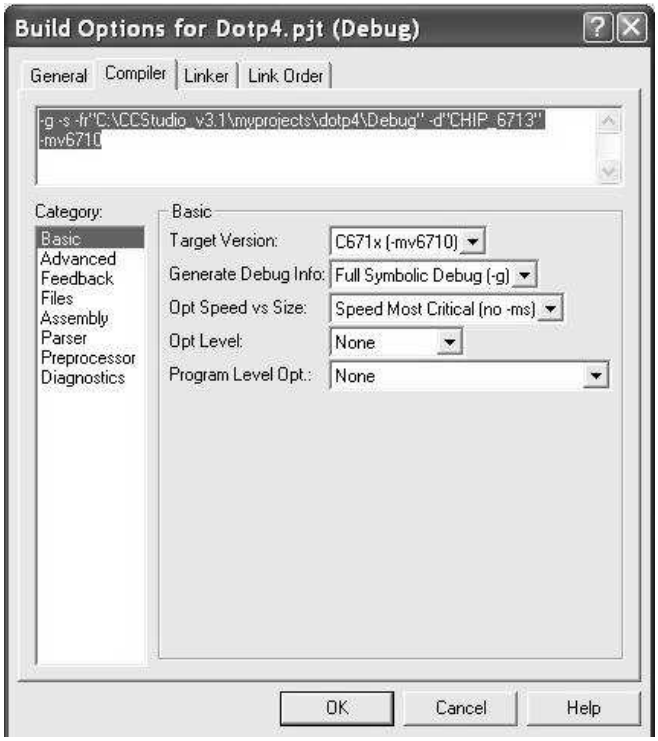
Şekil 14: `dotp4.c` proğramı.



Şekil 15: `dotp4` projesi için *Project View* penceresi.

Profile Clock'la dotp() Fonksiyonunun Çalışma Süresinin Tahmini
`dotp()` fonksiyonunun çalışması için geçen süre CCS'deki *Profile Clock* seçeneği kullanılarak tahmin edilebilir.

1. `dotp4.pjt` projesini açınız.
2. *Project* → *Build Options* seçiniz. *Basic* kategorisinde *Compiler* sekmesinde *Opt level* kısmını *none* değerine ayarlayınız.
3. *Project* → *Build* seçiniz ve daha sonra `dotp4.out` dosyasını oluşturmak ve yüklemek için *File* → *Load Program* seçiniz.



Şekil 16: `dotp4` projesi için derleme tercihleri.

4. `dotp4.c` kaynak dosyasını açınız ve tüm durma noktalarını kaldırınız.

   ```
   result = dotp(x,y,count);
   ```

   ve

   ```
   printf("result= d (decimal) /n", result);
   ```

   satırlarında durma noktaları oluşturunuz.

5. *Profile* → *Clock* → *Enable* seçiniz.

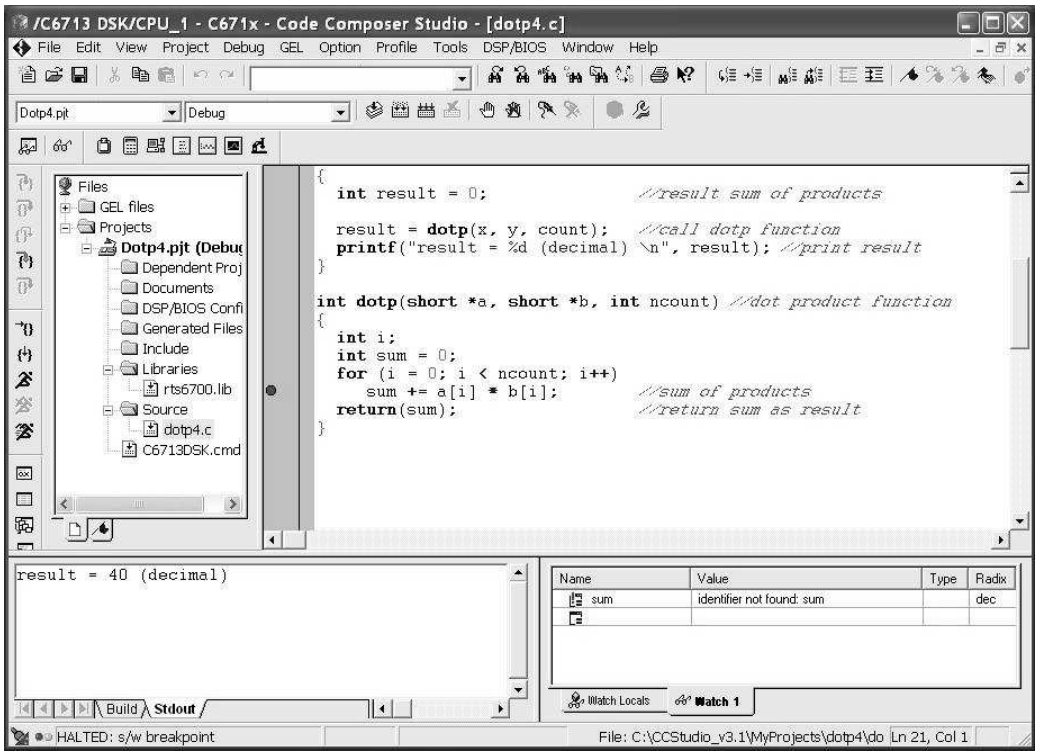
6. *Profile* → *Clock View* seçiniz. CCS penceresinin sağ alt köşesinde *Profile Clock* aracının saydığı işlemci komut çevrim sayısı ve küçük bir resim gözükmelidir.

7. Proğramı çalıştırdığınızda illk durma noktasında durmalıdır.

8. CCS penceresinin sağ alt köşesinde küçük resme iki kez tıklayarak *Profile Clock* aracını başlangıç konumuna getiriniz.

9. Proğramı çalıştırdığınızda ikinci durma noktasında durmalıdır.

İki durma noktası arasında, yani `dotp()` fonksiyonunun çalışması esnasında *Profile Clock* tarafından sayılan komut çevrim sayısı küçük resmin yanında gözükmelidir. 225 MHz bir işlemcide bir komut çevrimi 4.44 nano saniyedir. Derleyici optimizasyon seviyesini *Function (-o2)*'ye ayarlayarak deneyi tekrarlayınız ve `dotp()` fonksiyonunun kullandığı komut çevrim sayısında yaklaşık 2 oranında bir azalma görmelisiniz. Durma noktaları ve *Profile Clock* kullanmak bir proğramın kısımlarının çalışma süresini belirtir, ancak derleyici optimazyonunun yüksek seviyelerinde (örneğin *File (-o3)* herzaman çalışmaz. Proğram çalışmasının daha detaylı izlenmesi bir simülatör kullanılarak yapılabilir.



Şekil 17: `dotp4` proğramı için çeşitli pencereler.

4 Alıştırmalar

1. 3 kHz frekanslı bir sinüs dalgası üretmek için `sine8_buf.c` dosyasında gerekli değişiklikleri yapınız. Sonuçlarınızı, DSK üzerindeki LINE OUT çıkışına bağlı bir osiloskop kullanarak ve en son 32 çıkış örneğini zaman ve frekans uzaylarında çizmek amacıyla CCS kullanarak doğrulayınız.

2. 3 nolu DIP anahtarın konumu aşağıya doğruyken 3 nolu LED yanacak ve 5 saniyeliğine 500 Hz frekanslı bir kosinüs üretilecek şekilde yoklamaya dayalı bir proğram yazınız.

3. DSK üzerinde MIC IN girişi kullanılarak AIC23 ADC/DAC'den 16 kHz örnekleme frekansında okunan en son 128 örneği saklayan bir diziyi üreten kesme tabanlı bir proğram yazınız. Proğramın çalışmasını durdurunuz ve dizinin içeriğini CCS kullanarak çiziniz.

4. Giriş örneklerini, ADC'nin sol kanalından 16 kHz örnekleme frekansında `input_left_sample()` fonksiyonunu kullanarak okuduktan sonra DAC'nin sağ kanalına `input_right_sample()` fonksiyonunu kullanarak yazan bir proğram yazınız. Bir işaret üreticisi ve osiloskop kullanarak LINE IN girişinin sol kanalının LINE OUT çıkışının sağ kanalına bağlandığını teyit ediniz. Çıkış işaretinin genliği ciddi derecede azalıncaya kadar giriş işaretinin frekansını yavaşça arttırınız. Bu frekans, DSP sisteminin bandgenişliğine karşılık gelmektedir (ikinci laboratuarda daha detaylı tartışma yapılacaktır.)

EEM 437-SAYISAL İŞARET İŞLEME LABORATUARI
LAB 2: TMS320C6713 DSK'DA VERİ GİRİŞ VE ÇIKIŞI

1 Giriş

Ses frekansı işaretleri işlemeye uygun bir temel DSP sistemi Şekil 1'de gösterildiği gibi bir sayısal işaret işlemci ve analog arayüzlerden oluşur. C6713 DSK, TMS320C6713 (C6713) kayan-nokta işlemcisini ve TLV320AIC23 (AIC) analog-sayısal/sayısal-analog (AD/DA) dönüştürücüsü kullanarak böyle bir sistem sunmaktadır. AD/DA dönüştürücü terimi, analog dalgaşekillerinin sayısal işaretler olarak kodlanması ve sayısal işaretlerin analog dalgaşekilleri olarak çözülmesi anlamına gelmektedir.

Diğer bir seçenek olarak, DSK üstünde 80-bacaklı J3 harici çevrebirimi arayüzüne bağlanan giriş/çıkış (I/O) kartları analog giriş ve çıkış için kullanılabilir. Ancak, bu laboratuardaki proğramlama deneyleri DSK üzerindeki AIC23 AD/DA dönüştürücüsünü kullanmaktadır.

Örnekleme, Geri Elde Etme ve Örtüşme
Sayısal işlemcilerde, işaretler ayrık örnek dizisi olarak temsil edilir ve işaretler örneklendiğinde örtüşme olasılığı oluşur. İleride örtüşme olayı ayrıntılı olarak araştırılacaktır. Bu aşamada, örtüşmenin istenilmediğini ve Şekil 1'de gösterilen sistemin girişinde örtüşme önleyici bir filtre kullanılarak ve DAC'yi uygun bir şekilde tasarlayarak önlenebileceğini belirtmek yeterlidir. Temel band bir sistemde etkili bir örtüşme önleyici filtre, örnekleme frekansının yarısından az frekansları geçiren, ancak örenkleme frekansının yarısına eşit veya daha büyük frekansları oldukça zayıflatan bir filtredir. Temelband bir sistem için uygun bir DAC, örtüşme önleyici filtreninkine benzer karakteristiklere sahip bir alçak geçiren filtreden oluşur. AIC23 AD/DA dönüştürücüsü, sayısal örtüşme önleyici ve geri elde edici filtreler içermektedir.

2 Giriş ve Çıkış İçin TLV320AIC23 (AIC23) Stereo AD/DA Dönüştürücü

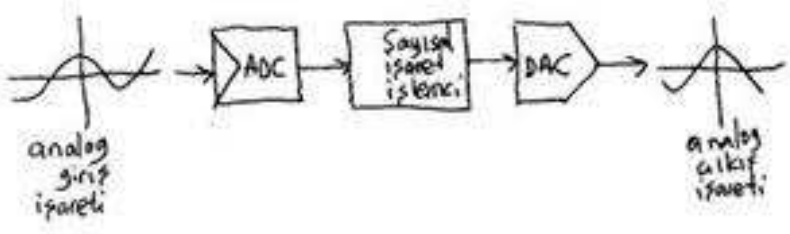
C6713 işlemcisi, analog giriş ve çıkış için TLV320AIC23 (AIC23) AD/DA dönüştürücüsü kullanmaktadır. Analog-sayısal dönüştürücü (ADC) kısmı, analog bir giriş işaretini sayısal işaret işlemcide işlenecek (16-bit işaretli tamasayı) örnek değerler dizisine dönüştürür. Sayısal-analog dönüştürücü (DAC) kısmı, sayısal işaret işlemciyle işlenen (16-bit işaretli tamsayı) örnek değerler dizisinden analog bir çıkış işaretini geri elde eder.

AIC23, ses uygulamalarına uygun, sigma-delta teknolojisine dayalı bir stereo AD/DA dönüştürücüdür. Dönüştürücünün fonksiyonel blok diyagramı Şekil 2'de gösterilmiştir.

12 Mhz bir kristal AIC23 (ve aynı zamanda DSP ile USB arayüzü) için saat işaretini sağlamaktadır. $250Fs$ ve $272Fs$ aşırı örnekleme oranlarıyla bu 12 MHz ana işaret kullanılarak 48 kHz (12MHz/250) ses örnekleme frekansı ve 44.1 kHz (12MHz/272) CD örnekleme frekansları elde edilebilir. AIC23'ün örnekleme frekansı 8,16,24,32,44,1 veya 96kHz'e ayarlanabilir.

Giriş ve çıkış için AIC23 ile haberleşme, C6713 üzerinde iki adet çokkanallı tamponlanmış seri port (McBSPs) kullanır. AIC23'e 16-bit bir kontrol kelimesi göndermek için McBSP0 tekyönlü bir kanal olarak kullanılır. Ses verisi almak ve göndermek için McBSP1 çiftyönlü bir kanal olarak kullanılır. AIC23, 16,20,24 veya 32-bit uzunluklu veri transferi yapacak şekilde ayarlanabilir.

AIC içindeki LINE IN ve HEADPHONE OUT işaret yolları sırasıyla 1.5 dB aralıklarla 12'den -34 dB'e ve 1 dB aralıkla 6'dan -73 dB'e kadar aralıkta ayarlanabilen ayarlanabilir kazanç elemanları içermektedir. C6713 DSK ile tümleştirilmiş AIC23'ün bir diyagramı CCS proğramında mevcut *6713_dsk_schem_pdf* belgesinde mevcuttur. Bazı durumlar hariç, bu laboratuardaki proğramların çoğunda AIC23 8 kHz örnekleme frekansında, LINE IN ve HEADPHONE OUT işaret yollarında 0-dB kazançla 32-bit veri transferi yapacak şekilde ayarlanmıştır.



Şekil 1: Temel dsp sistemi.

LINE IN girişlerinde AIC23'e izin verilen en büyük giriş işaret seviyesi rms cinsinden 1V'dur. Ancak, C6713 DSK üzerinde LINE In girişlerindeki izin verilen en büyük işaret seviyesi rms cinsinden 2V olacak şekilde işlemcinin LINE IN girişi ile AIC23'ün LINE IN girişi arasında kazancı 0.5 olan gerilim bölücü bir devre vardır. 2V (rms) seviyesinin üzerinde giriş işaretleri bozulacaktır. DSK üzerindeki giriş ve çıkış soketleri AIC23 ile ac olarak bağlıdır.

3 C Kullanılarak Proğramlama Örnekleri

Aşağıdaki örneklerde DSK kullanarak analog giriş ve çıkışın nasıl yapıldığı anlatılacaktır. Örneklerin amacı, hem DSK donanımını hem de CCS geliştirme ortamını tanıtmaktır. Örnek proğramlar, analog-sayısal dönüştürme ve sayısal-analog dönüştürme ile ilgili örnekleme, örtüşme ve geri elde etmeyi içeren önemli bazı kavramları açıklamaktadır. Örnekler, DSK kullanılarak gerçek zaman uygulamalarını gerçekleştirmek için kesme kullanımını da izah etmektedir. Bu deneyde açıklanan kavram ve yöntemlerin çoğu, diğer deneylerde tekrar kullanılacaktır.

Örnek 1: Yoklamayla Giriş ve Çıkış (loop_poll)

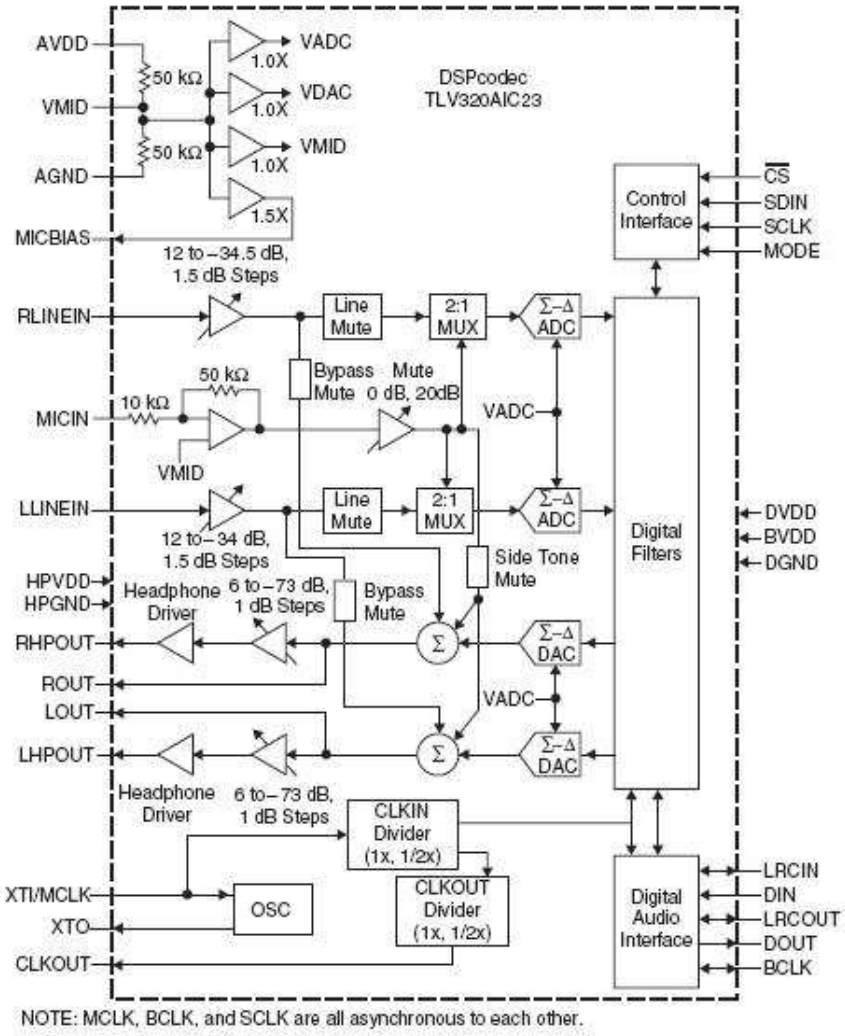
AIC23'ün ADC kısmından okunan giriş örneklerini herhangi bir değişiklik yapmadan AIC23'ün DAC kısmına çıkış olarak gönderen C kaynak dosyası Şekil 3'de verilmiştir. Aslında, AIC23 ve sayısal işaret işlemci aracılığıyla DSK üzerindeki MIC girişi HEADPHONE OUT çıkışına doğrudan bağlanmıştır. `loop_poll.c` dosyası, gerçek zaman giriş ve çıkış için birinci deneyde tanıtılan `sine8_LED.c` proğramının kullandığı kullandığı yoklama yönteminin aynısını kullanır.

c6713dskinit.c Destek Dosyasında Tanımlı Giriş ve Çıkış Fonksiyonları
`input_left_sample()`, `output_left_sample()` ve `comm_poll()` fonksiyonları `c6713dskinit.c` destek dosyasında tanımlıdır. Böylece, `loop_poll.c` C kaynak dosyası mümkün olduğu kadar küçük tutulmuş ve dikkat dağıtması muhtemel olan düşük seviye detayı saklanmış olur. Laboratuarda tanıtılacak örnekleri yapabilmek için `c6713dskinit.c`'de tanımlı bu ve diğer fonksiyonların detaylarını incelemeye gerek yoktur. Ancak, bütünlük olması bakımından fonksiyonlar tanıtılacaktır.

`DSK6713bsl.lib` destek kütüphane dosyasındaki düşük seviyeli fonksiyonlar, `input_left_sample()` ve `output_left_sample()` ile ayrıca çağrılır. `comm_poll()` fonksiyonu, DSK'yı ve özellikle örenkleme frekansı (`loop_poll.c`'de tanımlanmış) `fs` değişkenine, giriş kaynağı (`loop_poll.c`'de tanımlanmış) `input-source` değişkeninin değerine göre ayarlanacak ve yoklama seçilecek şekilde AIC23'ü başlangıç konumuna getirir. Diğer AIC23 konfigürasyon ayarları, `c6713dskinit.h`'de belirtilen parametrelerle belirlenir. Bu parametreler, LINE IN ve HEADPHONE çıkış işaretleri yollarını, sayısal ses arayüz formatını vb. içermektedir. Bu parametrelerin benzer değerleri laboratuarda tartışacağımız hemen hemen tüm proğram örneklerinde kullanılmaktadır. Çok nadir durumlarda değiştirileceklerinden, bu parametrelerin `c6713dskinit.h` dosyasında gizlenmesi faydalıdır.

Örnekleme frekansı ve giriş kaynağı ayarları bir proğramdan proğrama genelde değiştirildiğinden değerleri `fs` ve `inputsource` değişkenlerinin başlangıç değerlere verilerek ayarlanır. Bu değerler, `c6713dskinit.c` dosyasındaki `dsk6713_init()` fonksiyonunda sırasıyla `DSK6713_AIC23_ setFreq()` ve `DSK6713_ AIC23_ rset()` fonksiyonları tarafından kullanılır.

Yoklamada, `input_left_sample()` fonksiyonu, yeni verinin `MCBSP_read()` fonksiyonu kullanılarak okunmaya hazır olduğunu belirten seri port kontrol kütüğünün (SPCR) almaya hazır bitini (RRDY) yoklar veya test eder. `input_left_sample()` fonksiyonu, AIC23'ün yeni bir çıkış örneğini almaya hazır



Şekil 2: TLV320AIC23 blok diyagramı

olduğunu belirten SPCR'nin gödermeye hazır bitini (XRDY) yoklar veya test eder. Yeni bir çıkış örneği, `McBSP_ write()` fonksiyonu kullanılarak AIC23'e gönderilir.

Yoklama, `sine8_ buf.c`'de (ve ilerleyen haftalarda kullanılacak hemen hemen diğer tüm proğramlarda) kullanılan kesme yönteminden basit olmasına rağmen, işlemci zamanının çoğunu AIC23'ün veri almaya ve gödermeye hazır olup olupmadığını sürekli bir şekilde kontrol etmeye ayırdığından daha verimsizdir.

Proğramın Çalıştırılması

`loop_ poll.pjt` proje dosyası `loop_ poll` klasöründedir. `loop_ poll.pjt` projesini açınız ve çalıştırılabilir `loop_ poll.out` dosyasını işlemciye yükleyiniz. Proğramı çalıştırınız, bir mikrofon ve hoparlörler kullanarak proğramın istenilen şekilde çalıştığını doğrulayınız.

Proğramı daha yakından incelemek amacıyla bir işaret üreteci ve osiloskop kullanabilirsiniz. İşaret üretecini LINE IN girişine bağlamadan önce, DSK üzerinde giriş olarak MIC yerine LINE IN seçmek için

```
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC;
```

şeklinde yazan satırı

```
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;
```

şeklinde değiştirdikten sonra proğramı yeniden derlemelisiniz.

```
//loop_poll.c loop program using polling
#include "DSK6713_AIC23.h"         //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

void main()
{
  short sample_data;

  comm_poll();             //init DSK, codec, McBSP
  while(1)     //infinite loop
  {
    sample_data = input_left_sample(); //input sample
    output_left_sample(sample_data);   //output sample
  }
}
```

Şekil 3: Yoklama kullanılarak çevrim proğramı `loop_poll.c`.

İzin Verilen Giriş İşareti Genliğinin Testi

2V p-p genlikli ve yaklaşık 1 kHz frekanslı sinüzoidal bir giriş dalga şeklini DSK üzerindeki LINE IN girişine uygulayınız. DSK'nın LINE OUT çıkışını bir osiloskoba bağlayınız ve 1 kHz frekanslı ancak 1V p-p genlikli bir sinüzoidal işaret elde ettiğinizi gözlemleyiniz. Genlikteki zayıflama, LINE IN ve AIC23 girişi arasındaki gerilim bölücü devreden kaynaklanmaktadır.

AIC23'deki ADC ve DAC'nin tam ölçek aralığı 1V rms'dir (2.83V p-p).Giriş işaretinin genliğini 2V rms'den daha büyük yapınızve çıkış işaretinin bozulduğunu gözlemleyiniz.

AIC23'ün LINE IN Kazancının Değiştirilmesi

AIC23, sol ve sağ giriş kanalı ses kontrol kütüklerine farklı değerler yazarak sol ve sağ griş kanallarının kazançlarının 1.5 dB aralıklarla bağımsız olarak ayarlanmasına imkan vermektedir. `comm_poll()` fonksiyonu aracılığıyla bu kütüklere yazılan değerler, `c6713dskinit.h` başlık dosyasında tanımlıdır. Yazılan değerleri değiştirmek için `c6713dskinit.h` başlık dosyası değiştirilmelidir.

1. Orijinal başlık dosyasını değiştirmemek için, `Support` klsöründeki `c6713dskinit.h` ve `c6713dskinit.c` dosyalarını `loop_poll` klasörüne kopyalayınız.
2. *Project View* penceresinde `c6713dskinit.c` üzerinde sağ tıklayarak ve *Project-Remove from Project* seçerek `loop_poll` projesinden `c6713dskinit.h` ve `c6713dskinit.c` dosyalarını çıkartınız.
3. `loop_poll` klasöründeki `c6713dskinit.c` dosyasının kopyasını *Project-Add Files to Project* seçerek projeye ekleyiniz.
4. *Project View* penceresinde `c6713dskinit.c` dosyasının üzerinde sağa tıklayarak ve *Properties* seçerek dosyanın kopyasını eklediğinizi teyit edin.
5. `c6713dskinit.h` dosyasını `loop_poll` klasöründeki kopya ile değiştirmek için *Project-Scan all Dependencies* seçiniz.
6. Projeye eklenen (ve `loop_poll` klasöründe saklanan) `c6713dskinit.h` dosyasının kopyasını,

   ```
   0x0017 /* Set-Up Reg Left line volume control */
   ```

   yazan satırı

   ```
   0x001B /* Set-Up Reg Left line volume control */
   ```

   şeklinde değiştirerek yeniden düzenleyiniz. Bu işlemler, AIC23'ün sol kanal giriş kazanç kütüğüne

yazılan değeri 0x0017'den 0x001B'ye değiştirmektedir ve sonuçta kazanç 0 dB'den 6 dB'e artmaktadır.

7. Projede kullanılan `c6713dskinit.c` dosyasının kopyasının `loop_poll` klasöründeki kopya olduğundan emin olduktan sonra projeyi derleyiniz. Projeye eklenecek `c6713dskinit.h` başlık dosyası `loop_poll` klasöründen gelecektir.

8. Çalıştırılabilir `loop_poll.out` dosyasını işlemciye yükleyerek çalıştırınız. Çıkış işaretinin zayıflamadığını, aksine giriş işareti ile aynı genliğe (yani 2V p-p) sahip olduğunu gözlemleyiniz. Yaptığınız değişiklikler, `loop_poll` klasöründeki `c6713dskinit.h` dosyasının bir kopyasına uygulanmıştır ve sadece bu projeyi etkilemiştir.

Örnek 2: Kesme Kullanılarak Temel Giriş ve Çıkış (loop_ intr)

`loop_intr.c` proğramı, fonksiyonel olarak `loop_poll.c` proğramına eşdeğerdir, ancak kesme kullanmaktadır. Diğer laboratuarlardaki proğramların büyük çoğunluğu kesmeye dayalı olduğundan bu basit proğram önemlidir. Bir giriş işaretini temsil eden örnek dizisini DAC çıkışına değiştirmeden göndermek yerine, her defasında yeni bir giriş örneği alındığında br sayısal filtreleme gerçekleştirilebilir. Bu açıdan bakıldığında, `loop_intr.c` proğramının nasıl çalıştığını anlamak önemlidir. `main()` fonksiyonunun içinde, ildeğerlendirme fonksiyonu `comm_intr()` çağrılır. `comm_intr()`, `comm_poll()` fonksiyonuna benzemektedir, ancak DSK, ADC/DAC, MCBSP'yi ilk değerlendirmek ve yoklama modunu seçmemeye ek olarak,

```
Uint32 fs=DSK6713\_AIC23_FREQ\_8KHZ; //set sampling rate
```

satırıyla tanımlanan örnekleme frekansında AIC23 giriş analog işaretini örnekleyecek ve C6713 işlemcisnin çalışmasını durduracak (işlemciye kesme gönderecek9 şekilde kesmeler oluşturur. Ayrıca, McBSP aracılığıyla AIC23 ile haberleşmeyi de başlatır.

Bu örnekte, 8 kHz örnekleme frekansı kullanıldığından kesmeler 0.125 ms aralıklarla oluşacaktır. (16,24,32,44.1,48 ve 96 kHz örnekleme frekansları da mümkündür).

İlk değerlemeden sonra, `main()` fonksiyonu sonsuz bir döngüye girer, hiçber şey yapmadan kesmenin gelmesini bekler. farklı kesmeler için kesme servis rutini olarak vazife görecek fonksiyonlar `vectors_intr.asm` dosyasındaki kesme servis tablosunda belirtilmiştir. Assembly dili dosyası, `c_int11()` fonksiyonu INT11 kesmesi için kesme servis rutini olarak tanımlanması açısından `vectors_poll.asm`'den farklıdır.

Kesme oluştuğunda, `c_int11()` kesme servis rutini (ISR) çağrılır ve ISR içinde en önemli proğram ifadeleri çalıştırılır. `output_left_sample()` fonksiyonu, `input_left_sample()` fonksiyonu kullanılarak AIC23'den okunan bir veriyi dışarıya göndermede kullanılır.

AIC23'e İletilen ve AIC23'den Okunan Verinin Formatı

AIC23 ADC'si sol ve sağ kanal analog giriş işaretlerini 16-bit işaretli tamsayılara, DAC ise 16-bit işaretli tamsayıları sol ve sağ kanal analog çıkış işaretlerine dönüştürür. Sol ve sağ kanal örnekleri, McBSP aracılığıyla C6713 işlemcisine gönderilmek veya işlemciden okumak amacıyla 32-bit sayılar oluşturmak için birleştirilir. C proğramından ADC ve DAC'ye erişim, `Uint32 input_sample(), short input_left_sample(), short input_right_sample(), void output_sample(int out_data), void output_left_sample(short out_data)` ve textttvoid output_right_sample(short out_data) fonksiyonları aracılığıyla yapılır.

`input_sample()` ile elde edilen ve `output_sample()` fonksiyonuna gönderilen 32-bit işaretsiz tamsayılar (`Uint32`) sol ve sağ kanal örneklerin ikisini de içermektedir. `dsk6713init.h` dosyasında

```
union  {
  Uint32 uint;
  short channel [2];
} AIC_data;
```

ile verilen ifade, sol ve sağ kanal örnek değerlerini içeren bir 32-bit işaretsiz tamsayı (`AIC_data.uint` veya iki adet 16-bit işaretli tamsayı (`AIC_data.channel[0]` ve `AIC_data.channel[1]` olarak işlenilebilecek bir değişken tanımlar.

Laboratuar boyunca proğramların çoğunluğunda giriş ve çıkış için sadece bir kanal kullanılmıştır ve açıklık olması açısından çoğu proğramlarda `input_left_sample()` ve `output_left_sample()` fonksiyonları kullanılmıştır. Bu fonksiyonlar, sol ve sağ kanal 16-bit tamsayı örneklerinin AIC23'den alınan veya AIC23'e gönderilen 32-bit kelimelere dönüştürülme işleminin gerçekleştirildiği `c6713dskinit.c` dosyasında tanımlıdır.

```
//loop_intr.c loop program using interrupts
#include "DSK6713_AIC23.h"         //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

interrupt void c_int11()         //interrupt service routine
{
   short sample_data;

   sample_data = input_sample(); //input data
   output_sample(sample_data);   //output data
   return;
}

void main()
{
  comm_intr();                 //init DSK, codec, McBSP
  while(1);                    //infinite loop
}
```

Şekil 4: Kesme kullanılarak çevrim proğramı (`loop_intr.c`)

Proğramın Çalıştırılması

`c6713dskinit.c` ve `vectors_intr.asm` destek dosyalarının projeye dahil edildiğinden emin olduktan sonra bu projeyi `loop_intr` olarak oluşturun ve derleyin. `loop_poll` proğramıyla elde edilen sonuçların aynısını elde ettiğinizi teyit edin.

Örnek 3: Gecikme Oluşturmak Amacıyla loop_ intr.c Proğramının Değiştirlmesi (delay)

Örnekleri girişten çıkışa geçerken geciktirerek basit fakat çarpıcı bazı etkiler elde edilebilir. Şekil 'de verilen `delay.c` proğramı, gecikme oluşturulmasını açıklamaktadır. `buffer` dizisi kullanılarak, ADC'den okunurken örneklerin saklanması amacıyla bir gecikme gerçekleştirilir. Dizinin tüm elemanlarına ilk değer verildikten sonra, en geç saklanmış veri mevcut veya en yeni giriş örneğiyle değiştirilir. `buffer` dizisinde en son saklanmış değer yeni değerle değiştirilmeden önce, bu değer okunur, mevcut giriş değerine eklenir ve DAC'ye gönderilir. T ile isimlendirilen blok T saniye gecikmeyi temsil etmek üzere, Şekil 6,`delay.c` proğramının çalışmasını göstermektedir. Bu projeyi, `delay` olarak derleyin ve çalıştırın, mikrofon ve kulaklık kullanarak çalışmasını doğrulayın.

Örnek 4: Yankı Oluşturmak Amacıyla loop_ intr.c Proğramının Değiştirlmesi (echo)

Gecikme biriminin çıkışının bir kısmı girişe geribesleme yapılarak sönümlemeli bir yankı etkisi oluşturulabilir. Şekil 7'de verilen `echo.c` proğramı, bu amaçla yazılmıştır. Şekil 8, `echo.c` proğramının çalışmasının blok diyagram temsilini göstermektedir.

`BUF_SIZE` sabitinin değeri, `buffer` dizisinde saklanan örnek sayısını ve dolayısıyla gecikme süresini belirlemektedir. `GAIN` sabitinin değeri, girişe geri besleme yapılan gecikme çıkışı oranını ve dolayısıyla yankı etkisinin hangi hızda sönümlendiğini belirlemektedir. `GAIN` değerini 1'e eşit veya daha büyük yapmak geribeslemenin karasız olmasına neden olur. Bu projeyi, `echo` olarak derleyin ve çalıştırın. `GAIN` (0 ile 1 arasında) ve `BUF_SIZE` (100 ile 8000 arasında) için farklı değerler seçiniz. Parametrelerinin değerini değiştirmek için `echo.c` kaynak dosyası değiştirilmeli ve proje yeniden derlenmelidir.

Örnek 5: Gecikme ve Geribeslemenin GEL Çubuğu Kontrolüyle Yankı (echo_ control)

Bu örnek, yankı miktarını belirleyen kazanç ve geribesleme parametrelerinin gerçek-zamanda değiştirilmesine imkan vermek için Örnek 4'ü genişletmektedir. `echo_control.gel` dosyasında tanımlanan iki

```
//delay.c Basic time delay
#include "DSK6713_AIC23.h"              // codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;      //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

#define BUF_SIZE 8000
short input,output,delayed;
short buffer[BUF_SIZE];
int i;

interrupt void c_int11()  // interrupt service routine
{
  input = input_left_sample(); //read new input sample
  delayed = buffer[i];         //read output of delay line
  output = input + delayed;    //output sum of new and delayed samples
  buffer[i] = input;           //replace delayed sample with
  if(++i >= BUF_SIZE) i=0;     //new input sample then increment
  output_left_sample(output);  //buffer index
  return;                      //return from ISR
}

void main()
{
  for(i=0 ; i<BUF_SIZE ; i++)
    buffer[i] = 0;
  comm_intr();                  //init DSK, codec, McBSP
  while(1);                     //infinite loop
}
```

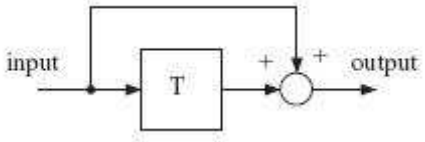
Şekil 5: Kesme kullanılarak gecikme proğramı (`delay.c`).

GEL çubuğu kullanılmaktadır. Şekil 9'de verilen `echo_control.c` proğramı, aşağıda listelenen yönlerden `echo.c` proğramından farklıdır:

1. `MAX_BUFFER_SIZE` boyutunda bir `buffer` dizisi tanımlanır.
2. Değişken uzunluklu gecikme elde etmek amacıyla `buffer` dizisi kullanılarak gerçekleştirilen dairesel tamponun uzunluğunu kontrol etmek için tamsayı değerler alan `buflength` değişkeni kullanılır. Dizinin elemanlarına nerişmede kullanılan `i`'nin değeri izin verilen maksimum değeri (`MAX_BUFFER_SIZE`) gectiğinde, `echo.c` proğramında olduğu gibi sıfıra değil, `MAX_BUFFER_SIZE-buffer` değerine eşitlenir.

Projeyi, `echo_control` olarak derleyiniz. `echo_control` dosyasını yükleyip çalıştırınız. *File* $\rightarrow$ *LOAD GEL* seçiniz ve `echo_control.gel` dosyasını yükleyiniz. Kazanç ve gecikme çubuklarını göstermek için *GEL* $\rightarrow$ *echo_ control* seçiniz.

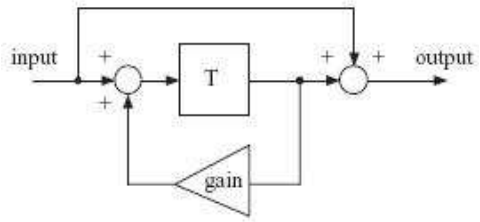
Örnek 6: Girişin Hafızada Saklandığı Çevrim Proğramı (loop_ buf)



Şekil 6: `delay.c` proğramının blok diyagram gösterilimi.

```
//echo.c echo with fixed delay and feedback
#include "DSK6713_AIC23.h"                 // codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;         // set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

#define GAIN 0.6                // fraction (0 - 1) of output fed back
#define BUF_SIZE 2000           // this sets length of delay
short buffer[BUF_SIZE];         // storage for previous samples
short input,output,delayed;
int i;                          // index into buffer

interrupt void c_int11()     // interrupt service routine
{
  input = input_left_sample();  // read new input sample from ADC
  delayed = buffer[i];          // read delayed value from buffer
  output = input + delayed;     // output sum of input and delayed values
  output_left_sample(output);
  buffer[i] = input + delayed*GAIN; // store new input and a fraction
                                // of the delayed value in buffer
  if(++i >= BUF_SIZE) i=0;      // test for end of buffer
  return;                       // return from ISR
}

void main()
{
  comm_intr();                  // init DSK, codec, McBSP
  for(i=0 ; i<BUF_SIZE ; i++)   // clear buffer
    buffer[i] = 0;
  while(1);                     //infinite loop
}
```

Şekil 7: Sönümlemeli yankı proğramı (`echo.c`).



Şekil 8: `echo.c` proğramının blok diyagram gösterilimi.

```
/echo_control.c echo with variable delay and feedback
#include "DSK6713_AIC23.h"                 // codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;         // set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

#define MAX_BUF_SIZE 8000       // this sets maximum length of delay
float gain = 0.5;
short buflength = 1000;
short buffer[MAX_BUF_SIZE];     // storage for previous samples
short input,output,delayed;
int i = 0;                      // index into buffer

interrupt void c_int11()     // interrupt service routine
{
  input = input_left_sample();  // read new input sample from ADC
  delayed = buffer[i];          // read delayed value from buffer
  output = input + delayed;     // output sum of input and delayed values
  output_left_sample(output);
  buffer[i] = input + delayed*gain; // store new input and a fraction
                                // of the delayed value in buffer
  if(++i >= MAX_BUF_SIZE)       // test for end of buffer
    i = MAX_BUF_SIZE - buflength;
  return;                       // return from ISR
}

void main()
{
  for(i=0 ; i<MAX_BUF_SIZE ; i++)   // clear buffer
    buffer[i] = 0;
  comm_intr();                  // init DSK, codec, McBSP
  while(1);                     //infinite loop
}
```

Şekil 9: Değişken geribesleme kazançlı ve gecikmeli yankı proğramı (`echo_control.c`).

Şekil 13'de verilen `loop_buf.c` proğramı kesmeye dayalıdır ve `loop_buf` klasöründe saklıdır. `loop_intr.c` proğramına benzerdir, ancak `buffer` dizisinin içindeki en son `BUF_SIZE` adet giriş örnek değerlerini içeren dairesel bir tampon içermektedir. Bu nedenle, proğramın çalışması durdurulduktan sonra bu verinin CCS'de çizilmesi mümkündür.

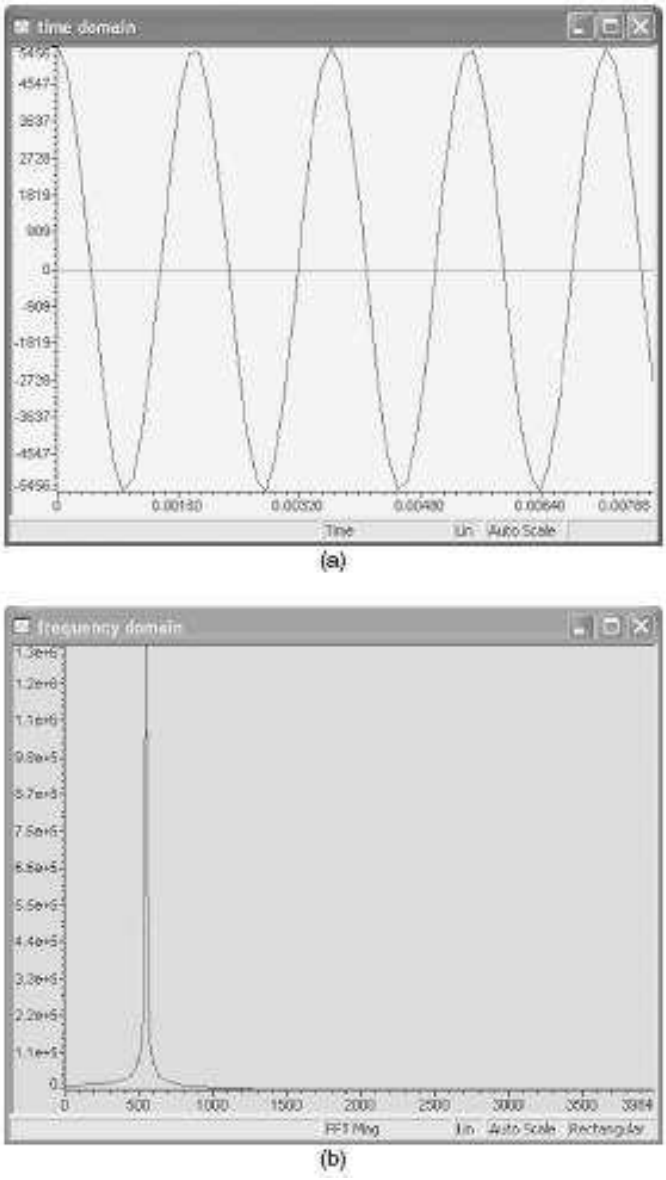
Bu projeyi texttt**loop_buf** olarak derleyiniz. 100 ila 3500 Hz arasında frekanslı sinüzoidal bir işaret uygulamak için LINE IN girişine bağlı bir işaret üreteci kullanın. Proğramı kısa bir süre çalıştırdıktan sonra durdurunuz ve *View* → *Graph* → *Time/Frequency* seçerek `buffer` dizisinin içeriğini çizdiriniz. Şekil 12 ve 13, verinin zaman ve frekans uzayı gösterilimlerini ve bu gösterilimleri elde etmede kullanılan parametrelerin değerlerini göstermektedir. Şekiller elde edilirken 550 Hz giriş frekansı kullanılmıştır.

```
//echo_control.gel

menuitem "echo control"

slider gain(0,18,1,1,gain_parameter)
{
  gain = gain_parameter*0.05;
}

slider delay(1,20,1,1,delay_parameter)
{
  buflength = delay_parameter*100;
}
```

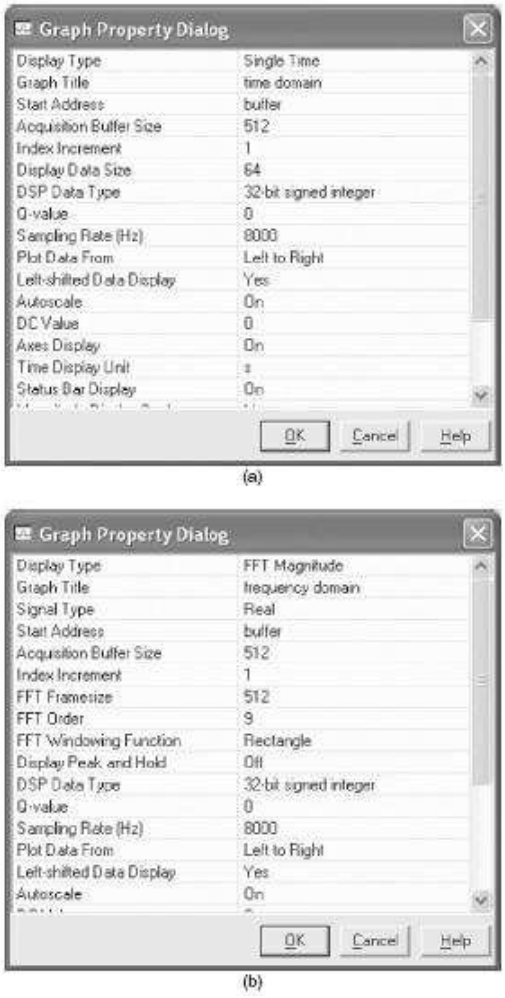
Şekil 10: `echo_control.c` proğramında gecikme ve geribeslemenin çubukla kontrolü için `echo_control.gel` GEL dosyası.

```
//loop_buf.c loop program with storage
#include "DSK6713_AIC23.h"                  // codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;          //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input
#define BUFSIZE 512

int buffer[BUFSIZE];
int buf_ptr = 0;

interrupt void c_int11()          // interrupt service routine
{
  int sample_data;

  sample_data = input_left_sample();    // read input sample
  buffer[buf_ptr] = sample_data;        // store in buffer
  if(++buf_ptr >= BUFSIZE) buf_ptr = 0; // update buffer index
  output_left_sample(sample_data);      // write output sample
  return;
}

void main()
{
  comm_intr();                    // init DSK, codec, McBSP
  while(1);                       // infinite loop
}
```

Şekil 11: Girişin hafızada saklandığı çevrim proğramı (`loop_buf.c`).



Şekil 12: `loop_buf.c` proğramı kullanılarak CCS ile çizilmiş 550 Hz sinüs dalgasına karşılık gelen giriş örnekleri: (a) Zaman uzayı, (b) frekans uzayı.



Şekil 13: `loop_buf.c` proğramı için kullanılan ve parametrelerin değerlerini gösteren *Graph Property Dialog* penceresi: (a) Zaman uzayı, (b) frekans uzayı.

EEM 437-SAYISAL İŞARET İŞLEME LABORATUARI
LAB 3: C6x İŞLEMCİSİNİN MİMARİSİ VE KOMUT KÜMESİ

1 C, Assembly ve Doğrusal Assembly Kullanılarak Proğramlama Örnekleri

Bu laboratuarda çeşitli proğramlama örnekleri tartışılacaktır. Örnekler, hem assembly hem de doğrusal assembly proğramlamayı açıklamaktadır: bir assembly proğramını çağıran bir C proğramı, bir doğrusal assembly proğramını çağıran bir C proğramı ve bir assembly proğramını bir çağıran assembly proğramı durumlarına ilişkin örnekler verilecektir. Bu laboratuardaki amacımız, assembly ve doğrusal assembly proğramlamanın formatını tanıtmaktır. Etkili ve verimli proğramlama yöntemleri başka bir laboratuarda ele alınacaktır.

Örnek 1: $\mathbf{1+2+\ldots+(n-1)+n}$ ***Toplamının Assembly Fonksiyonu Çağıran C Proğramıyla Hesaplanması (sum)***

Bu örnek, bir assembly proğramını çağıran bir C proğramını açıklamaktadır. Şekil 1'de verilen `sum.c` C kaynak proğramı, Şekil 2'de verilen `sumfunc.asm` assembly fonksiyonunu çağırmaktadır. Assembly fonksiyonu, $1+2+\ldots+(n-1)+n$ toplamını hesaplamaktadır. n değeri, ana C proğramında ayarlanmaktadır. Bu değer, A4 kütüğü aracılığıyla assembly fonksiyonuna gönderilmektedir. Örneğin, birden fazla değerin adresi, assembly fonksiyonuna A4,B4, A6, B6 vb. kütükleri aracılığıyla gönderilebilir. Assembly fonksiyonunda hesaplanan toplam, sonucu ekranda yazdıran C proğramındaki `result` değişkenine atanır.

Assembly fonksiyonunun isminden önce alt tire gelmelidir. Sadece, A1, A2, B0, B1 ve B2 kütükleri koşul ifadeleri için kullanılabileceğinden, A1 kütüğünü çevrim sayacı olarak kullanmak için, assembly fonksiyonunda A4 kütüğündeki n değeri A1 kütüğüne aktarılır. Daha sonra, A1'in değeri azaltılır. Proğramın çevrim kısmı, `LOOP` etiketi veya adresiyle başlar ve ilk dallanma ifadesi `B` ile sonlanır. İlk toplama, $n+(n-1)$ toplamını hesaplar ve sonucu A4 kütüğüne yazar. A1, $(n-2)$ değerine azaltılır. Dallanma, A1 kütüğünün içeriğine göre yapılır ve A1 sıfır olmadığından dallanma gerçekleştirilir, proğramın çalışması A4=$n+(n-1)$'in A1=$(n-2)$'ye eklendiği `LOOP` adresindeki komuta geçer. Bu işlem, A1=0 oluncaya kadar devam eder.

İkinci dallanma komutu, çağıran C proğramının dönüş adresi B3'dür. Sonuçlanan toplam, C proğramındaki `result` değişkenine geçirilen A4 kütüğünde saklıdır. Bir dallanma komutu beş gecikme aralığına karşılık geldiğinden komuttan sonra 5 `NOPS` (no operation) kullanılmıştır.

Seçilen `.S` ve `.L` fonksiyonel birimleri gösterilmiştir, ancak proğramda gerekli değildir. Proğramın verimliliğini arttırmak amacıyla hangi fonksiyonel birimlerin kullanıldığını incelemek için faydalı olabilirler. Benzer şekilde, `LOOP` etiketinden sonra üst üste iki nokta ve fonksiyon ismi gerekli değildir.

Bu projeyi, `sum` olarak derleyiniz ve çalıştırınız. C proğramının içinde n'nin değeri 6 olduğunda, 21 değerinin ekranda yazdırıldığını doğrulayınız.

Örnek 2: Bir Sayının Faktöryelinin Assembly Fonksiyonu Çağıran C Proğramıyla Hesaplanması (`factorial`)

Bu örnekte, bir sayının ($n \leq 7$) faktöryeli hesaplanacaktır. Assembly proğramlamanın formatı da açıklanmış olacaktır. Örnek, birinci örneğe benzemektedir. n'nin değeri, Şekil 4'de gösterilen `factfunc.asm` assembly fonksiyonunu çağıran ve Şekil 3'de verilen `factorial.c` C kaynak proğramında ayarlanır. Proğramlarda verilen yorumları okumak faydalıdır.

Önceki örnekte olduğu gibi, A1 kütüğü çevrim sayacı olarak kullanılır. Proğramın, `LOOP` adresiyle başlayan çevrim kısmında ilk çarpım $n(n-1)$'dir ve A4 kütüğünde saklıdır. n'nin ilk değeri, assembly fonksiyonuna A4 kütüğü aracılığıyla gönderilir. `MPY` komutu, bir gecikme çevrimi sürdüğünden komuttan

```
/Sum.c Finds n+(n-1)+...+1. Calls assembly function sumfunc.asm

#include <stdio.h>

main()
{
 short n=6;          //set value
 short result;              //result from asm function

 result = sumfunc(n);        //call assembly function sumfunc
 printf("sum = %d", result); //print result from asm function
}
```

Şekil 1: $1 + 2 + \ldots + (n-1) + n$ toplamını hesaplamak için bir assembly fonksiyonu çağıran C proğramı (`sum.c`).

```
;SUMFUNC.ASM Assembly function to find n+(n-1)+...+1

    .def   _sumfunc   ;function called from C
_sumfunc: MV   .L1  A4,A1    ;setup n as loop counter
    SUB  .S1  A1,1,A1    ;decrement n

LOOP:     ADD  .L1  A4,A1,A4 ;accumulate in A4
          SUB  .S1  A1,1,A1  ;decrement loop counter
    [A1]  B    .S2  LOOP     ;branch to LOOP if A1#0
          NOP   5          ;five NOPs for delay slots
          B    .S2  B3      ;return to calling routine
          NOP   5          ;five NOPs for delay slots
          .end
```

Şekil 2: `sum` projesinde, C proğramından çağrılan ASM fonksiyonu (`sumfunc.asm`).

sonra `NOP` kullanılmıştır. A1=0 oluncaya kadar, proğramın çevrim kısmındaki komutlar çalıştırılır. Fonksiyonel birimlerin, bu örnekte belirtilmediğine dikkat ediniz. Sonuçlanan faktöryel, çağıran C proğramına A4 kütüğü aracılığıyla gönderilir.

Bu projeyi, `factorial` olarak derleyiniz ve çalıştırınız. C proğramının içinde n'nin değeri 7 olduğunda, 5040 değerinin ekranda yazdırıldığını doğrulayınız. `result` değişkeninin tipi `short` olduğundan, n'nin en büyük değeri 7'dir (8!, `short` tipi ile temsil edilebilecek en büyük tamsayı 2^{15}'den büyüktür!).

Örnek 3: 32-bit Sözde Gürültünün Assembly Fonksiyonu Çağıran C Proğramıyla Üretilmesi(`noisegen_casm`)

Şekil 5'de gösterilen `noisegen_casm.c` C kaynak proğramı, aşağıda verilen yöntemi kullanarak 32-bit sözde gürültü dizisi üretmek için `noisegen_casmfunc.asm` (Şekil 6) dosyasındaki `noise_func` fonksiyonunu çağırmaktadır:

1. `0x7E521603` gibi bir 32-bit çekirdek değeri (16'lık tabanda) seçilir.
2. 17, 28, 30 ve 31. bitler ikili tabanda toplanır.
3. Toplamın en az anlamlı biti (LSB) seçilir. Bu bit, 0 veya 1'dir ve pozitif veya negatif bir değere göre uygun bir şekilde ölçeklenir.
4. Çekirdek değeri bir bit sola ötelenir, üçüncü adımda elde edilen bit değeri çekirdeğin LSB konumuna kopyalanır ve işlem yeni çekirdek değeri ile tekrarlanır.

32-bit gürültü üretme diyagramı Şekil 7'de verilmiştir. Assembly fonksiyonunun içinde, çekirdek değeri A4'den A1'e taşınır. Çekirdek değerini 17 bit sağa öteleme, 17. biti LSB konumuna getirir (toplama LSB'ler için anlamlıdır!). Önceden 17 bit sağa ötelenmiş 28. biti LSB konumuna getirmek için, sonuçlanan

```
//Factorial.c Finds factorial of n. Calls function factfunc.asm

#include <stdio.h>     //for print statement

void main()
{
 short n=7;      //set value
 short result;                //result from asm function

 result = factfunc(n);             //call assembly function factfunc
 printf("factorial = %d", result); //print result from asm function
}
```

Şekil 3: Bir sayının faktöryelini hesaplamak için bir assembly fonksiyonu çağıran C proğramı (`factorial.c`).

```
;Factfunc.asm  Assembly function called from C to find factorial

          .def _factfunc ;asm function called from C
_factfunc: MV A4,A1      ;setup loop count in A1
      SUB A1,1,A1   ;decrement loop count
LOOP:    MPY   A4,A1,A4  ;accumulate in A4
            NOP             ;for 1 delay slot with MPY
            SUB   A1,1,A1    ;decrement for next multiply
    [A1]    B     LOOP       ;branch to LOOP if A1 # 0
      NOP 5           ;five NOPs for delay slots
            B    B3      ;return to calling routine
            NOP 5 ;five NOPs for delay slots
            .end
```

Şekil 4: C proğramından çağrılan, bir sayının faktöryelini hesaplayan ASM fonksiyonu (`factfunc.asm`).

toplama 11 bit sağa ötelenir. Bu işlem, 17, 28, 30 ve 31. bitler toplanıncaya kadar tekrarlanır. Toplamın 0 veya 1 olan LSB'si, A4 kütüğünde saklanır ve sırasıyla negatif veya pozitif bir sayı olarak ölçeklendiği C proğramına gönderilir. Kesme her oluştuğunda, bu LSB bit (0 veya 1), gürültü örneğini temsil etmektedir.

Bu projeyi, `noisegen_casm` olarak derleyiniz ve çalıştırınız. Örnekleme frekansını 48 kHz seçerek, gürültü spektrumunun yaklaşık 23 kHz bandgenişlikli düzgün bir spektrum olduğunu doğrulayınız. Çıkışı bir hoparlöre bağlayınız ve gürültüyü dinleyiniz. Ölçekleme değerlerini ± 8000 şeklinde değiştirerek, üretilen gürültünün genliğinin azaldığını gözlemleyiniz.

Örnek 4: 4 Rakamlı Bir Şifrenin Assembly Fonksiyonu Çağıran C Proğramıyla Çözülmesi(`code_casm`)

Bu örnekte, başlangıç değeri değeri ana C kaynak proğramında ayarlanan dört rakamlı bir şifre çözülmektedir. Şekil 8, Şekil 9'de verilen `code_casmfunc.asm` assembly fonksiyonunu çağıran ana C kaynak proğramı `code_casm.c`'yi göstermektedir. Şifre `code1,...,code4` ile sırasıyla 1,2,2,4 olarak ayarlanmaktadır. 1,1,1,1 olarak ayarlanan `code1,...,code4`'ün ilk değerleri, dört şifre değeri ile karşılaştırılmak amacıyla assembly fonksiyonuna gönderilir. Assembly fonksiyonuna aktarılan rakamları değiştirmek için dört çubuk kullanılır. C kaynak proğramı, assembly fonksiyonu ve GEL dosyası `code_casm` klasöründedir.

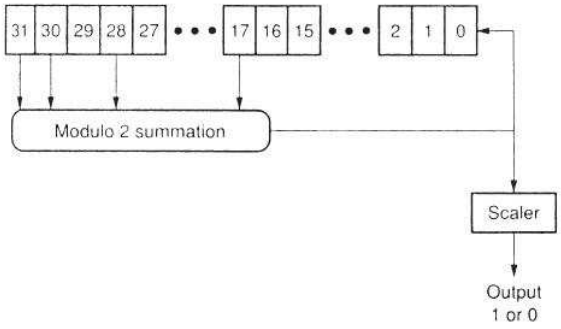
Bu projeyi `code_casm` olarak derleyiniz ve çalıştırınız. 0 nolu anahtarın (SW0) yönünü aşağıya doğru getirdiğinizde ekranda sürekli olarak eşleşmenin olmadığını belirten "no match" yqazıldığını doğrulayınız. `code_casm.gel` GEL dosyasını yükleyiniz ve `code1,...,code4` çubuklarını sırasıyla 1,2,2,4 konumlarına getiriniz. 0 nolu anahtarın (SW0) yönünü aşağıya doğruyken ekranda eşleşmenin olduğunu belirten "correct match" yazıldığını doğrulayınız. Örneğin, 2 nolu çubuğun konumu 3'e getirildiğinde ekranda eşleşmenin olmadığının belirtildiğini doğrulayınız. 3 nolu anahtarın (SW3) yönü aşağıya doğru olmadığı sürece proğram sonsuz bir döngünün içindedir. Şifrenin başlangıç değerinin (`code1,...,code4`) kolaylıkla değiştirilebileceğine dikkat ediniz.

```
//Noisegen_casm.c Pseudo-random noise generation calling ASM function

#include "dsk6713_aic23.h" //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_48KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select mic in
int previous_seed;
short pos=16000, neg=-16000; //scaling noise level

interrupt void c_int11()
{
previous_seed = noisefunc(previous_seed); //call ASM function

if (previous_seed & 0x01) output_left_sample(pos);//positive scaling
else output_left_sample(neg); //negative scaling
}

void main ()
{
comm_intr();   //init DSK, codec, McBSP
previous_seed = noisefunc(0x7E521603);  //call ASM function
   while (1); //infinite loop
}
```

Şekil 5: 32-bit gürültü dizisi üretmek için assembly fonksiyonu çağıran C proğramı (`noisegen_casm.c`).

```
;Noisegen_casmfunc.asm Noise generation C-called function

.def _noisefunc  ;ASM function called from C
_noisefunc ZERO A2 ;init A2 for seed manipulation
MV A4,A1 ;seed in A1
SHR A1,17,A1 ;shift right 17->bit 17 to LSB
ADD A1,A2,A2 ;add A1 to A2 => A2
SHR A1,11,A1 ;shift right 11->bit 28 to LSB
ADD A1,A2,A2 ;add again
SHR A1,2,A1 ;shift right 2->bit 30 to LSB
ADD A1,A2,A2 ;
SHR A1,1,A1 ;shift right 1->bit 31 to LSB
ADD A1,A2,A2 ;
AND A2,1,A2 ;Mask LSB of A2
SHL A4,1,A4 ;shift seed left 1
OR A2,A4,A4 ;Put A2 into LSB of A4
B B3 ;return to calling function
NOP 5 ;5 delays for branch
```

Şekil 6: 32-bit gürültü dizisi üretmek C proğramından çağrılan assebmly fonksiyonu (`noisegen_casmfunc.asm`).



Şekil 7: 32-bit gürültü ütretme diyagramı.

Örnek 5: İki Vektörün İç Çarpımının Assembly Fonksiyonu Çağıran Assembly Proğramıyla Hesaplanması(`dotp4a`)

Bu örnekte, 4 elemanlı iki vektörün iç çarpımı hesaplanacaktır. Hatırlanacağı gibi, birinci laboratuarda bu problem sadece C proğramları kullanılarak çözülmüştü. Şekil 10'da, iki vektöre başlangıç değeri veren ve Şekil 11'de verilen vektörlerin iç çarpımını hesaplayan `dotp4aafunc.asm` assembly fonksiyonunu çağıran `dotp4a_init.asm` assembly proğramı gösterilmiştir. `dotp4a_init.asm` assembly proğramı aynı zamanda B3 kütüğü aracılığıyla geri dönüş adresini ve A0 kütüğü aracılığıyla sonuç adresini saklamaktadır. İki vektörün adresleri ve vektör boyutu sırasıyla A4, A6 ve B4 kütükleri aracılığıyla `dotp4aafunc.m` assembly fonksiyonuna geçirilmektedir. Çağırılan fonksiyonda bulunan sonuç A4 kütüğü aracılığıyla ana proğrama gönderilmektedir. `STW` komutu, A4 kütüğündeki sonucu okumaktadır. A0 kütüğü, `result_addr` adresine sahip bir işareti olarak vazife yapmaktadır.

`MVK` komutu, en az anlamlı (LSB) 16 biti bir kütüğe aktarmaktadır. 32-bit adresli bir sonuç gerekliyse, sonucun alt ve üst 16 bitini akatarmak için `MVKL` ve `MVKH` komutları kullanılabilir. Çağıran assembly proğramının başlangıç adresi `init` olarak tanımlanmıştır. Giriş adresine referans `_c_int00`'dan `init` değerine değiştirilecek şekilde vektör dosyası değiştirilmiştir ve `dotp4a` klasöründe bulunabilir. Şekil 12'de gösterilen `vectors_dotp4a.asm` alternatif vektör dosyası, bu giriş adresine dallanma tanımlamaktadır. Çağrılan assembly fonksiyonu vektörlerin iç çarpımını hesaplamaktadır. A6 koşul komutları için kullanılamıyacağından çevrim sayası değeri A1'e taşınmıştır (sadece A1, A2, B0, B1 ve B2 koşul komutları için kullanılabilir). İki `LDH` komutu, *x_ addr* ve *y_ addr*'de başlayan iki dizinin adresini (16-bit yarı kelime) sırasıyla A2 ve B2 kütüklerine yükler. Örneğin,

```
LDH *B4++,B2
```

komutu, B4 kütüğüyle (ikinci vektörün adresi) işaret edilen bellekteki (ikinci vektördeem y_adrr adresinde başlayan ilk değer) içeriği yükler. Bir işaretçi olarak kullanılan B4 kütüğünün değeri, hafızada ikinci vektördeki ikinci elemanı bulunduran bir sonraki adrese arttırılır. Sonuç, çağıran fonksiyona A4 kütüğü aracılığıyla aktarıldığından çarpımları toplamak ve sonucu A4 kütüğüne taşımak için A7 kütüğü kullanılır.

Bu proje için destek dosyaları, aşağıdakileri içermektedir (herhangi bir kütüphane dosyası gerekli değildir):

1. `dotp4a_init.asm`
2. `dotp4afunc.asm`
3. `vecs_dotp4a.asm`

`vecs_dotp4a.asm` vektör dosyası (değiştirilmiş vektör dosyası) veya Şekil 12'de gösterilen alternatif vektör dosyası `vectors_dotp4a.asm` `dotp4a` klasöründe mevcuttur. Bu projeyi, `dotp4a` olarak derleyiniz ve çalıştırınız. "No Autoinitialization" seçmek için bağlayıcı ayarını değiştirin (Project → Options). Aksi takdirde, proje derlenirken "entry point symbol _c_init00 undefined" uyarısı gözükecektir (uyarı ihmal edilebilir). Uyarının gözükmesinin nedeni, assembly fonksiyonunun içinde *main()* fonksiyonu olmadığından projede geleneksel giriş noktası kullanılmamaktadır.

`dotp4a_init.asm` proğramında, aşağıda verilen ilk dallanma komutunda bir durdurma noktası oluşturun:

```
//Code_casm.c  Calls ASM function. Determines if code chosen match with slider values
#include <stdio.h>
short digit1=1, digit2=1, digit3=1, digit4=1;  //initialize slider values
main()
{
  short code1=1, code2=2, code3=2, code4=4;        //initialize code
  short result;
  DSK6713_init(); //init BSL
  DSK6713_DIP_init();          //init dip switches on DSK
 while(DSK6713_DIP_get(3) == 1)        //continue until SW #3 is pressed
 {
  if(DSK6713_DIP_get(0) == 0)       //true if DIP SW #0 is pressed
  {                                                  //call ASM function
  result=codefunc(digit1,digit2,digit3,digit4,code1,code2,code3,code4);
  if(result==0) printf("correct match\n"); //result from ASM function
  else printf("no match\n");
  }
 }
}
```

Şekil 8: Dört rakamlı bir şifreyi çözmek amacıyla assembly fonksiyonu çağıran C proğramı `code_casm.c`.

```
B dotp4afunc
```

View → *Memory* seçiniz, address alanını *result_ adrr*'ye ayarlayınız ve sayı formatı alanında *16 bit signed integer* tercihini seçiniz. Memory penceresinde sağ tıklayınız ve "Float in Main Window" tercihini kaldırınız. Bu sayede, `dotp4a_init.asm` kaynak dosyasını incelerken Memory penceresinin daha iyi bir görünümünw sahip olursunuz.

Proğramı çalıştırınız. Proğramın çalışması ilşk durma noktasında durur. *result_ adrr* adresinde hafızadaki içerik sıfırdır (çağırılan fonksiyon henüz koşturulmamıştır). Tekrar çalıştırın, proğramın çalışması

```
wait B wait ;wait here
```

ile verilen sonsuz bekleme çevriminde olduğundan proğramı tekrar durdurun. Sonuçlanan iç çarpımın 40 olduğunu doğrulayınız. A0 kütüğünün sonuç adresini (*result_ adrr* içerdiğine dikkat ediniz. *View* → *Registers* → *Core Registers* seçerek bu adresi teyit ediniz (16-tabanında).

Örnek 6: İki Vektörün İç Çarpımının Doğrusal Assembly Fonksiyonu Çağıran C Proğramıyla Hesaplanması(`dotp4clasm`)

Şekil 13'de, Şekil 14'de verilen `dotp4clasmfunc.sa` doğrusal assembly fonksiyonunu çağıran C kaynak proğramı `dotp4clasm.c` gösterilmiştir. Doğrusal assembly proğramları başlangıç ve bitimi sırasıyla *.cproc* ve *.endproc* ile belirtilir. Doğrusal assembly proğramımı çağıran proğram C dilinde yazıldığından çağırılan doğrusal assembly fonksiyonunun isminden önce alt tire kullanılır.

Assembly fonksiyonlarında olduğu gibi, fonksiyonel birimlerin kullanımı tercihe bağlıdır. *a,b,prod* ve *sum* kütükleri *.reg* kullanılarak tanımlanmaktadır. İki vektörün (x ve y) adresleri ve vektörlerin boyutu (count), *ap, bp* ve *count* kütükleri aracılığıyla doğrusal assembly fonksiyonuna geçirilmektedir. *ap* ve *bp*, C'de olduğu gibi işaretçiler olarak kullanılan kütüklerdir. Komut alanı assembly proğramına, geri kalan kısım ise C proğramlamaya benzemektedir. Örneğin,

```
loop: ldh * ap++,a
```

komutu, adresi *ap* kütüğüyle belirtilen hafıza içeriğini *a* kütüğüne yükler. Daha sonra, *ap* işaretçi kütüğü, x vektörünün ikinci elemanını içeren hafıza konumunu işaret edecek şekilde bir arttırılır. İki vektörün elemanlarının çarpımlarının toplamına eşit olan sonuç, çağıran C proğramına gönderilen *sum* değişkeninde saklanır.

Bu projeyi, `dotp4clasm` olarak derleyiniz ve çalıştırınız. Sonucun 40 olduğunu doğrulayınız. CCS'nin "profile clock" özelliğini kullanarak, birinci laboratuarda yaptığımıza benzer bir şekilde doğrusal assembly proğramını çalıştırmanın kaç çevrim gerektirdiğini belirleyebilir ve işlemi C'de yapmaya göre ne kadar

```
;Code_casmfunc.asm ASM function determines if code matches slider values
.def _codefunc ;ASM function called from C
_codefunc: MV A8, A2 ;correct code
MV    B8, B2
MV A10, A7
MV    B10, B7
CMPEQ A2,A4,A1 ;compare first digit(A1=1 if A2=A4)
CMPEQ A1,0,A1 ;otherwise A1=0
[A1] B DONE ;done if A1=0 since no match
NOP 5
MV    B2,A2
CMPEQ A2,B4,A1 ;compare second digit
CMPEQ A1,0,A1
[A1] B DONE
            NOP 5
MV    A7,A2
CMPEQ A2,A6,A1 ;compare third digit
CMPEQ A1,0,A1
[A1] B DONE
            NOP   5
            MV    B7,A2
CMPEQ A2,B6,A1 ;compare fourth digit
    CMPEQ A1,0,A1
DONE: MV A1,A4 ;return 1 if complete match
B B3 ;return to C program
      NOP 5
.end
```

Şekil 9: Dört rakamlı bir şifreyi çözmek amacıyla C proğramından çağrılan assembly fonksiyonu `code_casmfunc.asm`.

```
;Dotp4a_init.asm ASM program to init variables. Calls dotp4afunc.asm
      .def init       ;starting address
      .ref dotp4afunc ;called ASM function
      .text       ;section for code
x_addr    .short  1,2,3,4    ;numbers in x array
y_addr    .short   0,2,4,6    ;numbers in y array
result_addr .short   0       ;initialize sum of products

init      MVK result_addr,A4    ;A4=result addr
      MVK 0,A3        ;A3=0
      STH A3,*A4       ;init result to 0
      MVK  x_addr,A4   ;A4=address of x
      MVK y_addr,B4       ;B4=address of y
      MVK 4,A6       ;A6=size of array
      B dotp4afunc       ;branch to function dotp4afunc
      MVK ret_addr,b3       ;B3=return addr from dotp4a
      NOP 3                 ;3 more delay slots(branch)

ret_addr  MVK result_addr,A0    ;A0=result_addr
          STH A4,*A0       ;store result
wait   B wait       ;wait here
      NOP 5       ;delay slots for branch
```

Şekil 10: İki vektörün iç çarpımını hesaplamak için bir assembly fonksiyonu çağıran assembly fonksiyonu.

```
;Dotp4afunc.asm Multiply two arrays. Called from dotp4a_init.asm
;A4=x address,B4=y address,A6=count(size of array),B3=return address

  .def dotp4afunc  ;dot product function
       .text       ;text section
dotp4afunc MV A6,A1       ;move loop count -->A1
  ZERO   A7       ;init A7 for accumulation

loop   LDH  *A4++,A2    ;A2=(x. A4 as address pointer
    LDH  *B4++,B2    ;B2=(y). B4 as address pointer
      NOP 4       ;4 delay slots for LDH
      MPY .M1x B2,A2,A3    ;A3 = x * y
      NOP       ;1 delay slot for MPY
      ADD A3,A7,A7    ;sum of products in A7
           SUB A1,1,A1     ;decrement loop counter
    [A1]    B loop       ;branch back to loop till A1=0
      NOP 5       ;5 delay slots for branch

MV A7,A4       ;A4=result A4=return register
      B B3          ;return from func to addr in B3
      NOP  5       ;5 delay slots for branch
```

Şekil 11: İki vektörün iç çarpımını hesaplayan assembly fonksiyonu.

```
;vectors_dotp4a.asm Vector file for dotp4a project

        .ref      init      ;starting addr in init file
        .sect  "vectors" ;in section vectors
rst:    mvkl .s2 init,b0   ;init addr 16 LSB -->B0
        mvkh .s2 init,b0   ;init addr 16 MSB -->B0
        b   b0        ;branch to addr init
        nop 5
```

Şekil 12: Alternatif vektör dosyası.

```
//Dotp4clasm.c Multiplies two arrays using C calling linear ASM func

short dotp4clasmfunc(short *a,short *b,short ncount);  //prototype
#include <stdio.h>              //for printing statement
#include "dotp4.h"                          //arrays of data values
#define  count 4                            //number of data values
short x[count] = {x_array};   //declare 1st array
short y[count] = {y_array};      //declare 2nd array
volatile int result = 0;             //result

main()
{
  result = dotp4clasmfunc(x,y,count);   //call linear ASM func
  printf("result = %d decimal \n", result); //print result
}
```

Şekil 13: İki vektörün iç çarpımını hesaplamak amacıyla bir doğrusal assembly fonksiyonu çağıran C proğramı.

```
;Dotp4clasmfunc.sa  Linear assembly function to multiply two arrays
     .def   _dotp4clasmfunc ;ASM func called from C
_dotp4clasmfunc: .cproc   ap,bp,count     ;start section linear asm
           .reg   a,b,prod,sum    ;asm optimizer directive

           zero     sum          ;init sum of products
loop:            ldh      *ap++,a         ;pointer to 1st array->a
           ldh      *bp++,b       ;pointer to 2nd array->b
           mpy      a,b,prod        ;product= a*b
                 add      prod,sum,sum    ;sum of products-->sum
           sub      count,1,count   ;decrement counter
  [count]        b        loop            ;loop back if count # 0

     .return  sum       ;return sum as result
     .endproc              ;end linear asm function
```

Şekil 14: İki vektörün iç çarpımını hesaplayan doğrusal assembly fonksiyonu.

kazanç elde ettiğinizi belirleyebilirsiniz.

Örnek 7: Bir Sayının Faktöryelinin Assembly Fonksiyonu Çağıran C Proğramıyla Hesaplanması(`factclasm`)

('den küçük bir sayının faktöryelini hesaplamak amacıyla Şekil 16'da verilen `factclasmfunc.sa` doğrusal assembly fonksiyonunu çağıran C proğramı `factclasm.c`, Şekil 15'de gösterilmiştir. Örnek 2 ve 6, bu projeye temel teşkil etmektedir. Bu proje için destek dosyaları, `factclasm.c`, `factclasmfunc.sa`, `rts6700.lib`, ve `C6713dsk.cmd`'dir. Projeyi `factclasm` olarak derleyiniz ve çalıştırınız. 7 için sonucun 5040 olduğunu doğrulayınız.

```
//Factclasm.c Factorial of number. Calls linear ASM function

#include <stdio.h>     //for print statement

void main()
{
 short number = 7;      //set value
 short result;                //result of factorial

 result = factclasmfunc(number);   //call ASM function factlasmfunc
 printf("factorial = %d", result); //print from linear ASM function
}
```

Şekil 15: Bir sayının faktöryelini hesaplamak için bir doğrusal assembly fonksiyonu çağıran C proğramı.

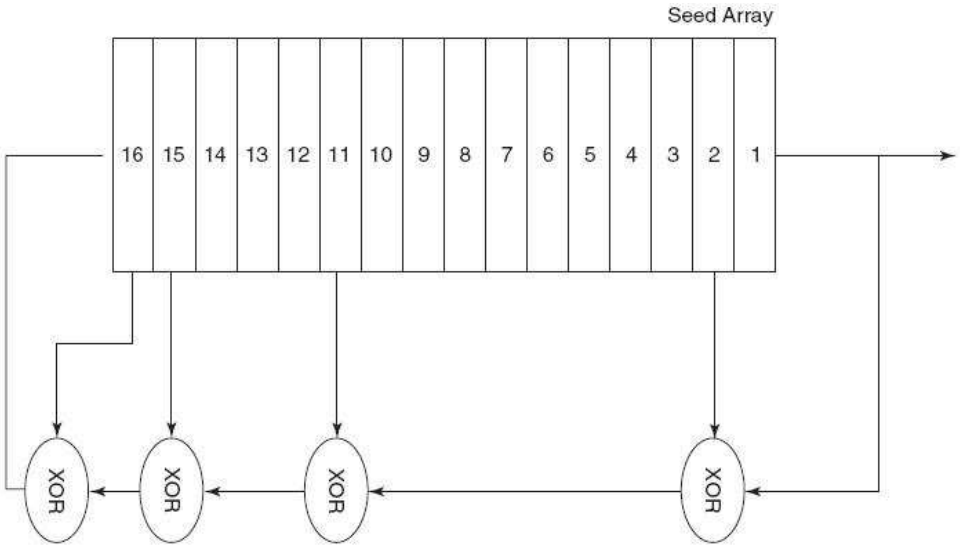
```
;Factclasmfunc.sa Linear ASM function called from C to find factorial

           .def _factclasmfunc ;Linear ASM func called from C
_factclasmfunc: .cproc number    ;start of linear ASM function
    .reg a,b             ;asm optimizer directive
    mv number,b       ;set-up loop count in b
    mv   number,a        ;move number to a
    sub     b,1,b           ;decrement loop counter

loop:     mpy a,b,a          ;n(n-1)
    sub b,1,b           ;decrement loop counter
   [b]      b loop            ;loop back to loop if count # 0
    .return a               ;result to calling function
    .endproc                ;end of linear asm function
```

Şekil 16: Bir sayının faktöryelini hesaplayan doğrusal assembly fonksiyonu.

2 Alıştırmalar

1. $[a^2+(a+1)^2+\ldots+(2a-1)^2]-[b^2+(b+1)^2+\ldots+(2b-1)^2]$ ifadesini hesaplamak için bir assembly fonksiyonu çağıran C proğramı yazınız. C proğramı gerekli değişkenleri assembly fonksiyonuna aktaracak, assembly fonksiyonu ise sonucu hesaplayıp çağıran fonksiyona göderecektir. C proğramının içinde $a=3$ ve $b=2$ seçip sonucun 37 olduğunu doğrulayınız.

2. 3×3 boyutlarında bir matrisin determinantını hesaplamak amacıyla bir assembly fonksiyonu çağıran bir C proğramı yazınız. Matrisin satırları C proğramında girilecek, assembly fonksiyonu ise determinantı hesaplayıp ana proğrama gönderecektir. Birinci satır [4,5,9]; ikinci satır [8,6,5] ve üçüncü satır [2,1,2] olduğunda sonucun -38 olduğunu doğrulayınız.



Şekil 17: LFSR kullanılarak gürültü üretme diyagramı.

3. Şekil 17'de verilen doğrusal geribeslemeli kaydırmalı kütük (LFSR) modeline dayalı olarak, rastgele bir gürültü dizisi üretmek amacıyla bir doğrusal assembly veya assembly fonksiyonu çağıran bir C proğramı yazınız. 16-bit bir çekirdek değeri yerine, bir dizideki 16 tamasayı çekirdekler olarak kullanılmaktadır. Böylece, her 32-bit çekirdek teorik bir bit olarak ele alınmaktadır. Şekilde gösterildiği gibi toplamanın yapıldığı 1,2,11,15 ve 16 nolu bitler geniş bir rastgele sayı dizisi üretmek için XOR işlemine uygulanmaktadır. Assembly veya doğrusal assembly fonksiyonu içinde her tamsayı bir çekirdek olarak alınır ve her bir çekirdeği yukarı taşımak için LDW/STW gibi komutları 15 kez kullanabilirsiniz. 1 ve 2 nolu bitleri XOR işlemine uygulayın, çıkan sonucu ve 11 nolu biti XOR işlemine uygulayın ve şekilde gösterildiği gibi bu işleme diğer bitler için tekrarlayın. Sonuçlanan çekirdek dizinin sonuna yerleştirilir ve işlem tekrarlanır. Çıkış, 32-bit bir değerdir. Örnekleme frekansını 8 kHz olarak seçin. Gürültü spektrumunun düzgün olduğunu ve yaklaşık 3.8 kHz'den sonra azalmaya başladığını gözlemleyiniz. Spektrumun 3.8 kHz'den sonra neden azalmaya başladığınıu açıklayınız.

SAYISAL İŞARET İŞLEME LABORATUARI

LAB 4: SONLU DÜRTÜ YANITLI (FIR) FİLTRELER

Bu bölümde aşağıdaki başlıklar ele alınacaktır.

- z- dönüşümü
- FIR filtrelerin tasarımı ve gerçekleştirilmesi
- C ve TMS320C6x kodları kullanılarak yapılan programlama örnekleri

4.1. z-Dönüşümü

z- dönüşümü, sürekli-zaman işaretler için kullanılan Laplace dönüşümüne benzer olarak, ayrık-zaman işaretlerin analizi için kullanılır. Laplace dönüşümünü analog bir filtreye ait diferansiyel eşitlikleri çözmek için, z-dönüşümünü ise sayısal bir filtreye ait fark eşitliklerini çözmek için kullanabiliriz. İdeal olarak örneklenmiş analog bir x(t) işaretini ala alalım

$$x_s(t) = \sum_{k=0}^{\infty} x(t)\delta(t-kT) \qquad (4.1)$$

Burada $\delta(t-kT)$, kT gecikmeli dürtü (impuls) fonksiyonu ve $T = 1/F_s$ örnekleme periyodudur. $X_s(t)$, t=kT dışında her yerde sıfırdır. $X_s(t)$'nin Laplace dönüşümü

$$\begin{aligned} X_s(s) &= \int_0^{\infty} x_s(t)e^{-st}dt \\ &= \int_0^{\infty} \{x(t)\,\delta(t) + x(t)\,\delta(t-T) + \cdots\}e^{-st}dt \end{aligned} \qquad (4.2)$$

eşitliği ile verilir. Dürtü fonksiyonunun

$$\int_0^{\infty} f(t)\delta(t-kT)dt = f(kT)$$

özelliğinden, Eşitlik (4.2)

$$X_s(s) = x(0) + x(T)e^{-sT} + x(2T)e^{-2sT} + \cdots = \sum_{n=0}^{\infty} x(nT)e^{-nsT} \qquad (4.3)$$

şeklinde düzenlenebilir. Eşitlik (4.3)'de $z = e^{sT}$ yazılırsa

$$X(z) = \sum_{n=0}^{\infty} x(nT)z^{-n} \qquad (4.4)$$

elde edilir. T örnekleme periyodu olmak üzere, x(nT), x(n) olarak yazılabilir. Bu durumda aşağıdaki eşitlik

$$X(z) = \sum_{n=0}^{\infty} x(n)z^{-n} = ZT\{x(n)\} \qquad (4.5)$$

x(n)'in z-dönüşümünü (ZT) göstermektedir. z-dönüşümünde x(n) ile X(z) arasında birebir bir ilişki vardır.

Örnek 4.1. x(n)=e^{nk} Üstel Fonksiyonun z-dönüşümü

n≥0 ve k sabit olmak üzere x(n)= e^{nk}'nın z-dönüşümü

$$X(z) = \sum_{n=0}^{\infty} e^{nk} z^{-n} = \sum_{n=0}^{\infty} (e^{k} z^{-1})^{n} \qquad (4.6)$$

Geometrik seri kullanılarak, Taylor serisi açılımından elde edilen

$$\sum_{n=0}^{\infty} u^{n} = \frac{1}{1-u} \qquad |u| < 1$$

eşitliği kullanılarak $|e^{k}z^{-1}| < 0 \; ve \; |z| > |e^{k}|$ için

$$X(z) = \frac{1}{1-e^{k}z^{-1}} = \frac{z}{z-e^{k}} \qquad (4.7)$$

elde edilir. k=0 ise x(n)=1'in z-dönüşümü *X*(*z*)=*z*\(*z-1*)'dir.

4.1.1. s-Düzleminden z-Düzlemine Geçiş

Laplace dönüşümü bir sistemin kararlılığını belirlemek için kullanılabilir. Eğer sistemin kutupları s-düzleminde *jw* ekseninin sol tarafında ise, sistem cevabı zamanla azalır ve sistem kararlıdır. Eğer kutuplar *jw* ekseninin sağ tarafında ise, sistem cevabı zamanla artar ve sistem kararsız hale gelir. *jw* ekseninin üzerinde buluna kutuplar ya da sanal kutuplar sinüzoidal bir cevaba neden olurlar. Sinüzoidal frekans *jw* ekseni ile gösterilir ve *w*=0 dc'yi ifade eder.

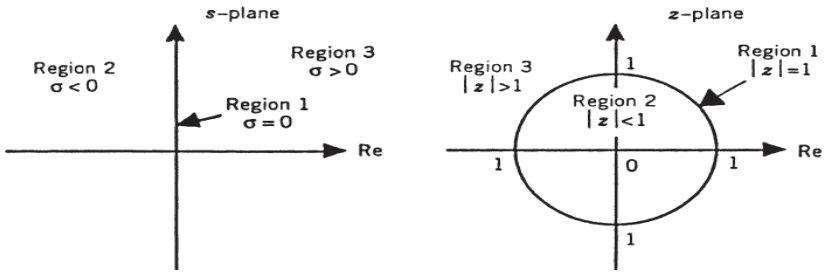
Benzer şekilde, bir sistemin kararlılığını z-dönüşümüne ilişkin z-düzleminde kutupların konumuna bakarak da belirleyebiliriz. Bunun için s-düzlemi ile z-düzlemi arasında birbirine karşılık gelen kutupları bulmamız gerekir. $z=e^{sT}$ ve s=σ+*jw* olduğundan,

$$z=e^{\sigma T} e^{jwT} \qquad (4.8)$$

O halde, z'in genliği $|z|= e^{\sigma T}$ fazı *Θ=wT=2πf \F$_s$*'dir. Burada F_s örnekleme frekansıdır. S-düzleminden z-düzlemine geçişi göstermek için Şekil 4.1'deki aşağıdaki bölgeleri ele alalım.

σ<0

s-düzleminde *jw* ekseninin sol tarafındaki kutuplar (2. Bölge) kararlı bir sistemi gösterir ve $e^{\sigma T}$<1 olduğundan (4.8)'den $|z| < 1$ olur. σ , -∞'dan 0'a değişirken $|z|$, 0'dan 1'e değişir. O halde, z-düzleminde ikinci bölge olarak gösterilen birim dairenin içindeki kutuplar karalı bir sistem oluştururlar. Böyle bir sistemin cevabı, kutuplar gerçel ise üstel olarak azalacaktır ya da kutuplar karmaşıksa sinüzoial olarak azalacaktır.



Şekil 4.1. s-düzleminden z-düzlemine geçiş.

σ>0

s-düzleminde *jw* ekseninin sağ tarafında bulunan kutuplar (3. Bölge) kararsız bir sistem olduğunu gösterir ve Eşitlik (4.8), $e^{\sigma T}>1$ olduğundan $|z| > 1$ şeklinde bir genlik ifade eder. σ , 0'dan ∞'a değişirken $|z|$, 1'den ∞′a değişir. O halde, üçüncü bölgede birim dairenin dışında kalan kutuplar kararsız bir sistem olduğunu gösterir. Böyle bir sistemin cevabı kutuplar gerçel ise üstel olarak artan olacaktır ya da kutuplar karmaşık ise simüzoidal olarak artan olacaktır.

σ=0

s-düzleminde *jw* ekseninin üzerinde bulunan kutuplar (1. Bölge) marjinal olarak kararlı bir sistem olduğunu gösterir ve Eşitlik (4.8), $|z| = 1$ genliğini ifade eder. O halde, z-düzleminde birinci bölgede birim daire üzerindeki kutuplar bir sinüzoidal verecektir. Bölüm 5'de birim daire üzerinde kutupları olan bir fark eşitliği ile programlanan sinüzoidal bir işaret gerçekleştirilecektir.

4.1.2. Fark Eşitlikleri

Analog filtreler diferansiyel eşitliklerle sayısal filtreler ise fark eşitlikleri ile ifade edilirler. Bir fark eşitliğini çözmek için, analog bir x(t) işaretinin k. türevine $d^kx(t)\backslash dt^k$ karşılık gelen, x(n-k) şeklindeki bir ifadenin z-dönüşümünü bulmamız gerekir. Fark eşitliğinin derecesi k'nın en büyük değeri ile belirlenir. Örneğin k=2 ikinci-dereceden türev anlamına gelir. Eşitlik (4.5)'den

$$X(z) = \sum_{n=0}^{\infty} x(n)z^{-n} = x(0) + x(1)z^{-1} + x(2)z^{-2} + \cdots \qquad (4.15)$$

elde edilir. dx\dt birinci türevine karşılık gelen x(n-1)'in z-dönüşümü

$$\begin{aligned} ZT[x(n-1)] &= \sum_{n=0}^{\infty} x(n-1)z^{-n} \\ &= x(-1) + x(0)z^{-1} + x(1)z^{-2} + x(2)z^{-3} + \cdots \\ &= x(-1) + z^{-1}\left[x(0) + x(1)z^{-1} + x(2)z^{-2} + \cdots\right] \\ &= x(-1) + z^{-1}X(z) \end{aligned} \qquad (4.16)$$

şeklinde hesaplanır. Burada Eşitlik (4.15) kullanılmıştır ve x(-1), birinci dereceden fark eşitliğine ilişkin başlangıç koşulunu göstermektedir. Benzer şekilde, x(n-2)'nin z-dönüşümü ikinci türeve eşdeğerdir ve

$$\begin{aligned} ZT[x(n-2)] &= \sum_{n=0}^{\infty} x(n-2)z^{-n} \\ &= x(-2) + x(-1)z^{-1} + x(0)z^{-2} + x(1)z^{-3} + \cdots \\ &= x(-2) + x(-1)z^{-1} + z^{-2}[x(0) + x(1)z^{-1} + \cdots] \\ &= x(-2) + x(-1)z^{-1} + z^{-2}X(z) \end{aligned} \qquad (4.17)$$

ile verilir. Burada x(-2) ve x(-1) ikinci dereceden bir fark eşitliğini çözmek için gereken iki başlangıç koşulunu göstermektedir. Genel olarak,

$$ZT[x(n-k)] = z^{-k}\sum_{m=1}^{k} x(-m)z^{m} + z^{k}X(z) \qquad (4.18)$$

elde edilir. Eğer tüm başlangıç koşulları sıfır kabul edilirse yani x(-m)=0, m=1,2,…,k. ise

$$ZT[x(n-k)] = z^{-k}X(z)$$

4.2.Ayrık İşaretler

Ayrık bir x(n) işareti

$$x(n) = \sum_{m=-\infty}^{\infty} x(m)\delta(n-m) \qquad (4.20)$$

şeklinde ifade edilir. Burada δ(n-m), n=m için 1 aksi durumlarda 0 değerini alan m gecikmeli δ(n) impuls dizisidir. Bu, n zamanı göstermek üzere, x(1), x(2)…. değerlerinde bir dizi içerir ve dizinin her bir örneğinin değeri örnekleme aralığı ya da örnekleme periyodu T=1/F ile belirlenen bir örnek zamanı kadar birbirinden ayrıktır.

Bu laboratuarda ilgileneceğimiz işaretler ve sistemler doğrusal ve zamanla değişmeyendir. Bir x(n) giriş işareti y(n) çıkışını versin yani x(n)→y(n). O halde, a_1 ve a_2 sabitler olmak üzere, $a_1x_1(n)\rightarrow a_1y_1(n)$ ve $a_2x_2(n)\rightarrow a_2y_2(n)$ iken $a_1x_1(n)+ a_2x_2(n)\rightarrow a_1y_1(n)+ a_2y_2(n)$ olmalıdır. Bu, tüm sistem çıkışının her bir girişe karşılık gelen ayrı ayrı çıkışların toplamı olduğunu ifade eden süperpozisyon özelliğidir. Zamandan bağımsızlık ise, eğer giriş m örnek gecikmişse çıkış da m örnek kadar gecikmelidir yani x(n-m)→y(n-m) anlamına gelir. Eğer giriş birim impuls δ(n) ise δ(n-m), zamandan bağımsızlık özelliği ile h(n-m) çıkış cevabını verir.

Ayrıca, eğer bu impuls x(m) ile çarpılırsa, x(m) δ(n-m)→x(m)h(n-m) elde edilir. Eşitlik (4.20) kullanılarak ,

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m) \qquad (4.21)$$

konvolüsyon eşitliği elde edilir. Nedensel bir sistem için, Eşitlik (4.21)

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m) \qquad (4.22)$$

k=n-m için,

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \qquad (4.23)$$

elde edilir.

4.3.FIR Filtreler

Filtreleme en önemli işaret işleme uygulamalarından biridir. Artık gerçek zamanda sayısal filtreleri gerçekleştirebileceğimiz DSP'ler mevcuttur. TMS320C6X komut kümesi ve mimarisi bu tür filtreleme uygulamaları için uygundur. Analog bir filtre sürekli-zaman işaretler üzerinde çalışır ve işlemsel kuvvetlendiriciler, dirençler ve kapasitörler gibi ayrık elemanlar ile gerçekleştirilir. Ancak, sayısal bir filtre ayrık-zaman işaretler üzerinde çalışır ve TMS320C6X gibi bir DSP ile gerçekleştirilebilir. Bu işlem, harici bir giriş işaretini almak için bir ADC kullanılarak ve elde edilen çıkış işaretini bir DAC'ye göndererek yapılır.

Son yıllarda, DSP'lerin maliyetleri oldukça düşmüştür. Bu sayede, analog filtre yerine birçok avantajından dolayı sayısal filtre kullanımı giderek artmaktadır. Daha yüksek dayanıklılık, doğruluk, sıcaklık ve kullanım süresine karşı daha düşük hassasiyet bu avantajlar arasında sayılabilir. Ayrıca, sayısal bir filtre ile daha keskin genlik ve faz karakteristikleri sağlamak mümkündür. Merkez frekansı, bant genişliği ve filtre türü gibi önemli filtre karakteristikleri kolayca değiştirilebilir. DSK tabanlı TMS320C6X kullanarak gerçek zamanda birkaç dakikada FIR bir filtre tasarlamak ve gerçekleştirmek için birçok araç mevcuttur. Filtre tasarımı transfer fonksiyonunun katsayılarını bulma yaklaşımına dayalıdır. FIR filtre tasarlamak için, Fourier serileri yaklaşımı ve bilgisayar-destekli yaklaşımlar gibi farklı yaklaşımlarda kullanılabilir.

FIR filtre tasarımı için, aşağıda verilen sonlu sayıda terim ile ifade edilebilen konvolüsyon eşitliği oldukça faydalı bir eşitliktir.

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \qquad (4.24)$$

Eğer giriş x(n)=$\delta(0)$ ile gösterilen birim dürtü fonksiyonu ise, çıkış dürtü yanıtı y(n)=h(n) olacaktır. Konvolüsyon eşitliğinden de anlaşılabileceği gibi, FIR bir filtre, n anındaki giriş işareti x(n) ve gecikmeli girişler x(n-k) biliniyorsa, kolayca gerçekleştirilebilir. Bu filtre yinelemeli değildir ve geri besleme ya da çıkışların önceki değerlerini gerektirmez. Çıkışın önceki değerlerini gerektiren geri beslemeli filtreler Bölüm 5'te tartışılacaktır.

Eşitlik (4.24)'ün başlangıç koşulları sıfırken z-dönüşümü

$$Y(z) = h(0)X(z) + h(1)z^{-1}X(z) + h(2)z^{-2}X(z) + \cdots + h(N-1)z^{-(N-1)}X(z) \quad (4.25)$$

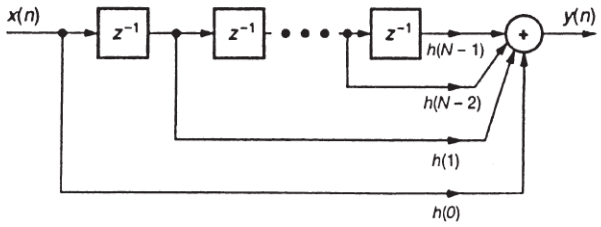
ile verilir. Eşitlik (4.24) katsayılar ve giriş işaretlerinin zamanda konvolüsyonudur. Konvolüsyon işlemi frekans ekseninde çarpmaya karşılık gelir.

$$Y(z)=H(z)\ X(z)$$

Burada H(z), h(n)'in z-dönüşümüdür ve filtrenin transfer fonksiyonu olarak adlandırılır.

$$\begin{aligned} H(z) &= \sum_{k=0}^{N-1} h(k)z^{-k} = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots + h(N-1)z^{-(N-1)} \\ &= \frac{h(0)z^{(N-1)} + h(1)z^{N-2} + h(2)z^{N-3} + \cdots + h(N-1)}{z^{N-1}} \end{aligned} \qquad (4.27)$$

Eşitlikten de görüldüğü gibi transfer fonksiyonu tamamı orijinde bulunan N-1 kutup içerir. O halde, FIR filtreler kararlı yapılardır çünkü kutupları birim dairenin içerisindedir. FIR bir filtre genellikle "kutupsuz" filtre olarak tanımlanır. Şekil 4.2 , (4.24) ve (4.25) eşitliklerini gösteren FIR bir filtre yapısını göstermektedir.



Şekil 4. 2. Gecikmeleri gösteren FIR filtre yapısı.

FIR filtrenin en kullanışlı özelliklerinden biri de doğrusal fazı garantilemesidir. *Doğrusal faz özelliği* faz bozunumlarının önemli olduğu ses analizi gibi uygulamalarda oldukça faydalıdır. Örneğin, doğrusal fazla, sinüzoidal giriş bileşenlerinin tamamı aynı miktarda geciktirilir. Aksi takdirde, harmonik bozunum oluşur. Doğrusal fazlı filtreler FIR filtrelerdir ancak, tüm FIR filtreler doğrusal fazlı değildir.

Gecikmeli bir giriş örneği x(n-k)'nın Fourier dönüşümü, fazı Θ=*-wkT* yani *w*'nun doğrusal bir fonksiyonu olan $e^{-jwkT}X(jw)$'dur. Fazın türevi olarak tanımlanan grup gecikme fonksiyonunun bir sabit olduğuna ya da $d\Theta/dw$=*-kT* olduğuna dikkat edilmelidir.

4.4.FIR Kafes Yapı

Kafes yapı uyarlamalı filtre ve ses işleme uygulamalarında yaygın olarak kullanılır. N. dereceden bir kafes yapı Şekil 4.3'de gösterilmiştir. k_1, k_2,…,k_N katsayıları yansıma katsayıları (ya da k-parametreleri) olarak adlandırılır. Bu yapının bir avantajı frekans cevabının katsayılardaki küçük değişimlere karşı önceki yapı gibi hassas olmamasıdır. Şekil 4.3'deki N=1 olan ilk kısımdan,

$$y_1(n) = x(n) + k_1 x(n-1) \tag{4.28}$$

$$e_1(n) = k_1 x(n) + x(n-1) \tag{4.29}$$

elde edilir. Birinci kısma kaskad bağlı ikinci kısımdan ise

$$\begin{aligned} y_2(n) &= y_1(n) + k_2 e_1(n-1) \\ &= x(n) + k_1 x(n-1) + k_2 k_1 x(n-1) + k_2 x(n-2) \\ &= x(n) + (k_1 + k_1 k_2) x(n-1) + k_2 x(n-2) \end{aligned} \tag{4.30}$$

ve

$$\begin{aligned} e_2(n) &= k_2 y_1(n) + e_1(n-1) \\ &= k_2 x(n) + k_2 k_1 x(n-1) + k_1 x(n-1) + x(n-2) \\ &= k_2 x(n) + (k_1 + k_1 k_2) x(n-1) + x(n-2) \end{aligned} \tag{4.31}$$

elde edilir. Örneğin, i. kısım için

$$y_i(n) = y_{i-1}(n) + k_i e_{i-1}(n-1) \tag{4.32}$$

$$e_i(n) = k_i y_{i-1}(n) + e_{i-1}(n-1) \tag{4.33}$$

yazılabilir. (4.30) ve (4.31) eşitliklerinin tersine çevrilmiş şekilde aynı katsayılara sahip olduğunu örmek bizim için yol gösterici olacaktır. Bu özelliğin daha yüksek dereceli yapılar için de doğru olduğu gösterilebilir. Genel olarak, N. dereceden FIR bir kafes yapı için (4.30) ve (4.31) eşitlikleri

$$y_N(n) = \sum_{i=0}^{N} a_i x(n-i) \tag{4.34}$$

ve

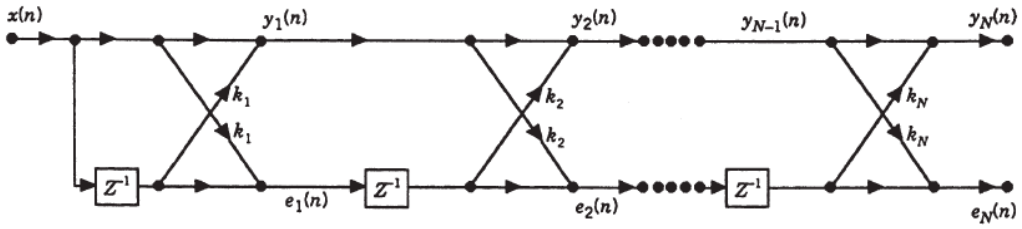
$$e_N(n) = \sum_{i=0}^{N} a_{N-i} x(n-i) \tag{4.35}$$

a_0=1 için şeklinde yeniden düzenlenebilir. (4.34) ve (4.35) eşitliklerinin z-dönüşümü hesaplanır ve dürtü yanıtları bulunursa,

$$Y_N(z) = \sum_{i=0}^{N} a_i z^{-i} \tag{4.36}$$

$$E_N(z) = \sum_{i=0}^{N} a_{N-i} z^{-i} \tag{4.37}$$

elde edilir.



Şekil 4.3. FIR Kafes Yapı

$$E_N(z) = z^{-N} Y_N(1/z) \qquad (4.38)$$

olduğuna dikkat edilmelidir. (4.36) ve (4.37) eşitlikleri görüntü polinomları olarak adlandırılır. İki kısım için, $k_2=a_2$'dir genel olarak ise

$$k_N = a_N \qquad (4.39)$$

Bu yapının faydalı olabilmesi için, k-parametreleri ile impuls cevabı katsayıları arasındaki ilişkinin bulunması gerekir. (4.39) eşitliğindeki k_N parametresi ile başlayarak, yinelemeli olarak(geriye doğru yineleme ile) önceki k-parametreleri k_{N-1}, …, k_1 hesaplanabilir.

r. ara kesiti ele alalım. (4.36) ve (4.37) eşitlikleri kullanılarak,

$$Y_r(z) = Y_{r-1}(z) + k_r z^{-1} E_{r-1}(z) \qquad (4.40)$$

$$E_r(z) = k_r Y_{r-1}(z) + z^{-1} E_{r-1}(z) \qquad (4.41)$$

yazılabilir. Eşitlik (4.41)'den E_{r-1}'i çekip Eşitlik (4.40)'da yerine yazarak $Y_r(z)$ şu şekilde elde edilir

$$Y_r(z) = Y_{r-1}(z) + k_r z^{-1} \frac{E_r(z) - k_r Y_{r-1}(z)}{z^{-1}} \qquad (4.42)$$

Eşitlik (4.42)'den $Y_{r-1}(z)$, $Y_r(z)$'e bağlı olarak çekilebilir

$$Y_{r-1}(z) = \frac{Y_r(z) - k_r E_r(z)}{1 - k_r^2}, \quad |k_r| = 1 \qquad (4.43)$$

N=r için Eşitlik (4.38) kullanılarak Eşitlik (4.48) yeniden düzenlenirse

$$Y_{r-1}(z) = \frac{Y_r(z) - k_r z^{-r} Y_r(1/z)}{1 - k_r^2} \qquad (4.44)$$

elde edilir. Eşitlik (4.44), geriye doğru bir yineleme işlemi kullanılarak $1 \le r \le N$ için Y_r'den Y_{r-1}'in elde edilebileceğini gösteren önemli bir ilişkidir. Sonuç olarak, k_r'den başlayarak önceki k-parametrelerine de ulaşabileceğimizi söyleyebiliriz. r kesiti için, Eşitlik (4.36)

$$Y_r(z) = \sum_{i=0}^{r} a_{ri} z^{-i} \qquad (4.45)$$

şeklinde yazılabilir. *i* yerine *r-i* ve *z* yerine *1/z* yazılarak

$$Y_r\left(\frac{1}{z}\right) = \sum_{i=0}^{r} a_{r(r-i)} z^{r-i} \qquad (4.46)$$

elde edilir. (4.45) ve (4.46) eşitlikleri kullanılarak Eşitlik (4.44) yeniden düzenlenirse,

$$\sum_{i=0}^{r} a_{(r-1)i} z^{-i} = \frac{\sum_{i=0}^{r} a_{ri} z^{-i} - k_r z^{-r} \sum_{i=0}^{r} a_{r(r-i)} z^{r-i}}{1-k_r^2} \qquad (4.47)$$

$$= \frac{\sum_{i=0}^{r} a_{ri} z^{-i} - k_r \sum_{i=0}^{r} a_{r(r-i)} z^{-i}}{1-k_r^2} \qquad (4.48)$$

ve *r*=*N*, *N*-1, …,1, $|k_r| \neq 1$, l=0,1,…,r-1, $k_r=a_{rr}$, için

$$a_{(r-1)i} = \frac{a_{ri} - k_r a_{r(r-i)}}{1-k_r^2}, \quad i = 0, 1, \ldots, r-1 \qquad (4.49)$$

yazılabilir.

Örnek 4.3. FIR Kafes Yapı

Bu örnekte k-parametrelerini hesaplamak için (4.49) ve (4.50) eşitliklerinin kullanımı gösterilmiştir. FIR filtrenin frekans uzayındaki impuls cevabı aşağıda verilmiştir.

$$Y_2(z) = 1 + 0.2z^{-1} - 0.5z^{-2}$$

Eşitlik (4.45)'den r=2 için

$$Y_2(z) = a_{20} + a_{21}z^{-1} + a_{22}z^{-2}$$

elde edilir. burada a_{20}=1, a_{21}=0.2 ve a_{22}=-0.5 'dir. Eşitlik (4.50)'de r=2 ile başlayarak

$$k_2 = a_{22} = -0.5$$

elde edilir. Eşitlik (4.49) kullanılarak, i=0 için

$$a_{10} = \frac{a_{20} - k_2 a_{22}}{1-k_2^2} = \frac{1-(-0.5)(-0.5)}{1-(-0.5)^2} = 1$$

ve i=1 için

$$a_{11} = \frac{a_{21} - k_2 a_{21}}{1-k_2^2} = \frac{0.2-(-0.5)(0.2)}{1-(-0.5)^2} = 0.4$$

Eşitlik (4.50)'den

$$k_1 = a_{11} = 0.4$$

elde edilir. k- parametrelerinin değerinin Eşitlik (4.30) kullanılarak da doğrulanabileceği unutulmamalıdır.

4.5.Fourier Serisi Kullanılarak FIR Filtre Gerçekleştirme

Fourier serisi yöntemi kullanılarak FIR filtre tasarımı transfer fonksiyonunun genlik cevabının istenilen bir genlik cevabına yaklaştığı bir yöntemdir. İstenilen transfer fonksiyonu,

$$H_d(\omega) = \sum_{n=-\infty}^{\infty} C_n e^{jn\omega T} \qquad |n| < \infty \tag{4.51}$$

Burada C_n'ler Fourier serisi katsayılarıdır. F_N Nyquist frekansı yani $F_N=F_s/2$ olmak üzere, normalleştirilmiş frekans değişkeni $v=f/F_N$ kullanılarak Eşitlik (4.51)'deki transfer fonksiyonu

$$H_d(v) = \sum_{n=-\infty}^{\infty} C_n e^{jn\pi v} \tag{4.52}$$

şeklinde yazılabilir. Burada $wT=2\pi f/F_s=\pi v$, $|v|<1$. C_n katsayıları şu şekilde tanımlanır

$$\begin{aligned} C_n &= \tfrac{1}{2}\int_{-1}^{1} H_d(v) e^{-jn\pi v} dv \\ &= \tfrac{1}{2}\int_{-1}^{1} H_d(v)(\cos n\pi v - j\sin n\pi v) dv \end{aligned} \tag{4.53}$$

$H_d(v)$'nin çift fonksiyon olduğunu varsayarsak Eşitlik (4.53)

$$C_n = \int_0^1 H_d(v)\cos n\pi v \, dv \qquad n \geqq 0 \tag{4.54}$$

şeklinde yeniden düzenlenebilir. $H_d(v)\sin(n\pi v)$ tek fonksiyon olduğundan ve

$$\int_{-1}^{1} H_d(v)\sin n\pi v \, dv = 0$$

$C_n = C_{-n}$. Eşitlik (4.52)'deki istenilen transfer fonksiyonu $H_d(v)$, sonsuz sayıda katsayı ile ifade edilir ve gerçekleştirilebilir bir filtre elde etmek için Eşitlik (4.52) kesilir. Böylece yaklaşık transfer fonksiyonunu elde edilir.

$$H_a(v) = \sum_{n=-Q}^{Q} C_n e^{jn\pi v} \tag{4.55}$$

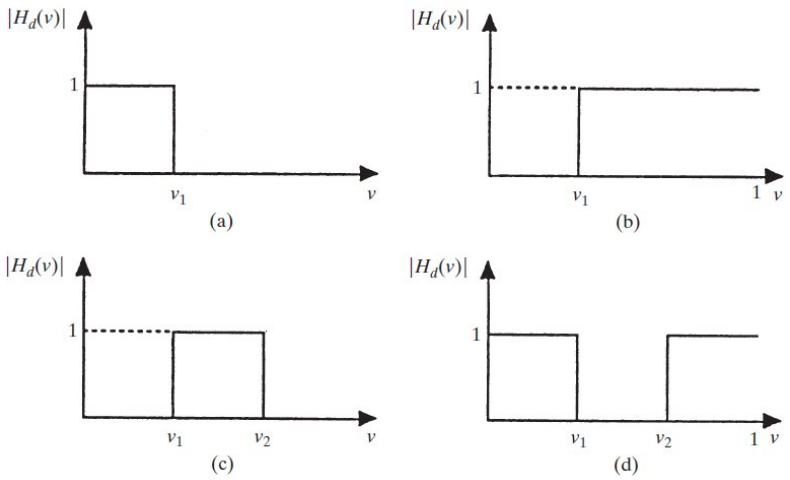
Burada Q filtrenin derecesini belirler, pozitif ve sonludur. Q'nın daha büyük olması FIR filtrenin derecesinin daha büyük olması demektir ve Eşitlik (4.55)'in istenilen transfer fonksiyonuna daha fazla yaklaşmasını sağlar. Sonsuz serilerin sonlu sayıda elemana sahip derilere dönüştürülmesi –Q - +Q arasındaki dikdörtgen pencere fonksiyonunun dışında kalan terimlerin katkısını göz ardı etmek demektir. Filtrelerin karakteristikleri dikdörtgen pencere dışında pencereler kullanılarak iyileştirilebilir.

$z=e^{j\pi v}$ olsun. O halde, Eşitlik (4.55)

$$H_a(z) = \sum_{n=-Q}^{Q} C_n z^n \tag{4.56}$$

olur. Bu durumda impuls cevabı katsayıları C_{-Q}, C_{-Q+1},…, C_{-1}, C_0, …, C_{Q-1}, C_Q'dır. Eşitlik (4.56)'daki z'in pozitif üslerini içeren yaklaşık transfer fonksiyonu giriş uygulanmadan çıkış üreten nedensel olmayan ve gerçekleştirilemez bir filtredir. Bu durumu düzeltmek için, Eşitlik (4.56)'da Q örnekleri için bir gecikme tanımlanır

$$H(z) = z^{-Q} H_a(z) = \sum_{n=-Q}^{Q} C_n z^{n-Q} \tag{4.57}$$



Şekil 4.4. İstenilen transfer fonksiyonu. a) alçak geçiren, b) yüksek geçiren, c) bant geçiren, d) bant durduran

n-Q=-i olsun. Bu durumda, Eşitlik (4.57)'deki H(z)

$$H(z) = \sum_{i=0}^{2Q} C_{Q-i} z^{-i} \qquad (4.58)$$

olacaktır. $h_i=C_{Q-i}$ ve N-1=2Q olsun H(z),

$$H(z) = \sum_{i=0}^{N-1} h_i z^{-i} \qquad (4.59)$$

Burada H(z), impuls cevabı katsayılarına bağlı olarak ifade edilmiştir ve $h_0=C_Q$, $h_1=C_{Q-1},\ldots,h_Q=C_0,h_{Q+1}=C_{-1},\ldots,h_{2Q}=C_{-Q}$'dır. İmpuls cevabı katsayıları h_Q civarında simetrik ve $C_n=C_{-n}$'dir.

Filtrenin derecesi N=2Q+1'dir. Örneğin, Q=5 ise, filtrenin 11 katsayısı olacaktır yani $h_0,h_1,\ldots,h_{10}$ ya da

$$h_0 = h_{10} = C_5$$
$$h_1 = h_9 = C_4$$
$$h_2 = h_8 = C_3$$
$$h_3 = h_7 = C_2$$
$$h_4 = h_6 = C_1$$
$$h_5 = C_0$$

Şekil 4.4. katsayıları $C_n=C_{-n}$ durumunu sağlayan ve frekans seçici filtreler olan alçak geçiren, yüksek geçiren, bant geçiren ve bant durduran filtreler için istenilen transfer fonksiyonu $H_d(v)$'yi ideal olarak göstermektedir.

1. *Alçak geçiren*:$C_0=v_1$

$$C_n = \int_0^{v_1} H_d(v)\cos n\pi v \, dv = \frac{\sin n\pi v_1}{n\pi} \qquad (4.60)$$

2. *Yüksek geçiren*: $C_0=1-v_1$

$$C_n = \sum_{v_1}^{1} H_d(v)\cos n\pi v\, dv = \frac{\sin n\pi v_1}{n\pi} \qquad (4.61)$$

3. *Bant geçiren*: $C_0=v_2-v_1$

$$C_n = \int_{v_1}^{v_2} H_d(v)\cos n\pi v\, dv = \frac{\sin n\pi v_2 - \sin n\pi v_1}{n\pi} \qquad (4.62)$$

4. *Bant durduran*: $C_0=1-(v_2-v_1)$

$$C_n = \int_0^{v_1} H_d(v)\cos n\pi v\, dv + \int_{v_2}^{1} H_d(v)\cos n\pi v\, dv = \frac{\sin n\pi v_1 - \sin n\pi v_2}{n\pi} \qquad (4.63)$$

Burada v_2 ve v_1 Şekil 4.4'de gösterilen normalleştirilmiş kesim frekanslarıdır.

FIR filtre tasarlamak için çeşitli filtre tasarım paketleri mevcuttur. Bir FIR fitler gerçekleştirdiğimizde, filtrenin tipini(örneğin alçak geçiren ya da band geçiren olup olmadığını) belirleyecek özel katsayıları veren bir program geliştirmiş oluruz.

Örnek 4.4. *Alçak geçiren FIR filtre*

N=11, örnekleme frekansı 10kHz ve kesim frekansı f_c=1kHz olan bir Fır filtrenin impuls cevabı katsayıları bulalım. Eşitlik (4.60)'dan

$$C_0 = v_1 = \frac{f_c}{F_N} = 0.2$$

Burada $F_N=F_s/2$ değerindeki Nyquist frekansıdır ve

$$C_n = \frac{\sin 0.2n\pi}{n\pi} \qquad n = \pm 1, \pm 2, \ldots, \pm 5 \qquad (4.64)$$

Impuls cevabı katsayıları $h_i=C_{Q-i}$, $C_n=C_{-n}$ ve Q=5 olduğundan,

$$\begin{aligned} h_0 &= h_{10} = 0 & h_3 &= h_7 = 0.1514 \\ h_1 &= h_9 = 0.0468 & h_4 &= h_6 = 0.1872 \\ h_2 &= h_8 = 0.1009 & h_5 &= 0.2 \end{aligned} \qquad (4.65)$$

elde edilir. Bu katsayıların Q=5 civarında simetrik olduğu unutulmamalıdır. FIR bir filtrede pratik uygulamalar için N=11 düşük iken, bu sayıyı ikiye katlamak daha iyi karakteristiklere (seçicilik özelliği gibi) sahip bir FIR filtre elde etmemizi sağlar. FIR bir filtrede doğrusal faza sahip olabilmek için katsayıları Eşitlik (4.65)'deki gibi simetrik olması gerekir.

4.6.FIR Filtreler için Programlama Örnekleri

Aşağıdaki örnekler FIR filtrelerin gerçekleştirilmelerini göstermektedir. Programların çoğu C dilinde yazılmıştır. FIR bir filtrenin genlik frekans cevabını göstermek için farklı yöntemler gösterilmiştir.

Örnek 1. Kayan Ortalamalı Filtre (average)

Kayan ortalamalı filtreler DSP'de yaygın olarak kullanılır ve tüm sayısal filtreler arasında anlaşılması en kolay olanıdır. Bu filtre özellikle bir işaretten gürültüyü (yüksek frekanslı) temizlemek için kullanılır. Kayan ortalamalı filtre, her bir çıkış örneğini üretmek için girişin belli sayıda geçmiş örneğinin aritmetik ortalamasını alarak çalışır. Bu işlem aşağıdaki eşitlik ile gösterilebilir

$$y(n) = \frac{1}{N} \sum_{i=0}^{N-1} x(n-i)$$

Average dosyasındaki average.c programını inceleyiniz. Dosyayı yükleyip çalıştırınız. Beş noktalı kayan ortalamalı filtrenin karakteristikleri ile gerçekleştirilebilecek birçok yöntem gösterilebilir. Average dosyasında bulunan mefsin.wav test dosyası sinüzoidal bir ton eklenerek bozulmuş bir ses kaydıdır. Bu dosyayı Goldwave, Media Player gibi bir program kullanarak dinleyiniz. Daha sonra bilgisayarın ses kartı çıkışını DSK'nin LINE IN girişine bağlayarak filtrelenmiş test işaretini dinleyiniz (LINE OUT veya HEADPHONE). Sinüzoidal tonun kaybolduğu ve sesin boğuklaştığı anlaşılmalıdır. Bu gözlemler filtrenin alçak geçiren frekans cevabına sahip olduğunu doğrular.

```
//average.c

#include "DSK6713_AIC23.h"            //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;    //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select input

#define N 5                           //no of points averaged
float x[N];                           //filter input delay line
float h[N];                           //filter coefficients

interrupt void c_int11()              //interrupt service routine
{
  short i;
  float yn = 0.0;



  x[0]=(float)(input_left_sample()); //get new input sample
  for (i=0 ; i<N ; i++)               //calculate filter output
    yn += h[i]*x[i];
  for (i=(N-1) ; i>0 ; i--)           //shift delay line contents
    x[i] = x[i-1];
  output_left_sample((short)(yn));    //output to codec
  return;
}

void main()
{
  short i;                            //index variable

  for (i=0 ; i<N ; i++)               //initialise coefficients
    h[i] = 1.0/N;
  comm_intr();                        //initialise DSK
  while(1);                           //infinite loop
}
```

Şekil 4.1. Beş noktalı kayan ortalamalı filtre programı (average.c).

Örnek 2. Kayan Ortalamalı, Band Durduran ve Band Geçiren FIR Filtre Karakteristikleri

Şekil 4.2'deki fir.c programında kullanılan yöntem, average.c programı ile yapılan işleme benzer olarak, herbir çıkış örneğini hesaplamaktır. c_int11( ) fonksiyonu her iki programda da aynı tanımlamaya sahiptir. Average.c programı main ( ) fonksiyonundaki katsayı değerlerini hesaplarken fir.c programı ayrı bir dosyadan katsayı değrlerini okumaktadır.

Beş Noktalı Kayan Ortalama (ave5f.cof)

Şekil 4.3'de ave5f.cof dosyası verilmektedir. Bu dosyayı kullanarak, fir.c programı average.c programı ile gerçekleştirilen beş noktalı kayan ortalamalı filtreyi aynı şekilde gerçekleştirmektedir. Filtre katsayılarının sayısı .cof dosyasında tanımlanan N sabitinin değeri ile belirlenir. fir.c projesini derleyiniz ve çalıştırınız.

2700 Hz Merkez Frekanslı Band Durduran (bs2700.cof)

fir.c dosyasındaki

#include "ave5f.cof"

satırını

#include "bs2700f.cof"

ile değiştiriniz. fir. projesini yükleyip çalıştırınız. Sinüzoidal bir giriş işareti uygulayınız ve işaretin frekansını 2700'ün biraz altında ve üzerinde olacak şekilde değiştiriniz. Çıkışın 2700 Hz'de minimum olduğunu gözlemleyiniz. Bu filtrenin katsayıları, Şekil 4.4'de gösterilen, MATLAB'ın fdatool filtre tasarım ve analiz aracı kullanılarak hesaplanmıştır.

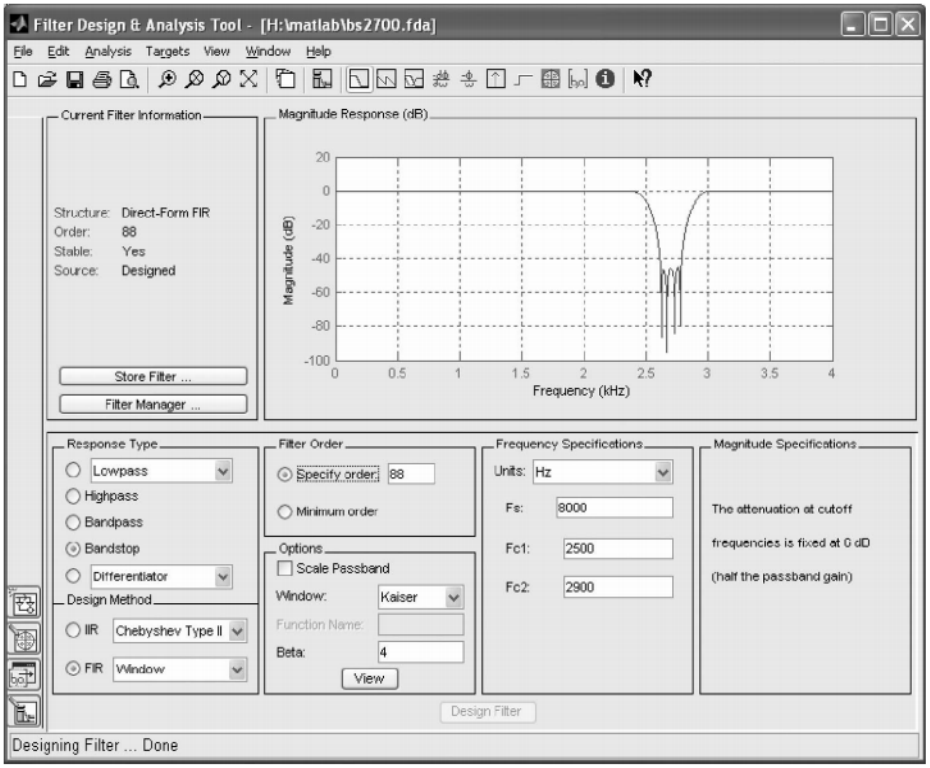
```
//fir.c

#include "DSK6713_AIC23.h"           //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;   //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select line in

#include "ave5f.cof"                 //filter coefficient file
float x[N];                          //filter delay line

interrupt void c_int11()             //interrupt service routine
{
  short i;
  float yn = 0.0;
```

```
interrupt void c_int11()              //interrupt service routine
{
  short i;
  float yn = 0.0;

  x[0]=(float)(input_left_sample()); //get new input sample
  for (i=0 ; i<N ; i++)              //calculate filter output
    yn += h[i]*x[i];
  for (i=(N-1) ; i>0 ; i--)          //shift delay line contents
    x[i] = x[i-1];
  output_left_sample((short)(yn));   //output to codec
  return;
}

void main()
{
  comm_intr();                       //initialise DSK
  while(1);                          //infinite loop
}
```

Şekil 4.2 FIR filtre programı (fir.c).

```
// ave5f.cof
// this file was generated automatically using function dsk_fir67.m

#define N 5

float h[N] = {
2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001
};
```

Şekil 4.3. Katsayı dosyası (ave5f.cof).



Şekil 4.4. 2700Hz frekanslı band durduran filtre için MATLAB fdatool penceresi.

1750 Hz Merkez Frekanslı Band Geçiren (bp1750f.cof)

fir.c programını bp1750f.cof dosyasını içerecek şekilde değiştiriniz. bp1750.cof dosyası 1750 merkez frekanslı FIR bant geçiren bir filtrenin katsayılarını (81 katsayı) göstermektedir. Filtre yine fdatool kullanılarak tasarlanmıştır (Şekil 4.5). Projeyi derleyip çalıştırınız.

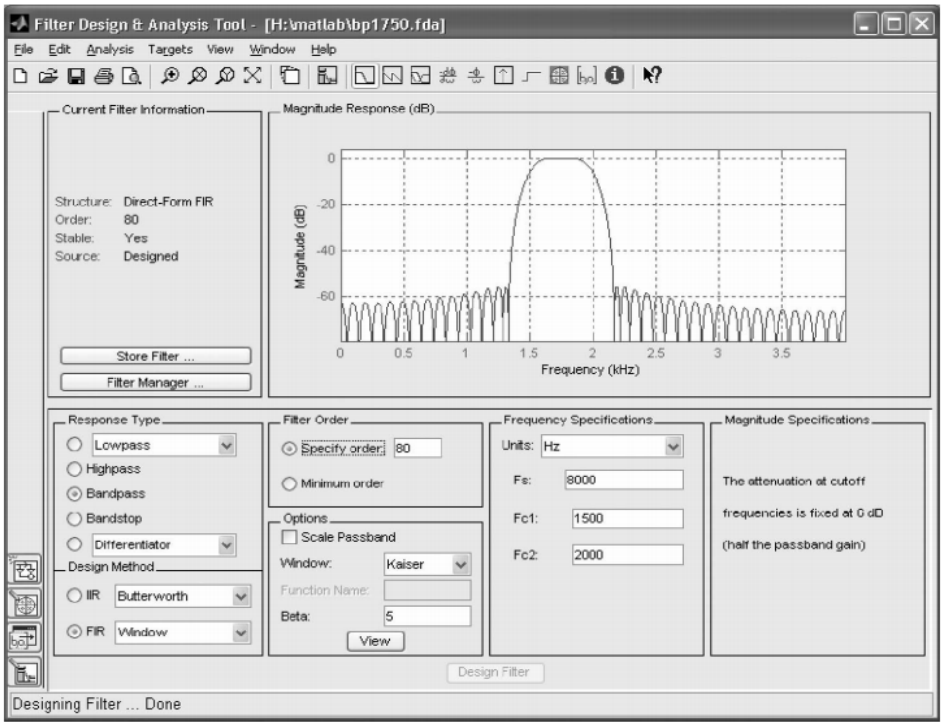
MATLAB Kullanarak Filtre Katsayılarının Üretilmesi

Filtre katsayılarının sayısı azsa .cof uzantılı bir katsayı dosyası kullanılabilir. Katsayıların fazla olması durumunda dsk_fir67 ( ) fonksiyonu ve dsk_fir67.m MATLAB dosyası kullanılabilir. Şekil 4.6'da verilen bu fonksiyon bir MATLAB katsayı vektörü gelmesini bekler ve kullanıcıdan çıkış dosya adı ister. Örneğin ave5f.cof katsayı dosyası aşağıdaki MATLAB komutları ile oluşturulabilir:

```
>> x = [0.2, 0.2, 0.2, 0.2, 0.2];
>> dsk_fir67(x)
enter filename for coefficients ave5f.cof
```

Katsayı dosyasının adı .cof uzantısını da içerecek şekilde tam olarak girilmelidir.

FIR filtre katsayılarını hesaplamak için MATLAB'ın fdatool aracı kullanılıp MATLAB'ın workspace'ine transfer edilebilir. Daha sonra dsk_fir67 ( ) fonksiyonu fir.c programına uygun bir katsayı dosyası oluşturmak için kullanılır. dsk_fir67 ( ) fonksiyonu ile geçirilen filtre katsayılarının toplamları bir olacak şekilde normalleştirilmesi tavsiye edilir.



Şekil 4.5. 1750Hz frekanslı bant geçiren filtre için MATLAB fdatool penceresi.

```
% DSK_FIR67.M
% MATLAB function to write FIR filter coefficients
% in format suitable for use in C6713 DSK programs
% firnc.c and firprn.c
% written by Donald Reay
%
function dsk_fir67(coeff)
%
coefflen=length(coeff);
fname = input('enter filename for coefficients ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
fprintf(fid,'// this file was generated automatically using function
dsk_fir67.m\n',fname);
fprintf(fid,'\n#define N %d\n',coefflen);
fprintf(fid,'\nfloat h[N] = { \n');
% j is used to count coefficients written to current line
% in output file
j=0;
% i is used to count through coefficients
for i=1:coefflen
% if six coeffs have been written to current line
% then start new line
  if j>5
    j=0;
    fprintf(fid,'\n');
  end

% if this is the last coefficient then simply write
% its value to the current line
% else write coefficient value, followed by comma
  if i==coefflen
   fprintf(fid,'%2.4E',coeff(i));
  else
    fprintf(fid,'%2.4E,',coeff(i));
    j=j+1;
  end
end
fprintf(fid,'\n};\n');
fclose(fid);
```

Şekil 4.6. dsk_fir67.m Matlab fonksiyonu.

Örnek 3:Filtre Girişi Olarak Sözde-Rastgele Gürültü Dizisi Kullanan FIR Gerçekleştirme

Şekil 4.7'de verilen firprn.c programı, filtrenin girişine dahili olarak üretilen sözde rastgele gürültü dizisi uygulayarak FIR bir filtre gerçekleştirmektedir. Diğer tüm yönleri ile fir.c programına benzerdir. Başlangıç olarak bs200f.cof katsayı dosyası kullanılmıştır.

Projeyi derleyip çalıştırınız. Çıkış işaretinin, 2700Hz merkez frekanslı bant geçiren FIR bir filtre ile filtrelenmiş gürültü olduğunu doğrulayınız. Osiloskobunuzun FFT fonksiyonu ile çıkış işaretini gözlemleyiniz.

Farklı FIR Filtrelerin Test Edilmesi

Programı durdurun. Farklı FIR filtreleri gösteren katsayı dosyalarını test etmek için firprn.c kaynak dosyasını açınız. Her bir katsayı dosyası 55 katsayıdan oluşmaktadır (comb14f.cof hariç).

1. bp55f.cof: $F_s/4$ merkez frekanslı bant geçiren
2. bs55f.cof: $F_s/4$ merkez frekanslı bant durduran
3. lp55f.cof: $F_s/4$ kesim frekanslı alçak geçiren
4. hp55f.cof: $F_s/4$ bant genişliğine sahip yüksek geçiren
5. pass2bf.cof: iki geçirme bantlı
6. pass3bf.cof: üç geçirme bantlı
7. pass4bf.cof: dört geçirme bantlı
8. comb14f.cof: çok çentikli (tarak filtre)

Bu filtreler MATLAB kullanılarak tasarlanmıştır. FIR filtrelerin çıkışları FFT kullanılarak gözlemlenebilir.

Örnek 4: Bozulmuş Ses Kaydını Geri Elde Etmek için İki Çentik Filtre Kullanımı

Bu örnekte 900 ve 2700Hz frekanslarında iki sinüzoidal işaret eklenerek bozulmuş bir ses kaydının tekrar geri elde edilmesi için iki çentik filtre kullanımı açıklanacaktır. notch2.c programı Şekil 4.8'de verilmiştir. Matlab kullanılarak tasarlanmış her biri 89 katsayıdan oluşan bs900.cof ve bs2700.cof katsayı dosyaları program tarafından kullanılmaktadır. Bu dosyalar sırasıyla 900 ve 2700Hz frekanslı çantik filtreleri gerçekleştirmektedir.

Projeyi derleyip çalıştırınız. notch.2 dosyasında bulunan corrrupt.wav dosyasını dinleyiniz. Bilgisayarın ses kartı çıkışını DSK'nın LINE IN girişine bağlayarak çıkışı dinleyiniz. Örnek 1'de elde ettiğiniz sonuçla burada elde ettiğiniz sonucu karşılaştırınız.

```
//firprn.c FIR with internally generated input noise sequence

#include "DSK6713_AIC23.h"           //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;   //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select line in

#include "bs2700f.cof"               //filter coefficient file
#include "noise_gen.h"               //support file for noise
int fb;                              //feedback variable
shift_reg sreg;                      //shift register
#define NOISELEVEL 8000              //scale factor for noise
float x[N];                          //filter delay line

int prand(void)                      //pseudo-random noise
{
  int prnseq;
  if(sreg.bt.b0)
    prnseq = -NOISELEVEL;            //scaled -ve noise level
  else
    prnseq = NOISELEVEL;             //scaled +ve noise level
  fb =(sreg.bt.b0)^(sreg.bt.b1);     //XOR bits 0,1
  fb^=(sreg.bt.b11)^(sreg.bt.b13);   //with bits 11,13 -> fb
  sreg.regval<<=1;                   //shift register 1 bit left
  sreg.bt.b0=fb;                     //close feedback path
  return prnseq;
}

void resetreg(void)                  //reset shift register
{
  sreg.regval=0xFFFF;                //initial seed value
  fb = 1;                            //initial feedback value
}
```

```
interrupt void c_int11()            //interrupt service routine
{
  short i;                          //declare index variable
  float yn = 0.0;

  x[0] = (float)(prand());          //get new input sample
  for (i=0 ; i<N ; i++)             //calculate filter output
    yn += h[i]*x[i];
  for (i=(N-1) ; i>0 ; i--)         //shift delay line contents
    x[i] = x[i-1];
  output_left_sample((short)(yn));  //output to codec
  return;                           //return from interrupt
}

void main()
{
  resetreg();                       //reset shift register
  comm_intr();                      //initialise DSK
  while (1);                        //infinite loop
}
```

Şekil 4.7. Girişi dahili olarak üretilen sözde gürültü dizisi olan FIR filtre programı.

```
//notch2.c Two FIR notch filters to remove sinusoidal noise

#include "DSK6713_AIC23.h"          //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;  //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select line in
#include "bs900.cof"                //BS 900 Hz coefficient file
#include "bs2700.cof"               //BS 2700 Hz coefficient file
short dly1[N]={0};                  //delay for 1st filter
short dly2[N]={0};                  //delay for 2nd filter
int y1out = 0, y2out = 0;           //init output of each filter
short out_type = 2;                 //slider for output type

interrupt void c_int11()            //ISR
{
 short i;

 dly1[0] = input_left_sample();     //new input @ top of buffer
 y1out = 0;                         //init output of 1st filter
 y2out = 0;                         //init output of 2nd filter
 for (i = 0; i< N; i++)
  y1out += h900[i]*dly1[i];         //y1(n)+=h900(i)*x(n-i)
 dly2[0]=(y1out>>15);
 for (i = 0; i< N; i++)
  y2out += h2700[i]*dly2[i];        //y2(n)+=h2700(i)*x(n-i)
 for (i = N-1; i > 0; i--)          //from bottom of buffer
 {
  dly1[i] = dly1[i-1];              //update samples in buffers
  dly2[i] = dly2[i-1];
 }

 if (out_type==1)                   //if slider is in position 1
  output_left_sample((short)(y2out>>15)); //output of 1st filter
 if (out_type==2)
  output_left_sample((short)(y1out>>15)); //output of 2nd filter
 return;                            //return from ISR
}

void main()
{
 comm_intr();                       //init DSK, codec, McBSP
 while(1)                           //infinite loop
}
```

Şekil 4. 8. İstenmeyen iki sinüzoidal işareti yok eden iki çentik filtresini gerçekleştiren program (notch2.c)

SAYISAL İŞARET İŞLEME LABORATUARI

LAB 5: SONSUZ DÜRTÜ YANITLI (IIR) FİLTRELER

Bu bölümde aşağıdaki başlıklar ele alınacaktır.

- Sonsuz dürtü yanıtlı filtre yapıları: Direkt Şekil-1, Direkt Şekil-II, Kaskad ve Parelel.
- Filtre tasarımı için çift doğrusal dönüşüm.
- Fark eşitlikleri kullanarak sinüzoidal dalga şekli üretimi.
- Filtre tasarımı ve faydalı paketler.
- TMS320C6X and C kodları kullanarak oluşturulan programlama örnekleri.

Bölüm 4'de tartışılan FIR filtrelerin analog karşılıkları yoktur. Bu kısımda tartışacağımız IIR filtreler analog filtreler ile de gerçekleştirilebilirler. Tasarım, çift doğrusal dönüşüm tekniği (BLT) kullanarak analog filtreyi sayısal eşdeğerine dönüştürerek gerçekleştirilir. BLT yöntemi aslında, analog bir filtrenin s-domenindeki transfer fonksiyonunu z-domeninde ayrık-zaman eşdeğer transfer fonksiyonuna dönüştüren bir tekniktir.

5.1.Giriş

$$y(n)=\sum_{k=0}^{M} b_k x(n-k)-\sum_{l=1}^{N} a_l y(n-l) \qquad (5.1)$$

şeklinde verilen bir giriş-çıkış eşitliğini ele alalım. Bu eşitlik,

$$\begin{aligned} y(n)=&\,b_0x(n)+b_1x(n-1)+b_2x(n-2)+\cdots+b_Mx(n-M)\\ &-a_1y(n-1)-a_2y(n-2)-\cdots-a_Ny(n-N)\end{aligned} \qquad (5.2)$$

eşitliği ile eşdeğerdir. Bu yinelemeli eşitlik IIR bir filtreyi ifade etmektedir. Çıkış, giriş kadar çıkışın önceki değerlerine (geri besleme ile) de bağlıdır. Eşitlik (5.2)'den görüldüğü gibi, n anındaki çıkış y(n), sadece girişin o andaki değeri x(n) ve geçmiş değerleri x(n-1),…,x(n-M)'e değil çıkışın önceki değerleri y(n-1),…,y(n-N)'e de bağlıdır.

Tüm başlangıç koşulları sıfır kabul edilirse Eşitlik (5.2)'nin z-dönüşümü

$$Y(z)=(b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Nz^{-N})X(z)-(a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M})Y(z) \quad (5.3)$$

ile verilir. N=M kabul edilirse, transfer fonksiyonu H(z),

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Nz^{-N}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Nz^{-N}}=\frac{N(z)}{D(z)} \qquad (5.4)$$

şeklinde elde edilir. Burada N(z) ve D(z) sırasıyla pay ve payda polinomlarını göstermektedir. transfer fonksiyonunu z^n ile çarpıp bölersek,

$$H(z)=\frac{b_0z^N+b_1z^{N-1}+b_2z^{N-2}+\cdots+b_N}{z^N+a_1z^{N-1}+a_2z^{N-2}+\cdots+a_N}=C\prod_{i=1}^{N}\frac{z-z_i}{p-p_i} \qquad (5.5)$$

Bu transfer fonksiyonunun N adet kutbu ve N adet sıfırı vardır. Eşitlik (5.5)'deki tüm a katsayıları sıfır olursa transfer fonksiyonu N kutbu z-düzleminde orijinde olan FIR bir filtrenin transfer fonksiyonuna indirgenir ve her zaman kararlıdır. IIR bir filtrenin kararlı olabilmesi için her bir kutbunun genliği 1'den küçük olmalıdır. Yani,

1. $|p_i < 1|$ ise, $n \to \infty \; iken \; h(n) \to 0$ olur ve sistem karalıdır.
2. $|p_i| > 1$ ise $n \to \infty \; iken \; h(n) \to \infty$ olur ve sistem karasızdır.
3. $|p_i| = 1$ ise sistem kısmen karalıdır ve osilasyon yapar.

Ayrıca, birim dairedeki katlı kutuplar da sistemin karasız olmasına neden olur. Tüm a katsayılarının sıfır olmasının sistemin yinelemesiz ve karalı bir FIR filtreye dönüşmesine neden olduğu unutulmamalıdır.

5. 2.IIR Filtre Yapıları

IIR bir filtreyi göstermek için kullanılan farklı yapılar vardır. Bu yapılar aşağıda tartışılmıştır.

5.2.1. Direkt Şekil-I Yapısı

Şekil 5.1'de gösterilen direkt şekil-I yapısı ile Eşitlik (5.2) ile verilen filtre gerçekleştirilebilir. N. dereceden bir filtre için bu yapıda, z^{-1} ile gösterilen 2N adet gecikme elemanı vardır. Örneğin, ikinci dereceden bir filtre için dört gecikme elemanı kullanılır.

5.2.2. Direkt Şekil-II Yapısı

Şekil (5.2)'de verilen direkt şekil-II yapısı en yaygın olarak kullanılan yapılardan biridir. Direkt şekil-I'e göre yarı yarıya daha az gecikme elemanı gerektirir. Örneğin, ikinci derecen bir filtre iki adet gecikme elemanı gerektirir.

Şekil 5.2'de verilen blok diyagramdan,

$$w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2) - \cdots - a_N w(n-N) \qquad (5.6)$$

$$y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) + \cdots + b_N w(n-N) \qquad (5.7)$$

olduğu görülmektedir. Eşitlik (5. 6) ve (5.7)'nin z-dönüşümü alınarak

$$W(z) = X(z) - a_1 z^{-1} W(z) - a_2 z^{-2} W(z) - \cdots - a_N z^{-N} W(z) \qquad (5.8)$$

elde edilir. O halde,

$$X(z) = (1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}) W(z) \qquad (5.9)$$

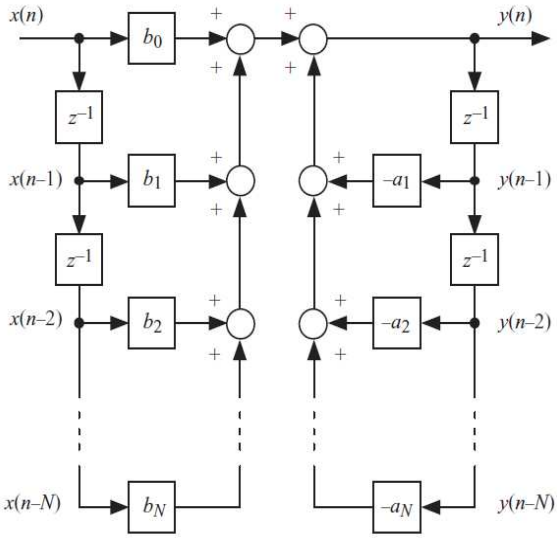
$$Y(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}) W(z) \qquad (5.10)$$

olacaktır. Transfer fonksiyonu,

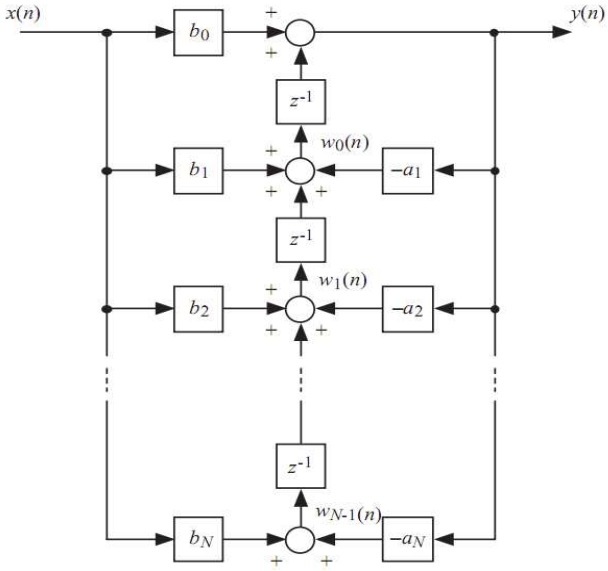
$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}} \qquad (5.11)$$

Görüldüğü gibi, Eşitlik (5.11) Eşitlik (5.4)'ün aynısıdır.

Direkt şekil-II yapı, fark eşitlikleri olan Eşitlik (5.6) ve (5.7) , Eşitlik (5.2)'de yerine konularak ifade edilebilir.



Şekil 5.1. Direkt Şekil-I IIR Filtre Yapısı.



Şekil 5.2. Direkt Şekil-II IIR Filtre Yapısı.

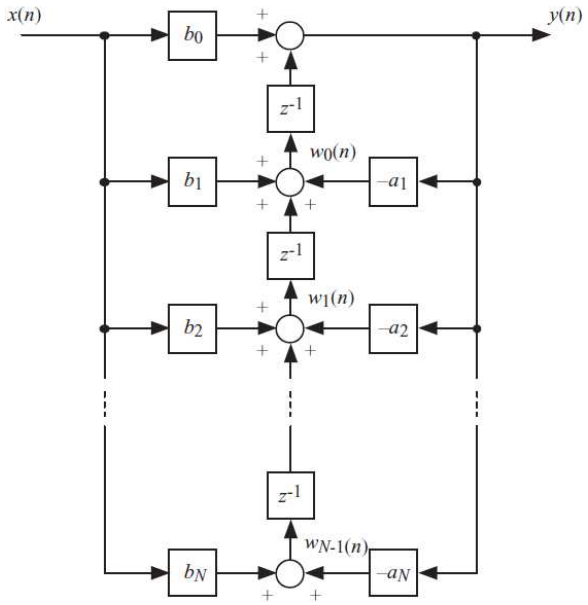
(5.6) ve (5.7) eşitlikleri IIR bir filtreyi programlamak için kullanılır. Başlangıç olarak, w(n-1), w(n-2),… sıfıra eşitlenir. n anında yeni bir x(n) örneği elde edilir ve (5.6) eşitliği w(n)'e göre çözülür. Daha sonra, Eşitlik (5.7) kullanılarak y(n) hesaplanır.

5.2.3. Direkt Şekil-II Evrik Yapı

Şekil 5.3'de gösterilen direkt şekil-II evrik yapı, direkt şekil-II'nin düzenlenmiş bir versiyonudur ve aynı sayıda gecikme elemanı gerektirir.

Blok diyagramdan, filtre çıkışı aşağıdaki şekilde hesaplanabilir

$$y(n) = b_0 x(n) + w_0(n-1) \qquad (5.12)$$



Şekil 5.3. Direkt Şekil-II Evrik IIR Filtre Yapısı.

Daha sonra, gecikme hattında bulunanlar

$$w_0(n) = b_1 x(n) + w_1(n-1) - a_1 y(n) \qquad (5.13)$$

$$w_1(n) = b_2 x(n) + w_2(n-1) - a_2 y(n) \qquad (5.14)$$

eşitlikleri ile güncellenirler ve

$$w_{N-1}(n) = b_N x(n) - a_N y(n) \qquad (5.15)$$

elde edilene kadar bu şekilde devam edilir. $w_0(n-1)$'i bulmak için Eşitlik (5.13) kullanılırsa,

$$w_0(n-1) = b_1 x(n-1) + w_1(n-2) - a_1 y(n-1)$$

$$y(n) = b_0 x(n) + [b_1 x(n-1) + w_1(n-2) - a_1 y(n-1)]$$

Benzer şekilde, $w_1(n-2)$'yi bulmak için Eşitlik (5.14) kullanılırsa

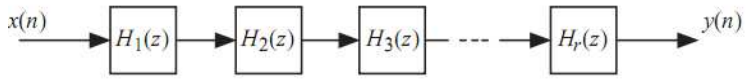
$$w_1(n-2) = b_2 x(n-2) + w_2(n-3) - a_2 y(n-2)$$

$$y(n) = b_0 x(n) + [b_1 x(n-1) + [b_2 x(n-2) + w_2(n-3) - a_2 y(n-2)] - a_1 y(n-1)] \qquad (5.16)$$

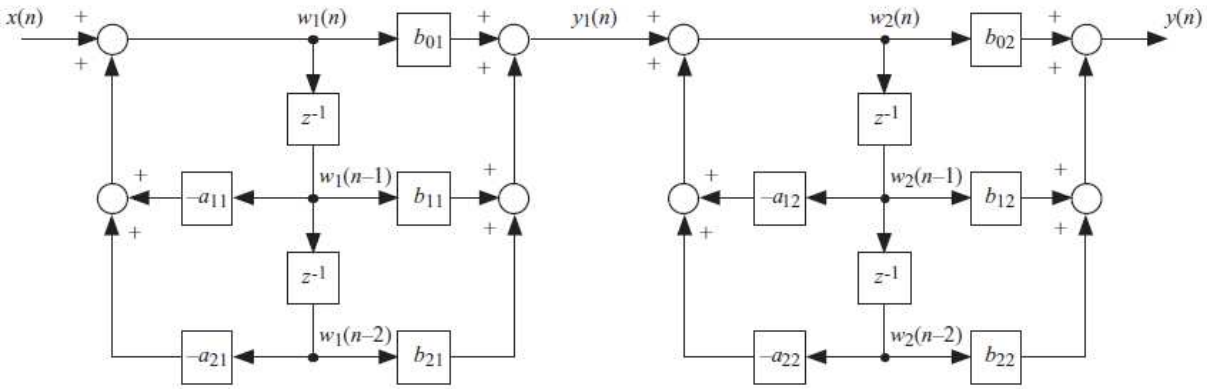
Eşitlik (5.15)'e kullanılana kadar bu yöntemi devam ettirerek, Eşitlik (5.12)'nin Eşitlik (5.2)'ye eşdeğer olduğu gösterilebilir. O halde, Şekil (5.3)'deki blok diyagram da Şekil 5.1 ve Şekil 5.2'dekine eşdeğerdir. Evrikleştirilmiş yapı ilk olarak sıfırları sonra kutupları gerçekleştirir oysa direkt şekil-II yapısı ilk olarak kutupları gerçekleştirir.

5.2.4. Kaskad Yapı

Eşitlik (5.5) ile verilen transfer fonksiyonu birinci ve ikinci dereceden transfer fonksiyonları cinsinden aşağıdaki şekilde yazılabilir



Şekil 5.4. Kaskad IIR Filtre Yapısı.



Şekil 5.5. Kaskad bağlı direkt şekil-II iki parçadan oluşmuş dördüncü dereceden IIR bir filtre.

$$H(z) = CH_1(z)H_2(z)\cdots H_r(z)$$

Kaskad (ya da seri) yapı Şekil 5.4'de gösterilmiştir. Sistemin tamamının transfer fonksiyonu kaskad transfer fonksiyonları ile gösterilebilir. Her bir parça için, direkt şekil-II yapısı veya evriği kullanılabilir. Şekil 5.5'de dördüncü dereceden IIR bir filtre kaskad bağlı direkt şekil-II ile gerçekleştirilmiş ikinci dereceden iki filtre olarak gösterilmektedir. Transfer fonksiyonu H(z), kaskad bağlı ikinci dereceden transfer fonksiyonları cinsinden yazılabilir.

$$H(z) = \prod_{i=1}^{N/2} \frac{b_{0i} + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}} \qquad (5.18)$$

Eşitlik (5.17)'deki C sabiti katsayıların içerisine dahil edilmiştir ve farklı her bir parça *i* ile gösterilmiştir. Örneğin dördüncü dereceden bir transfer fonksiyonu için N=4'dür ve Eşitlik (5.8)

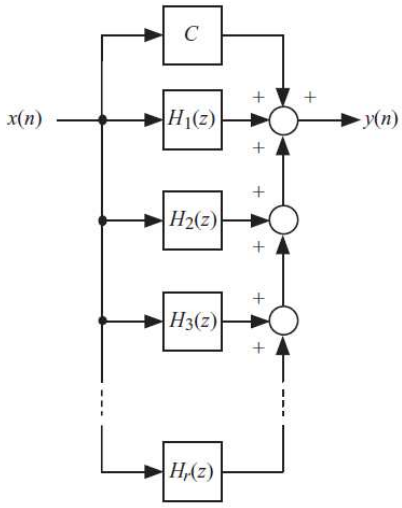
$$H(z) = \frac{(b_{01} + b_{11}z^{-1} + b_{21}z^{-2})(b_{02} + b_{12}z^{-1} + b_{22}z^{-2})}{(1 + a_{11}z^{-1} + a_{21}z^{-2})(1 + a_{12}z^{-1} + a_{22}z^{-2})} \qquad (5.19)$$

şeklinde yeniden düzenlenebilir. Bu eşitlik Şekil (5.5) ile de doğrulanmıştır.

Matematiksel açıdan pay ve payda katsayılarının doğru sıralanması çıkışı etkilemez. Ancak, pratik açıdan her bir parçanın doğru sırada olması kuantalama gürültüsünü azaltır. Birinci parçanın çıkışının ikinci parçanın girişi olduğuna dikkat ediniz. Bu kısımda verilecek bir programlama örneği IIR bir filtrenin ikinci dereceden direkt şekil-II yapısı kullanılarak seri şekilde nasıl gerçekleştirildiğini göstermektedir.

5.2.5 Paralel Şekil Yapı

Eşitlik (5.5)'de verilen transfer fonksiyonu kısmi kesirlerine ayırma (PFE) yöntemi kullanılarak



Şekil 5.6. Paralel şekil IIR filtre yapısı.

$$H(z) = C + H_1(z) + H_2(z) + \cdots + H_{r1}(z)$$

şeklinde gösterilebilir. Paralel yapı Şekil 5.6'da gösterilmiştir. Transfer fonksiyonlarından her biri birinci ya da ikinci dereceden fonksiyonlar olabilir. Paralel şekil etkin bir şekilde ikinci dereceden direkt şekil-II yapıları ile gösterilebilir. H(z) aşağıdaki şeklide ifade edilebilir

$$H(z) = C + \sum_{i=1}^{N/2} \frac{b_{0i} + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}} \qquad (5.21)$$

Örneğin, dördüncü dereceden bir transfer fonksiyonu için Eşitlik (5.21)'deki H(z)

$$H(z) = C + \frac{b_{01} + b_{11}z^{-1} + b_{21}z^{-2}}{1 + a_{12}z^{-1} + a_{22}z^{-2}} + \frac{b_{02} + b_{12}z^{-1} + b_{22}z^{-2}}{1 + a_{12}z^{-1} + a_{22}z^{-2}} \qquad (5.22)$$

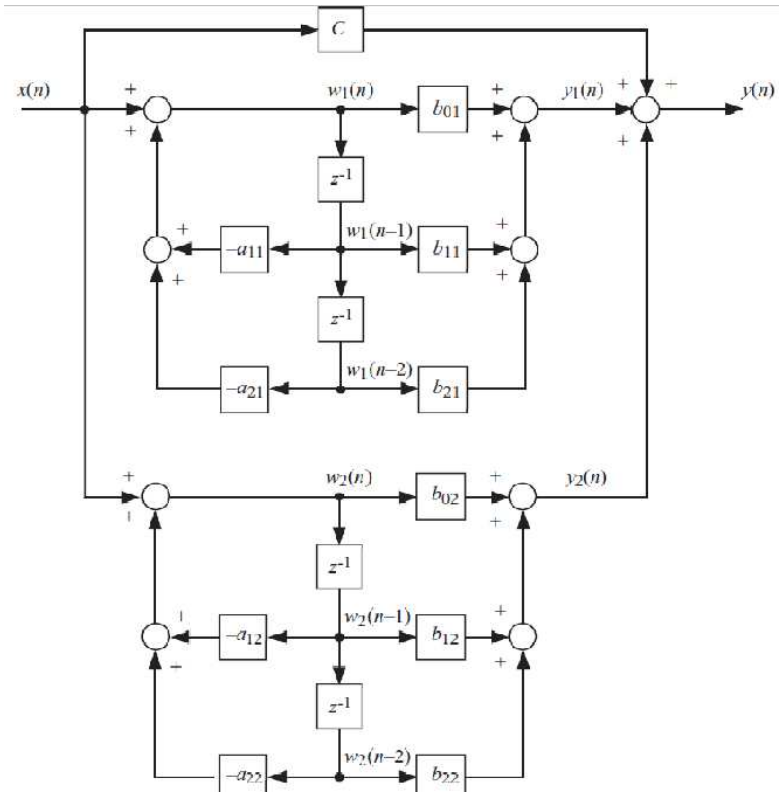
şeklinde yeniden düzenlenebilir. Dördüncü dereceden bu paralel yapı Şekil 5.7'de gösterildiği gibi ikinci dereceden iki direkt şekil-II parça ile gösterilebilir. Şekilden, y(n) çıkışı her bir parçanın çıkışına bağlı olarak ifade edilebilir

$$y(n) = Cx(n) + \sum_{i=1}^{N/2} y_i(n) \qquad (5.23)$$

IIR filtrenin katsayılarına ilişkin kuantalama hatası, filtrenin kutuplarının ve sıfırlarının karmaşık düzlemdeki yerinin kayma miktarına bağlıdır. Yani belirli bir kutbun karmaşık düzlemdeki yerinin değişmesi diğer kutupların tümünün konumuna bağlıdır. Kutuplar arasındaki bu bağımlılığı en küçük yapmak için, N. dereceden IIR bir filtre ikinci dereceden filtrelerin seri bağlanması ile gerçekleştirilir.

5.3. Çift Doğrusal Dönüşüm (BLT)

BLT analog bir filtreyi sayısal bir filtreye dönüştürmek için kullanılan en yaygın tekniktir. Bu dönüşüm



Şekil 5.7. Paralel bağlı ikinci dereceden iki direkt şekil-II parçadan oluşan dördüncü dereceden IIR filtre.

$$s = K\frac{(z-1)}{(z+1)} \qquad (5.24)$$

eşitliğini kullanarak analog s-düzleminden z-düzlemine bire bir geçiş sağlar. T saniyedeki örnekleme sayısı olmak üzere, K sabiti genellikle 2/T olarak seçilir. Sonraki kısımlarda K'nın diğer değerleri de verilecektir. Bu dönüşüm sayesinde

i) s-düzleminin sol tarafı z-düzleminde birim dairenin içine

ii) s-düzleminin sağ tarafı z-düzleminde birim dairenin dışına

iii) s-düzleminde sanal jw ekseni z-düzleminde birim dairenin üzerine

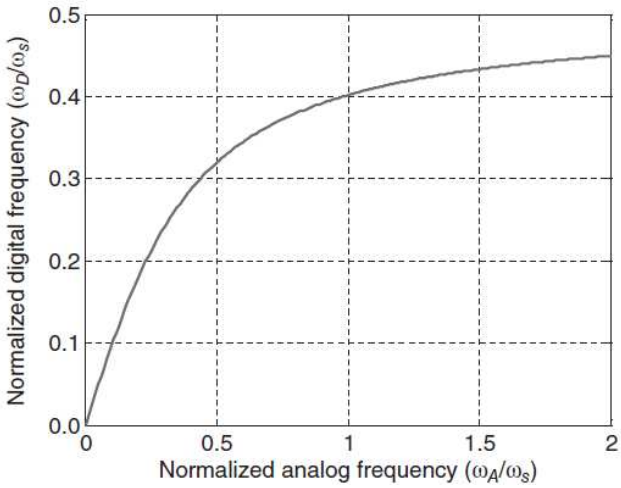
haritalanır. w_A ve w_D sırasıyla analog ve sayısal frekansları göstersin. s=jw$_A$ ve z=e^{jw_DT} olan-k üzereEşitlik (5.24),

$$j\omega_A = K\frac{(e^{j\omega_D T}-1)}{(e^{j\omega_D T}+1)} = K\frac{e^{j\omega_D T/2}(e^{j\omega_D T/2}-e^{-j\omega_D T/2})}{e^{j\omega_D T/2}(e^{j\omega_D T/2}+e^{-j\omega_D T/2})} \qquad (5.25)$$

olacaktır. Karmaşık üstel fonksiyonlar cinsinden sinüs ve kosinüs için Euler bağlantılarını kullanarak Eşitlik (5.25)'deki w_A

$$\omega_A = K\tan\left(\frac{\omega_D T}{2}\right) \qquad (5.26)$$

elde edilir. Bu eşitlik analog frekansla sayısal frekans arasındaki ilişkiyi verir. Bu ilişki pozitif w$_A$ değerleri için Şekil 5.8'de çizilmiştir.



Şekil 5.8. Analog ve sayısal frekanslar arasındaki ilişki.

5.3.1. BLT Tasarım Yöntemi

H(s) transfer fonksiyonuna sahip analog bir filtreyi H(z) transfer fonksiyonlu sayısal bir filtreye BLT yöntemi ile dönüştürmek şu şekilde ifade edilebilir:

$$H(z) = H(s)\big|_{s=2(z-1)/T(z+1)}$$

H(s) analog filtre tasarımından iyi bilinen filtrelerden birinin (Butterworth, Chebyshev, Bessel vb.) transfer fonksiyonu olarak seçilebilir. Genellikle K sabiti 2/T seçilir. Ancak

$$K = \frac{\omega_c}{\tan(\pi\omega_c/\omega_S)}$$

olarak da seçilebilir. Burada $w_A = w_c = w_D$ 'dir.