

PETUNJUK PRAKTIKUM

PRAKTIKUM

PENGOLAHAN SINYAL DIGITAL



**Laboratorium Dasar
Teknik Elektro**

**Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
2019**

**BUKU PETUNJUK
PRAKTIKUM PENGOLAHAN SISTEM DIGITAL
EL 3110**

Mervin T. Hutabarat

Armein Z. R. Langi

Yoanes Bandung

Erwin Cahyadi

Nina Lestari

Laboratorium Dasar Teknik Elektro

**Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
2019**

DAFTAR ISI

DAFTAR ISI.....	i
ATURAN UMUM LABORATORIUM.....	i
Kelengkapan.....	i
Persiapan	i
Sebelum Praktikum	i
Selama Praktikum	i
Setelah praktikum	i
pergantian jadwal.....	ii
Kasus biasa.....	ii
Kasus sakit atau urusan mendesak pribadi lainnya.....	ii
Kasus "kepentingan massal".....	iii
sanksi.....	iii
PANDUAN UMUM KESELAMATAN DAN PENGGUNAAN PERALATAN LABORATORIUM	iv
Keselamatan	iv
Bahaya listrik.....	iv
Bahaya api atau Panas berlebih	v
Bahaya benda Tajam dan logam	v
Lain-lain.....	v
Penggunaan PERalatAN Praktikum	v
sanksi.....	vi
TABEL SANKSI PRAKTIKUM	vii
PERCOBAAN I	1
Pengenalan MATLAB	1
1. TUJUAN	1
2. DASAR TEORI	1
2.1. MATLAB HELP.....	2
2.2. VARIABEL DAN OPERASI MATRIKS	2

2.2. 1. Operator colon (:)	3
2.2.2. Operasi Matriks dan Array	3
2.4. KONSTRUK PEMROGRAMAN	6
2.5. MATLAB SCRIPTS	7
2.6. MENULIS FUNGSI MATLAB	9
2.7. TIPS PEMROGRAMAN	12
3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN	17
4. PERCOBAAN	17
4.1. PERALATAN YANG DIGUNAKAN	17
4.2. PROSEDUR PRAKTIKUM	17
5. MENGAKHIRI PERCOBAAN	19
CONTOH FORMAT LAPORAN	20
1.1. Perancangan Filter dengan Matlab	20
1.2. Desain Filter FIR	20
PERCOBAAN II	21
SIMULASI FILTER FIR REALTIME	21
1. TUJUAN	21
2. DASAR TEORI	21
2.1. FILTER FIR REALTIME	21
2.2. ISU NUMERIK	22
3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN	23
4. PERCOBAAN	23
4.1. PERALATAN YANG DIGUNAKAN	23
4.2. PROSEDUR PRAKTIKUM	23
5. MENGAKHIRI PERCOBAAN	26
PERCOBAAN III	27
PENGUNAAN VISUAL DSP++ 5.0	27
1. TUJUAN	27
2. DASAR TEORI	27
2.1. View	27
2.1.2. View grafik waktu dan frekuensi	28

3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN	29
4. PERCOBAAN	30
4.1. PERALATAN YANG DIGUNAKAN.....	30
4.2. PROSEDUR PRAKTIKUM	30
5. MENGAKHIRI PERCOBAAN	35
PERCOBAAN IV	36
DESAIN FILTER DAN IMPLEMENTASI ALGORITMA DSP	36
1. TUJUAN	36
2. DASAR TEORI	36
2.1. Periferal-periferal ADSP-BF561	37
2.2. Arsitektur Inti (Core) ADSP-BF561.....	38
2.3. Dasar Teori Pemfilteran.....	40
2.4. Penyimpanan koefisien filter dan sinyal input	41
2.5. White Noise.....	42
3. TUGAS PENDAHULUAN	42
4. PERCOBAAN	43
4.1. PERALATAN YANG DIGUNAKAN.....	43
4.2. PROSEDUR PRAKTIKUM	43
4.2.1. Percobaan Implementasi filter FIR : low-pass filter	43
4.2.2. Percobaan Implementasi filter FIR: Band-Pass Filter	44
4.2.3. Percobaan Implementasi filter FIR: High Pass Filter	45
5. MENGAKHIRI PERCOBAAN	46
PERCOBAAN V	48
IMPLEMENTASI ALGORITMA DSP LANJUTAN	48
1. TUJUAN	48
2. DASAR TEORI	48
2.1. Scrambler	48
2.2. Efek Pelebaran Stereo	48
3. TUGAS PENDAHULUAN	49
4. PERCOBAAN	49
4.1. PERALATAN YANG DIGUNAKAN.....	49
4.2. PROSEDUR PRAKTIKUM	49

5. MENGAKHIRI PERCOBAAN	51
PERCOBAAN VI	52
DESAIN DAN IMPLEMENTASI FILTER IIR PADA BLACKFIN DSP.....	52
1. TUJUAN	52
2. TUGAS BELAJAR DAN TUGAS PENDAHULUAN	52
3. PERCOBAAN	53

ATURAN UMUM LABORATORIUM

KELENGKAPAN

Setiap praktikan wajib berpakaian lengkap, mengenakan celana panjang/ rok, kemeja dan mengenakan sepatu. Praktikan wajib membawa kelengkapan berikut:

- Modul praktikum
- Buku Catatan Laboratorium (BCL)
- Alat tulis (dan kalkulator, jika diperlukan)
- *Name tag*
- Kartu Praktikum

PERSIAPAN

SEBELUM PRAKTIKUM

- Membaca dan memahami isi modul praktikum
- Mengerjakan hal-hal yang dapat dikerjakan sebelum praktikum dilaksanakan, misalnya mengerjakan soal perhitungan, membuat *source code*, mengisi Kartu Praktikum dll.
- Mengerjakan Tugas Pendahuluan
- Mengisi daftar hadir
- Mengambil kunci loker dan melengkapi administrasi peminjaman kunci loker (tukarkan dengan kartu identitas: KTM/ SIM/ KTP)

SELAMA PRAKTIKUM

- Menuliskan identitas diri pada Berita Acara Praktikum yang diedarkan oleh asisten,
- Perhatikan dan kerjakan setiap percobaan dengan waktu sebaik-baiknya, diawali dengan kehadiran praktikan secara tepat waktu
- Kumpulkan Kartu Praktikum pada asisten
- Dokumentasikan pada BCL (lihat Petunjuk Penggunaan BCL) tentang hal-hal penting terkait percobaan yang sedang dilakukan

SETELAH PRAKTIKUM

- Memastikan BCL telah ditandatangani oleh asisten,

- Mengembalikan kunci loker dan melengkapi administrasi pengembalian kunci loker (pastikan kartu identitas KTM/ SIM/ KTP diperoleh kembali),
- Mengerjakan laporan dalam bentuk SoftCopy (lihat Panduan Penyusunan Laporan di laman <http://ldte.stei.itb.ac.id>),
- Mengirimkan file dengan cara mengunggah di laman <http://praktikum.ee.itb.ac.id>. Waktu pengiriman paling lambat jam 11.00 WIB, dua hari kerja berikutnya setelah praktikum kecuali ada kesepakatan lain antara Dosen Pengajar dan/ atau Asisten

PERGANTIAN JADWAL

KASUS BIASA

- Lihatlah format Pertukaran Jadwal di <http://ldte.stei.itb.ac.id> pada halaman Panduan
- Salah satu praktikan yang bertukar jadwal harus mengirimkan e-mail ke labdasar@stei.itb.ac.id . Waktu pengiriman paling lambat jam 16.30, sehari sebelum praktikum yang dipertukarkan
- Pertukaran diperbolehkan setelah ada email konfirmasi dari Lab. Dasar

KASUS SAKIT ATAU URUSAN MENDESAK PRIBADI LAINNYA

Jadwal pengganti dapat diberikan kepada praktikan yang sakit atau memiliki urusan mendesak pribadi.

- Praktikan yang hendak mengubah jadwal untuk urusan pribadi mendesak harus memberitahu staf tata usaha laboratorium sebelum jadwal praktikumnya melalui email.
- Segera setelah praktikan memungkinkan mengikuti kegiatan akademik, praktikan dapat mengikuti praktikum pengganti setelah mendapatkan konfirmasi dari staf tata usaha laboratorium dengan melampirkan surat

keterangan dokter bagi yang sakit atau surat terkait untuk yang memiliki urusan pribadi.

KASUS "KEPENTINGAN MASSAL"

- "Kepentingan massal" terjadi jika ada lebih dari 1/3 rombongan praktikan yang tidak dapat melaksanakan praktikum pada satu hari yang sama karena alasan yang terkait kegiatan akademis
- Isi Form Pergantian Jadwal dan serahkan pada TU Lab. Dasar secepatnya. Jadwal praktikum pengganti satu hari itu akan ditentukan kemudian oleh Kordas praktikum yang bersangkutan

SANKSI

Pengabaian aturan-aturan di atas dapat dikenakan sanksi pengurangan nilai praktikum terkait.

PANDUAN UMUM KESELAMATAN DAN PENGGUNAAN PERALATAN LABORATORIUM

KESELAMATAN

Pada prinsipnya, untuk mewujudkan praktikum yang aman diperlukan partisipasi seluruh praktikan dan asisten pada praktikum yang bersangkutan. Dengan demikian, kepatuhan setiap praktikan terhadap uraian panduan pada bagian ini akan sangat membantu mewujudkan praktikum yang aman.

BAHAYA LISTRIK

- Perhatikan dan pelajari tempat-tempat sumber listrik (stop-kontak dan *circuit breaker*) dan cara menyala-matikannya. Jika melihat ada kerusakan yang berpotensi menimbulkan bahaya, laporkan pada asisten
- Hindari daerah atau benda yang berpotensi menimbulkan bahaya listrik (sengatan listrik/ *strum*) secara tidak disengaja, misalnya kabel jala-jala yang terkelupas dll.
- Tidak melakukan sesuatu yang dapat menimbulkan bahaya listrik pada diri sendiri atau orang lain
- Keringkan bagian tubuh yang basah karena, misalnya, keringat atau sisa air wudhu
- Selalu waspada terhadap bahaya listrik pada setiap aktivitas praktikum

Kecelakaan akibat bahaya listrik yang sering terjadi adalah tersengat arus listrik. Berikut ini adalah hal-hal yang harus diikuti praktikan jika hal itu terjadi:

- Jangan panik
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing dan di meja praktikan yang tersengat arus listrik
- Bantu praktikan yang tersengat arus listrik untuk melepaskan diri dari sumber listrik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya kecelakaan akibat bahaya listrik

BAHAYA API ATAU PANAS BERLEBIH

- Jangan membawa benda-benda mudah terbakar (korek api, gas dll.) ke dalam ruang praktikum bila tidak disyaratkan dalam modul praktikum
- Jangan melakukan sesuatu yang dapat menimbulkan api, percikan api atau panas yang berlebihan
- Jangan melakukan sesuatu yang dapat menimbulkan bahaya api atau panas berlebih pada diri sendiri atau orang lain
- Selalu waspada terhadap bahaya api atau panas berlebih pada setiap aktivitas praktikum

Berikut ini adalah hal-hal yang harus diikuti praktikan jika menghadapi bahaya api atau panas berlebih:

- Jangan panik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya bahaya api atau panas berlebih
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing
- Menjauh dari ruang praktikum

BAHAYA BENDA TAJAM DAN LOGAM

- Dilarang membawa benda tajam (pisau, gunting dan sejenisnya) ke ruang praktikum bila tidak diperlukan untuk pelaksanaan percobaan
- Dilarang memakai perhiasan dari logam misalnya cincin, kalung, gelang dll.
- Hindari daerah, benda atau logam yang memiliki bagian tajam dan dapat melukai
- Tidak melakukan sesuatu yang dapat menimbulkan luka pada diri sendiri atau orang lain

LAIN-LAIN

- Dilarang membawa makanan dan minuman ke dalam ruang praktikum

PENGUNAAN PERALATAN PRAKTIKUM

Berikut ini adalah panduan yang harus dipatuhi ketika menggunakan alat-alat praktikum:

- Sebelum menggunakan alat-alat praktikum, pahami petunjuk penggunaan alat itu. Petunjuk penggunaan beberapa alat dapat didownload di <http://labdasar.ee.itb.ac.id>
- Perhatikan dan patuhi peringatan (*warning*) yang biasa tertera pada badan alat

- Pahami fungsi atau peruntukan alat-alat praktikum dan gunakanlah alat-alat tersebut hanya untuk aktivitas yang sesuai fungsi atau peruntukannya. Menggunakan alat praktikum di luar fungsi atau peruntukannya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan
- Pahami *rating* dan jangkauan kerja alat-alat praktikum dan gunakanlah alat-alat tersebut sesuai *rating* dan jangkauan kerjanya. Menggunakan alat praktikum di luar *rating* dan jangkauan kerjanya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan
- Pastikan seluruh peralatan praktikum yang digunakan aman dari benda/ logam tajam, api/ panas berlebih atau lainnya yang dapat mengakibatkan kerusakan pada alat tersebut
- Tidak melakukan aktifitas yang dapat menyebabkan kotor, coretan, goresan atau sejenisnya pada badan alat-alat praktikum yang digunakan

SANKSI

Pengabaian uraian panduan di atas dapat dikenakan sanksi tidak lulus mata kuliah praktikum yang bersangkutan

TABEL SANKSI PRAKTIKUM

Berlaku mulai: 14 Agustus 2017

Level	Waktu	Kasus	Sanksi	Pengurangan nilai per modul
Akademik	Saat dan setelah praktikum	Semua kegiatan plagiasi (mencontek): tugas pendahuluan, test dalam praktikum, laporan praktikum	Gugur praktikum	
		Sengaja tidak mengikuti praktikum		
Berat	Saat praktikum	Tidak hadir praktikum	Gugur modul	
		Terlambat hadir praktikum		
		Pakaian tidak sesuai: kemeja, sepatu		
		Tugas pendahuluan tidak dikerjakan/hilang/tertinggal		
Ringan	Saat Praktikum	Pertukaran jadwal tidak sesuai aturan/ketentuan		-25 nilai akhir
		Tidak mempelajari modul sebelum praktikum/tidak mengerti isi modul	Dikeluarkan dari praktikum	-25 nilai akhir
		BCL tertinggal/hilang		-100% nilai BCL
		Name Tag tertinggal/hilang		-10 nilai akhir
		Kartu praktikum tertinggal/hilang		-25 nilai akhir
		Kartu praktikum tidak lengkap data dan foto		-10 nilai akhir
		Loker tidak dikunci/kunci tertinggal		-10 nilai akhir
	Setelah Praktikum	Tidak ada paraf asisten di BCL/kartu praktikum		-25 nilai akhir
		Terlambat mengumpulkan laporan		-1/min nilai akhir, maks -50
		Terlambat mengumpulkan BCL		-1/min nilai BCL, maks -50
		Tidak bawa kartu praktikum saat pengumpulan BCL		-50 nilai BCL
		Tidak minta paraf admin saat pengumpulan BCL		-50 nilai BCL

Catatan:

1. Pelanggaran akademik menyebabkan gugur praktikum, nilai praktikum E
2. Dalam satu praktikum, praktikan maksimal boleh melakukan
 - a. 1 pelanggaran berat dan 1 pelanggaran ringan; atau
 - b. 3 pelanggaran ringan
3. Jika jumlah pelanggaran melewati point 2, praktikan dianggap gugur praktikum.
4. Praktikan yang terkena sanksi gugur modul wajib mengganti praktikum pada hari lain dengan nilai modul tetap 0. Waktu pengganti praktikum ditetapkan bersama asisten. Jika praktikan tidak mengikuti ketentuan praktikum (pengganti) dengan baik, akan dikenakan sanksi gugur praktikum.
5. Setiap pelanggaran berat dan ringan dicatat/diberikan tanda di kartu praktikum
6. Waktu acuan adalah waktu sinkron dengan NIST
7. Sanksi yang tercantum di tabel adalah sanksi minimum.
8. Sanksi yang belum tercantum akan ditentukan kemudian.

PLAGIARISME DAN KECURANGAN AKADEMIK

Plagiarisme merupakan salah satu bentuk kecurangan akademik. Definisi plagiarisme sesuai Peraturan Akademik ITB adalah menggunakan kata-kata atau karya orang lain sebagai kata-kata atau karya sendiri dalam suatu kegiatan akademik tanpa menyebutkan acuan yang dipakai. Plagiarisme bisa dilakukan secara sengaja, akibat kecerobohan, maupun tidak sengaja. Plagiarisme merupakan pelanggaran integritas akademik. Prinsip kejujuran intelektual menyiratkan bahwa semua anggota komunitas akademik harus mengakui peran pemilik gagasan awal dalam hal kata-kata dan data yang membentuk dasar untuk pekerjaan mereka sendiri. Mengakui karya orang lain sebagai milik anda memberi makna bahwa anda telah gagal menyelesaikan proses pembelajaran. Plagiarisme adalah sangat tidak etis dan memiliki konsekuensi serius bagi karir masa depan Anda sekaligus merusak reputasi institusi.

BENTUK-BENTUK PLAGIARISME :

1. Mengutip kata demi kata (Verbatim)
2. Parafrase: menuliskan kembali karya hasil orang lain dengan mengubah kata atau mengubah urutan kalimat, dengan mengikuti struktur argumen orang lain tersebut tanpa menyebutkan acuan.
3. Kolusi: kolaborasi tidak sah antar mahasiswa tanpa atribusi terhadap bantuan dari luar yang diterima, atau tidak mengikuti sebenarnya pada peraturan kerja berkelompok
4. Kutipan tidak akurat: salah kutip atau mencantumkan referensi yang tidak pernah dikutip.
5. Apresiasi (acknowledgement) tidak akurat: tidak menyebutkan kontribusi pihak yang berkontribusi atau sebaliknya memberi apresiasi pada pihak yang tidak berkontribusi.
6. Menggunakan jasa pihak ketiga, profesional maupun tidak.

PRINSIP MENGHINDARI PLAGIARISME :

1. Semua karya ilmiah harus dilandasi latar belakang, motivasi, dan lain sebagainya yang bisa dipertanggungjawabkan secara ilmiah. Adalah wajib untuk menggunakan referensi untuk mendukung ide-ide yang telah Anda kembangkan.
2. Dalam karya ilmiah, Anda harus menunjukkan bahwa Anda memiliki pemahaman yang jelas dan benar tentang materi yang telah Anda dapatkan dari referensi.
3. Berikan kejelasan antara analisa (ide) original Anda dengan apa yang telah diambil dari referensi:
 - Berikan penanda bagian mana suatu paragraf adalah berasal dari referensi.
 - Kutipan harus selalu diidentifikasi dengan menggunakan tanda kutip atau indentasi, dan dengan referensi penuh dari sumber yang dikutip.
 - Untuk menghindari parafrase, lebih baik menuliskan kembali ringkasan singkat dari keseluruhan sumber dengan kata-kata sendiri, dan dengan jelas

menunjukkan bahwa itu yang dilakukan sehingga jelas bagian mana yang merupakan ide original Anda, mana yang diambil dari referensi.

- Untuk menghindari kolusi, adalah tanggung jawab Anda untuk memastikan bahwa Anda sepenuhnya jelas tentang sejauh mana kolaborasi/kerja kelompok diizinkan, dan bagian mana dari pekerjaan itu harus Anda kerjakan sendiri.
- Tidak boleh memasukkan apa pun dalam referensi atau bibliografi yang sebenarnya tidak direferensikan.
- Jika akses ke sumber utama tidak diperoleh, boleh menggunakan teks sekunder.
- Sitasi (menyebutkan) referensi harus diikuti dengan identifikasi pengutipannya dalam paragraf.

KECURANGAN AKADEMIK DALAM PELAKSANAAN PRAKTIKUM:

Tugas pendahuluan harus dikerjakan sendiri dalam setiap aspeknya, baik apabila tugas berupa analisis, perhitungan, atau simulasi. Kegiatan mencontoh atau meniru tugas pendahuluan tidak diperkenankan, dan apabila terbukti/bisa dibuktikan dapat dianggap melakukan kecurangan akademik seperti halnya mencontek. Apabila tugas yang diberikan membutuhkan referensi dari buku, internet dan sejenisnya, berlaku aturan plagiarisme. Untuk menghindari plagiarisme dalam mengerjakan tugas pendahuluan yang membutuhkan referensi, gunakan minimal 3 referensi dengan melakukan elaborasi dari referensi-referensi tersebut. Hindari dalam menggunakan hanya satu referensi meskipun dengan melakukan parafrase.

Tes awal termasuk dalam kategori yang sama dengan kuis atau **ujian**, dimana segala bentuk upaya mendapatkan bantuan dari pihak luar (mencontek pekerjaan peserta lain dengan bekerjasama atau tidak, menerima bantuan melalui alat komunikasi, memakai joki, dsb) dan menggunakan metode diluar yang diperkenankan (memakai contekan: melalui catatan, smartphone, dsb) adalah terlarang dan merupakan pelanggaran akademik.

Laporan praktikum sebagaimana laporan teknis, makalah, dan buku TA termasuk dalam kategori karya ilmiah, sehingga definisi dan aturan mengenai plagiarisme berlaku. Kecurangan yang biasa dilakukan diantaranya menggunakan data dari peserta lain, menggunakan template laporan peserta lain dan hanya mengganti datanya dan melakukan parafrase isi laporan yang lain.

PERCOBAAN I

PENGENALAN MATLAB

MATLAB akan digunakan secara ekstensif pada praktikum ini. Modul ini memberikan tinjauan singkat mengenai MATLAB dan kapabilitasnya dengan penekanan pada isu pemrograman. Seluruh materi pada modul ini merupakan terjemahan bebas dari buku “*DSP First, A Multimedia Approach*” karangan James H. McClellan, Ronald W. Schafer, dan Mark A. Yoder, terbitan Prentice-Hall (1998), khususnya Appendix B dan C.

1. TUJUAN

1. Mempelajari penggunaan sistem help untuk mengetahui *commands* dan *syntax* dasar MATLAB
2. Dapat menggunakan MATLAB untuk desain filter
3. Mempelajari bagaimana menulis fungsi dan *m-file* pada MATLAB
4. Merancang pem-filter-an FIR dengan MATLAB
5. Memahami pem-filter-an lewat MATLAB secara mendalam

2. DASAR TEORI

MATLAB (Matrix Laboratory) adalah sebuah program untuk analisis dan komputasi numerik. Pada awalnya, program ini merupakan *interface* untuk koleksi rutin-rutin numerik dari proyek LINPACK dan EISPACK, namun sekarang merupakan produk komersial dari perusahaan Mathworks, Inc. MATLAB telah berkembang menjadi sebuah *environment* pemrograman yang canggih yang berisi fungsi-fungsi *built-in* untuk melakukan tugas pengolahan sinyal, aljabar linier, dan kalkulasi matematis lainnya. MATLAB juga berisi *toolbox* yang berisi fungsi-fungsi tambahan untuk aplikasi khusus .

MATLAB bersifat *extensible*, dalam arti bahwa seorang pengguna dapat menulis fungsi baru untuk ditambahkan pada *library* ketika fungsi-fungsi *built-in* yang tersedia tidak dapat melakukan tugas tertentu. Kemampuan pemrograman yang dibutuhkan tidak terlalu sulit bila Anda telah memiliki pengalaman dalam pemrograman bahasa lain seperti C, PASCAL, atau FORTRAN.

2.1. MATLAB HELP

MATLAB menyediakan sistem *help on-line* yang dapat diakses dengan perintah `help`. Misalnya, untuk memperoleh informasi mengenai fungsi `filter`, Anda hanya perlu mengetikkan perintah

```
>> help filter
```

Perintah di atas akan menampilkan informasi dalam bentuk teks pada layar MATLAB Anda. Sebuah perintah yang sangat berguna untuk mempelajari pemrograman MATLAB adalah `intro`, yang membahas konsep-konsep dasar tentang bahasa MATLAB. Selain itu, juga terdapat banyak program demonstrasi yang mengilustrasikan berbagai kapabilitas MATLAB, yang dapat dimulai dengan perintah `demo`.

2.2. VARIABEL DAN OPERASI MATRIKS

Tipe variabel dasar pada MATLAB adalah matriks (pada versi 5 dan ke atas, MATLAB juga menyediakan berbagai tipe data seperti pada bahasa pemrograman lainnya). Untuk mendeklarasikan sebuah variabel, Anda hanya perlu memberikan nilai tertentu padanya pada MATLAB *prompt*. Sebagai contoh,

```
>> M = [ 1  2  6; 5  2  1]
```

M =

1	2	6
5	2	1

Ketika definisi sebuah matriks melibatkan sebuah rumus yang panjang atau banyak entri, maka sebuah perintah MATLAB yang sangat panjang dapat dipecah menjadi dua (atau lebih) baris dengan cara menempatkan sebuah tanda (...) pada akhir dari sebuah baris yang ingin dilanjutkan. Sebagai contoh,

```
P = [ 1, 2, 4, 6, 8 ] + [ pi, 4, exp(1), 0, -1] + ...
```

```
    [ cos(0.1*pi), sin(pi/3), tan(3), atan(2), sqrt(pi) ];
```

Ketika sebuah ekspresi perintah atau pernyataan diakhiri dengan tanda *semicolon* (;), maka hasilnya tidak akan ditampilkan di layar. Hal ini sangat membantu ketika Anda bekerja dengan matriks dengan ukuran yang sangat besar.

Ukuran dari sebuah matriks dapat diketahui dengan operator `size`:

```
>> Msize = size(M)
```

Msize =

2	3
---	---

Oleh karena itu, kita tidak perlu menggunakan variabel khusus untuk melacak jumlah baris dan kolom suatu matriks. Ada dua jenis variabel matriks pada MATLAB, yakni skalar (*scalars*) dan vektor (*vectors*). Sebuah skalar adalah sebuah matriks yang hanya berisi satu elemen, jadi berukuran 1 x 1. Sebuah vektor adalah sebuah matriks yang hanya berisi satu baris atau kolom.

Elemen individu dari sebuah variabel matriks dapat diakses dengan memberikan indeks baris dan kolom, sebagai contoh

```
>> M13 = M(1,3)

M13 =

     6
```

Submatriks juga dapat diakses dengan cara yang mirip dengan menggunakan operator *colon* (:) seperti yang dijelaskan pada sesi berikut.

2.2.1. OPERATOR COLON (:)

Operator *colon* (:) sangat berguna untuk membuat *index arrays*. Gunakan perintah *help colon* untuk mengetahui deskripsi detail tentang kapabilitasnya.

Notasi *colon* didasarkan pada ide bahwa sebuah selang indeks dapat dihasilkan dengan memberikan sebuah nilai awal, interval, dan sebuah nilai akhir. Karena itu, sebuah vektor yang terpartisi secara teratur dapat diperoleh dengan perintah

iii = nilai awal : interval : nilai akhir

Tanpa parameter interval, nilai *default*-nya adalah 1. Metode perhitungan ini mirip dengan notasi *loop DO* pada FORTRAN, namun metode pada MATLAB selangkah lebih maju dengan cara menggabungkannya dengan pengindeksan matriks. Untuk sebuah matriks A 9 x 8, A(2,3) adalah elemen skalar yang berada pada baris kedua dan kolom ketiga dari matriks A. Jadi sebuah submatriks 4 x 3 dapat diekstrak dengan perintah A(2:5,1:3). Tanda colon juga berfungsi sebagai sebuah *wild card*, misalnya, A(2,:) adalah baris kedua matriks A.

Pengindeksan mundur akan membalikkan sebuah vektor, misalnya X(9:-1:1) untuk sebuah vektor yang berisi 9 buah elemen. Kadang-kadang, Anda juga memerlukan sebuah daftar yang berisi semua nilai elemen pada matriks, jadi A(:) memberikan sebuah vektor kolom 72 x 1, yang merupakan hasil *concatenation* elemen-elemen setiap kolom matrik A. Ini merupakan contoh *reshaping* matriks. Teknik *reshaping* yang lebih umum dapat dilakukan dengan fungsi *reshape*(A,M,N). Sebagai contoh, matriks A 9 x 8 dapat di-*reshape* menjadi sebuah matriks 12 x 6 dengan Anew = *reshape*(A,12,6).

2.2.2. OPERASI MATRIKS DAN ARRAY

Operasi *default* pada MATLAB adalah operasi matriks. Jadi A*B berarti perkalian matriks, yang akan dibahas pada bagian berikut.

2.2.2a. Tinjauan Perkalian Matriks

Operasi perkalian matriks AB hanya dapat dilakukan bila kedua matriks tersebut memiliki dimensi yang kompatibel, yakni jumlah kolom matriks A harus sama dengan jumlah baris matriks B. Sebagai contoh, sebuah matriks 5 x 8 dapat mengalikan sebuah matriks 8 x 3 untuk menghasilkan sebuah matriks AB 5 x 3. Secara umum, bila A adalah m x n, maka B haruslah n x p, dan hasil perkalian AB akan memiliki dimensi m x p. Umumnya perkalian matriks tidak bersifat komutatif, yakni $AB \neq BA$. Bila $p \neq m$, maka perkalian AB tidak terdefinisi.

Beberapa kasus khusus untuk perkalian matriks adalah *outer product* dan *inner product*. Pada *outer product*, sebuah vektor kolom mengalikan sebuah vektor baris untuk menghasilkan sebuah matriks. Bila kita membiarkan semua elemen salah satu vektor tersebut berupa '1', maka kita akan memperoleh hasil yang berulang.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

Untuk *inner product*, sebuah vektor baris mengalikan sebuah vektor kolom, jadi hasilnya berupa skalar. Bila kita membiarkan semua elemen salah satu vektor tersebut berupa '1', maka kita akan memperoleh penjumlahan semua elemen vektor lainnya.

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = a_1 + a_2 + a_3 + a_4$$

2.2.2b. Operasi pointwise array

Bila kita ingin melakukan perkalian *pointwise*, ada beberapa kebingungan yang bisa muncul. Pada kasus *pointwise*, kita ingin mengalikan matriks secara elemen per elemen, jadi mereka harus memiliki dimensi yang sama. Sebagai contoh, dua matriks 5 x 8 dapat dikalikan secara *pointwise*, walaupun keduanya tidak bisa melakukan perkalian matriks biasa. Untuk melakukan perkalian *pointwise* pada MATLAB, kita menggunakan operator "*point-star*" $A .* B$. Misalnya bila A dan B keduanya adalah matriks 3 x 2 maka

$$C = A .* B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} .* \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11} * b_{11} & a_{12} * b_{12} \\ a_{21} * b_{21} & a_{22} * b_{22} \\ a_{31} * b_{31} & a_{32} * b_{32} \end{bmatrix}$$

Untuk selanjutnya, perkalian semacam ini kita sebut dengan istilah perkalian *array*. Perhatikan bahwa perkalian *array* bersifat komutatif karena kita akan memperoleh hasil yang sama bila kita menghitung $D = B .* A$.

Dalam MATLAB, bila sebuah “titik” digunakan dengan operator aritmetik, maka ia akan mengubah definisi operator tersebut ke operasi *pointwise*. Jadi operator ./ berarti pembagian *pointwise*, .^ berarti pemangkatan *pointwise*. Misalnya, $xx = (0.9).^{(0:49)}$ akan menghasilkan suatu vector yang nilainya sama dengan $(0,9)^n$ untuk $n = 0, 1, 2, \dots, 49$.

2.2.2c. Operasi concatenation array

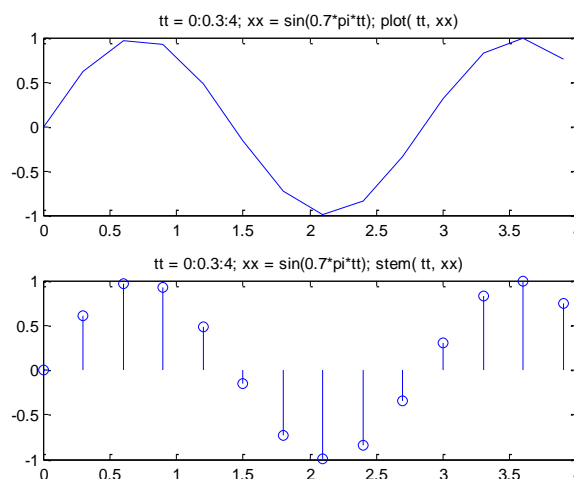
Operasi ini digunakan untuk menempelkan dua atau lebih array dengan syarat syarat tertentu sesuai dengan operasi concatenation yang diinginkan. Dalam MATLAB terdapat dua buah fungsi yang dapat digunakan untuk melakukan proses concatenation (penempelan) arrays. Fungsi tersebut adalah vertcat dan horzcat. Penjelasan lanjut dapat dilihat pada help MATLAB untuk fungsi-fungsi tersebut.

2.3. PLOT DAN GRAFIK

MATLAB dapat menghasilkan plot dua dimensi x-y dan plot tiga dimensi, menayangkan citra, dan bahkan membuat dan memutar video. Dua fungsi yang sering digunakan pada praktikum ini adalah plot dan stem. Untuk memanggil fungsi ini, umumnya kita membutuhkan dua vektor (satu vektor juga bisa, namun untuk definisi yang berbeda, gunakan perintah help untuk melihat informasi yang lebih lengkap), untuk sumbu x dan sumbu y. Pemanggilan fungsi `plot(x,y)` akan menghasilkan suatu plot yang terkoneksi dengan garis lurus untuk setiap dua titik

$\{ (x(1),y(1)), (x(2),y(2)), (x(3),y(3)), \dots, (x(N),y(N)) \}$ seperti yang ditunjukkan pada gambar PA.1.

Pemanggilan fungsi `stem(x,y)` akan menghasilkan presentasi seperti yang ditunjukkan pada Gambar 1. 1 PA.1



Gambar 1. 1 PA.1

MATLAB memiliki banyak opsi *plotting* yang dapat dipelajari dengan `help plotxy`, `help plotxyz`, dan `help graphics` (versi 4) atau `help graph2d`, `help graph3d`, dan `help specgraph` (versi 5).

2.3.1. Figure Windows

Ketika MATLAB membuat sebuah plot, MATLAB menulis grafik tersebut ke *figure windows*. Anda bisa membuka beberapa *figure windows* namun setiap saat hanya satu *window* yang aktif. Setiap perintah plot pada *command window* akan mengalihkan keluarannya ke *window* yang aktif. Perintah `figure(n)` akan menampilkan sebuah *figure window* yang baru yang ditandai dengan bilangan *n*, atau membuatnya aktif kembali bila telah ada sebelumnya. Pengendalian terhadap berbagai atribut *window* (ukuran, lokasi, warna) juga mungkin dilakukan dengan perintah `figure`, yang melakukan inisialisasi terhadap *window plot*.

2.3.2. Mem-Plot beberapa grafik

Anda juga dapat membuat beberapa grafik/plot pada satu *window* dengan menggunakan fungsi `subplot`. Fungsi ini tidak melakukan proses *plotting*, namun hanya membagi *window* menjadi beberapa segmen. Sebagai contoh, perintah `subplot(3,2,3)` akan membagi *figure window* menjadi tiga baris dan dua kolom (jadi terdapat enam segmen) dan mengarahkan plot berikutnya ke segmen kiri baris kedua. Grafik pada PA.1 diperoleh dengan perintah `subplot(2,1,1)` dan `subplot(2,1,2)`.

2.4. KONSTRUK PEMROGRAMAN

MATLAB mendukung paradigma pemrograman fungsional, di mana Anda dapat menyusun fungsi-fungsi secara *nested*. Perhatikan persamaan di bawah

$$\sum_{n=1}^L \log(|x_n|)$$

yang dapat diimplementasikan dengan hanya menggunakan satu baris kode MATLAB, yakni

```
sum( log( abs(x) ) )
```

di mana *x* adalah sebuah vektor yang berisi elemen-elemen x_n . Contoh ini mengilustrasikan MATLAB dalam bentuk yang paling efisien, di mana fungsi-fungsi individu dikombinasikan untuk menghasilkan keluaran. Penulisan kode-kode MATLAB yang efisien memerlukan gaya pemrograman yang menghasilkan fungsi-fungsi kecil yang divektorisasi. *Loop-loop* harus dihindari. Cara utama untuk menghindari *loop* adalah memanggil fungsi-fungsi *toolbox* sebanyak/sesering mungkin.

2.4.1. Fungsi-fungsi built-in MATLAB

Banyak fungsi-fungsi MATLAB yang dapat beroperasi pada skalar sama mudahnya dengan operasi pada *array*. Sebagai contoh, bila x adalah sebuah *array*, maka $\cos(x)$ mengembalikan sebuah *array* dengan ukuran yang sama seandainya x berisi kosinus dari setiap elemen x .

$$\cos(x) = \begin{bmatrix} \cos(x_{1,1}) & \cos(x_{1,2}) & \cdots & \cos(x_{1,n}) \\ \cos(x_{2,1}) & \cos(x_{2,2}) & \cdots & \cos(x_{2,n}) \\ \vdots & \vdots & \vdots & \vdots \\ \cos(x_{m,1}) & \cos(x_{m,2}) & \cdots & \cos(x_{m,n}) \end{bmatrix}$$

Perhatikan bahwa tidak ada *loop* yang diperlukan, meskipun $\cos(x)$ melakukan operasi kosinus pada setiap elemen *array*. Kebanyakan fungsi *transcendental* mengikuti aturan *pointwise* ini. Pada beberapa kasus khusus, adalah sangat penting untuk membedakan eksponensial matriks (\expm) dengan eksponensial *pointwise* (\exp):

$$\exp(A) = \begin{bmatrix} \exp(a_{1,1}) & \exp(a_{1,2}) & \cdots & \exp(a_{1,n}) \\ \exp(a_{2,1}) & \exp(a_{2,2}) & \cdots & \exp(a_{2,n}) \\ \vdots & \vdots & \vdots & \vdots \\ \exp(a_{m,1}) & \exp(a_{m,2}) & \cdots & \exp(a_{m,n}) \end{bmatrix}$$

2.4.2. Aliran Program (*Program Flow*)

Aliran program dapat dikendalikan pada MATLAB menggunakan pernyataan *if*, *loopwhile*, dan *loopfor*. Pada MATLAB versi 5, terdapat juga pernyataan *switch*. Hal ini mirip dengan bahasa-bahasa tingkat tinggi seperti C++ atau PASCAL. Deskripsi dan contoh dari setiap konstruk program tersebut dapat dilihat dengan menggunakan perintah *help*.

2.5. MATLAB SCRIPTS

Setiap perintah/ Pernyataan yang dapat dimasukkan pada *window prompt* dapat disimpan pada sebuah *file* teks dan dieksekusi sebagai *script*. *File* teks tersebut dapat dibuat dengan menggunakan sembarang *editor* ASCII seperti program Notepad atau pada editor teks MATLAB. Ekstensi *file* harus berupa *.m* dan *script* tersebut dieksekusi pada MATLAB dengan hanya mengetikkan nama *file* (dengan atau tanpa ekstensi). Program-program tersebut umumnya dikenal dengan istilah *m-file*. Berikut merupakan contoh sebuah *m-file*:

```

tt = 0:0.3:4;

xx = sin(0.7*pi*tt);

subplot(2,1,1)

plot( tt, xx)

title('tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')

subplot(2,1,2)

stem( tt, xx)

title('“tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)’)

```

Bila perintah-perintah ini disimpan dengan *file* bernama plotstem.m maka pengetikan plotstem pada *command prompt* akan menjalankan *file* tersebut, dan kedelapan baris perintah akan dieksekusi sama halnya bila mereka diketikkan baris per baris pada *command prompt*. Hasilnya adalah dua buah plot seperti yang tampak pada gambar A.1.

Setiap perintah/ Pernyataan yang dapat dimasukkan pada *window prompt* dapat disimpan pada sebuah *file* teks dan dieksekusi sebagai *script*. *File* teks tersebut dapat dibuat dengan menggunakan sembarang *editor* ASCII seperti program Notepad atau pada editor teks MATLAB. Ekstensi *file* harus berupa .m dan *script* tersebut dieksekusi pada MATLAB dengan hanya mengetikkan nama *file* (dengan atau tanpa ekstensi). Program-program tersebut umumnya dikenal dengan istilah *m-file*. Berikut merupakan contoh sebuah *m-file*:

```

tt = 0:0.3:4;

xx = sin(0.7*pi*tt);

subplot(2,1,1)

plot( tt, xx)

title('tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')

subplot(2,1,2)

stem( tt, xx)

title('“tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)’)

```

Bila perintah-perintah ini disimpan dengan *file* bernama plotstem.m maka pengetikan plotstem pada *command prompt* akan menjalankan *file* tersebut, dan kedelapan baris perintah akan dieksekusi sama halnya bila mereka diketikkan baris per baris pada *command prompt*. Hasilnya adalah dua buah plot seperti yang tampak pada gambar A.1.

2.6. MENULIS FUNGSI MATLAB

Anda dapat menulis fungsi sendiri dan kemudian ditambahkan pada *environment* MATLAB. Fungsi-fungsi ini merupakan jenis lain dari *m-file*, dan dibuat sebagai sebuah *file* ASCII menggunakan *editor* teks. Kata pertama pada *m-file* tersebut haruslah *keyword* *function* untuk memberitahukan MATLAB bahwa *file* tersebut diperlakukan sebagai sebuah fungsi dengan argumen. Pada baris yang sama juga berisi *calling template* yang menyatakan argumen input dan output dari fungsi. Nama *file* untuk *m-file* tersebut haruslah berekstensi *.m* dan nama fungsi tersebut akan menjadi nama dari perintah baru pada MATLAB. Sebagai contoh, perhatikan *file* berikut, yang mengekstrak *L* buah elemen terakhir dari sebuah vektor

```
function y = foo( x, L )

%FOO mengambil L buah titik terakhir dari x

%  penggunaan:

%          y = foo( x, L )

%  di mana:

%          x = vektor input
%          L = jumlah titik yang ingin diambil
%          y = vektor output

N = length(x);

If (L > N)

error('vektor input terlalu pendek')

end

y = x(( N-L+1):N );
```

Bila *file* ini disimpan dengan nama *foo.m*, operasi ini dapat dipanggil dari MATLAB *command line* dengan cara mengetikkan

```
aa = foo( (1:2:37), 7 );
```

Outputnya akan berupa tujuh elemen terakhir dari vektor (1:2:37), yakni

```
aa = [ 25 27 29 31 33 35 37 ]
```

2.6.1. Membuat sebuah fungsi *clip*

Kebanyakan fungsi dapat ditulis menurut format standar. Perhatikan sebuah **fungsi** *m-fileclip* yang mengambil dua buah argumen (sebuah vektor sinyal dan nilai skalar pembatas/*threshold*) dan mengembalikan sebuah vektor sinyal keluaran. Anda dapat menggunakan sebuah *editor* untuk menghasilkan *file* ASCII dari *clip.m* yang berisi pernyataan-pernyataan berikut.

Komentar-komentar ini akan ditampilkan bila Anda mengetikkan help clip

Langkah pertama adalah mencari dimensi matriks x

Input dapat berupa vektor kolom atau baris

Karena x adalah variabel lokal, kita dapat mengubahnya tanpa mempengaruhi workspace

Menghasilkan vektor output

```
function y = clip( x, Limit )
%CLIP mensaturasikan magnitud of x[n] pada
Limit
% ketika | x[n] | > Limit, buat | x[n] | = Limit
%
% penggunaan: y = clip( x, Limit )
%
% x          – vektor sinyal input
% Limit      – nilai pembatas
% y          – vektor output setelah clipping
[nrows ncols] = size(x);
if(ncols ~= 1 & nrows ~=1 ) %--bukan keduanya
error( 'CLIP: input bukan sebuah vektor' )
end
Lx = max( [nrows ncols] ); %--Panjang
for n = 1: Lx %--Loop seluruh vektor
if( abs( x( n ) ) > Limit )
x( n ) = sign( x ( n ) ) * Limit; %--saturasi
end %--mempertahankan tanda x(n)
end
y = x; %-- kopi ke vektor output
```

Kita dapat memecah *m-fileclip.m* menjadi empat bagian:

1. Definisi input-output: Setiap *m-file* fungsi harus diawali kata *function*. Informasi yang mengikuti kata *function* pada baris yang sama merupakan deklarasi bagaimana fungsi

tersebut dipanggil dan argumen apa yang dibutuhkan. Nama **fungsi harus sama dengan nama *m-file***; bila terjadi konflik maka nama *m-file*-lah yang dikenali pada MATLAB *command environment*.

Argumen-argumen input ditulis di dalam tanda kurung setelah nama fungsi. Setiap input adalah sebuah matriks. Argumen output (yang juga berupa matriks) berada di sebelah kiri tanda =. Argumen output ganda juga dimungkinkan dengan menggunakan tanda kurung siku [...], misalnya perintah `size(x)` mengembalikan jumlah baris dan kolom kedua variabel yang berbeda. Akhirnya, perhatikan bahwa tidak ada *command* eksplisit untuk mengembalikan sebuah output. MATLAB mengembalikan semua nilai yang terkandung pada matriks output ketika perhitungannya selesai dilakukan. Baris terakhir pada `clip.m` mengembalikan nilai `y` yang ter-*clip* ke `y`, jadi vektor yang ter-*clip* dikembalikan. MATLAB tidak memiliki perintah `return`, tapi ia langsung mengakhiri fungsi tersebut tanpa memerlukan argumen.

Perbedaan esensial antara *m-file* fungsi dan *m-filescripts* adalah *dummy variables* vs *permanent variables*. MATLAB menggunakan metode “*call by value*” sehingga fungsi tersebut membuat kopian lokal dari argumennya. Variabel-variabel lokal tersebut akan segera hilang saat fungsi tersebut selesai. Sebagai contoh, pernyataan berikut menghasilkan sebuah vektor `wclipped` dari vektor input `ww`.

```
>> wwclipped = clip(ww, 0.9999);
```

Array `ww` dan `wwclipped` adalah *permanent variables* pada MATLAB *workspace*. Array sementara yang dihasilkan di dalam `clip.m` (yaitu `nrows`, `ncols`, `Lx` dan `n`) hanya ada saat fungsi tersebut dijalankan. Mereka akan segera hilang ketika fungsi selesai dijalankan. Lebih lanjut, nama-nama variabel tersebut bersifat lokal (hanya bagi `clip.m`) sehingga nama variabel `x` juga dapat digunakan pada *workspace* sebagai *permanent variables*. Tentunya, ide-ide ini bukanlah sesuatu yang asing bagi Anda yang pernah mengenal bahasa seperti C, FORTRAN, atau PASCAL.

2. Dokumentasi. Setiap baris yang diawali dengan tanda % adalah baris komentar. Jadi, Anda dapat mengetikkan `help clip` dan komentar-komentar dari *m-file* Anda akan ditampilkan pada layar Anda. Format tersebut mengikuti “aturan” : nama fungsi, prosedur/ urutan pemanggilan, penjelasan singkat, dan definisi argumen input dan output.
3. Pengujian ukuran (*size*) dan kesalahan (*error*): Fungsi tersebut harus menguji dimensi dari matriks atau vektor yang akan dioperasikan. Informasi tersebut tidak perlu disampaikan sebagai argumen input yang terpisah, namun dapat diekstraks dengan fungsi `size`. Pada kasus fungsi `clip.m`, kita kita membatasi bahwa operasi hanya dapat dilakukan pada vektor, baik itu vektor baris $1 \times L$ atau vektor kolom $L \times 1$. Artinya, salah satu variabel tersebut haruslah sama dengan 1; kita mengakhiri fungsi tersebut dengan fungsi `error`, yang akan mencetak kalimat pada *command line* tersebut dan mengakhiri fungsi.
4. Operasi fungsi yang sebenarnya: Pada kasus fungsi `clip.m`, proses *clipping* dilakukan oleh loop `for`, yang menguji besarnya nilai setiap elemen pada vektor `x` terhadap nilai

pembatas Limit. Pada kasus bilangan negatif, nilai yang di-*clip* akan diset ke $-\text{Limit}$, melalui perkalian dengan $\text{sign}(x(n))$. Ini mengasumsikan bahwa Limit dilewatkan sebagai sebuah bilangan positif, suatu fakta yang dapat diuji juga pada fase pengujian kesalahan.

Implementasi fungsi *clip.m* ini sangat tidak efisien karena adanya *loopfor*. Pada sesi A.7.1 kita akan menunjukkan bagaimana men-vektorisasi program untuk meningkatkan efisiennya.

2.6.2. Debugging MATLAB m-file

Karena MATLAB adalah sebuah *environment* yang interaktif, *debugging* dapat dilakukan dengan cara menguji variabel-variabel pada *workspace*. MATLAB versi 4 dan 5 menyediakan *debugger* simbolik yang mendukung *breakpoints*. Karena fungsi yang berbeda dapat menggunakan nama variabel yang sama, adalah sangat penting untuk melacak konteks lokal ketika menguji variabel. Beberapa perintah *debugging* yang berguna didaftarkan di sini, dan yang lainnya dapat Anda temukan di *help debug*.

dbstop digunakan untuk mengeset sebuah *breakpoint* pada sebuah *m-file*. Ia juga dapat digunakan untuk memberikan sebuah peringatan ketika sebuah kesalahan terjadi dengan mengetikkan *dbstop if error* sebelum mengeksekusi *m-file* tersebut. Hal ini memungkinkan Anda menguji variabel di dalam fungsi dan *workspace* (dengan mengetikkan *dbup*)

dbstep akan mengembalikan sebuah peringatan (*prompt*) ketika setiap baris perintah dieksekusi.

dbcont menyebabkan sebuah eksekusi program yang normal berhenti, atau bila ada kesalahan, mengembalikan status Anda ke MATLAB *command prompt*.

dbquit menyebabkan Anda keluar dari *modedebbug* dan kembali ke MATLAB *command prompt*.

keyboard dapat disisipkan ke *m-file* untuk menghentikan sementara eksekusi program, yang memberikan sebuah MATLAB *prompt* dalam bentuk *K>* untuk mengindikasikan bahwa itu bukan *command-line prompt*.

2.7. TIPS PEMROGRAMAN

Bagian ini akan memperkenalkan beberapa *tips* pemrograman yang akan meningkatkan kecepatan program MATLAB Anda. Untuk mengetahui lebih banyak tentang *tips* dan ide, perhatikan gaya-gaya penulisan pada *m-file* (*built-in function*) yang tersedia pada *toolbox* MATLAB. Sebagai contoh, ketikkan perintah

```
type angle
type conv
type trapz
```

Mempelajari gaya pemrograman orang lain merupakan cara yang efisien untuk memahami suatu bahasa pemrograman komputer. Pada petunjuk-petunjuk berikut, kita akan membahas hal-hal yang terpenting dalam penulisan kode MATLAB yang baik. Petunjuk-petunjuk ini akan menjadi sangat berguna ketika Anda semakin mengenal MATLAB.

2.7.1. Menghindari *loop*

Karena MATLAB merupakan sebuah bahasa interpretasi (*interpreted language*), maka terdapat beberapa gaya pemrograman yang sangat tidak efisien. Salah satunya adalah penggunaan *loopfor* untuk melakukan operasi sederhana terhadap sebuah matriks atau vektor. Bila dimungkinkan, Anda seharusnya mencoba mencari sebuah fungsi vektor (atau komposisi *nested* dari beberapa fungsi vektor) yang akan memberikan hasil yang sama, daripada menulis sebuah *loop*. Misalnya, bila operasi tersebut adalah penjumlahan semua elemen pada sebuah matriks, perbedaan antara memanggil fungsi *sum* dan menulis sebuah *loop* seperti pada bahasa FORTRAN dapat sangat mengejutkan; *loop* tersebut akan sangat lambat dalam menjalankan eksekusi akibat sifat *interpreted* dari MATLAB. Perhatikan tiga metode pada penjumlahan matriks berikut

Loop ganda diperlukan untuk mengindeks semua elemen matriks

```
[Nrows, Ncols] = size(x);  
xsum = 0.0;  
for m = 1: Nrows  
    for n = 1: Ncols  
        xsum = xsum + x(m,n);
```

sum menjumlahkan semua elemen setiap kolom

```
Xsum = sum( sum(x) );
```

x(:) adalah sebuah vektor yang berisi semua elemen matriks

```
Xsum = sum( x(:) );
```

Metode pertama adalah ekivalen MATLAB untuk pemrograman konvensional. Dua metode terakhir mengandalkan fungsi *built-in sum*, yang memiliki perbedaan karakteristik, tergantung dari apakah argumennya berupa sebuah matriks atau vektor (disebut operator *overloading*). Ketika beraksi pada sebuah matriks, *sum* mengembalikan sebuah vektor baris yang berisi penjumlahan kolom, dan ketika beraksi pada sebuah vektor baris (atau kolom), penjumlahan

tersebut adalah skalar. Pada metode ketiga, yang merupakan metode terefisien, matriks x dikonversikan ke vektor kolom dengan operator *colon* (:). Dan selanjutnya Anda hanya perlu memanggil fungsi `sum` sekali saja.

2.7.2. Pengulangan Baris atau Kolom

Sering kali kita perlu membentuk sebuah matriks dari sebuah vektor dengan cara mereplika vektor tersebut pada baris atau kolom matriks. Bila matriks tersebut memiliki nilai yang sama maka fungsi seperti `ones(M,N)` dan `zeros(M,N)` dapat digunakan. Tetapi untuk mereplika suatu vektor kolom x untuk menghasilkan sebuah matriks yang memiliki kolom-kolom yang identik, sebuah *loop* dapat dihindari dengan menggunakan operasi perkalian *outer-product matrix* yang telah dibahas pada sesi A2.2. Fragmen kode MATLAB berikut akan melakukan tugas tersebut untuk sebelas kolom

```
x = (12:-2:0)' ;           % tanda ' menunjukkan conjugatetranspose  
X = x * ones(1,11)
```

Bila x adalah sebuah vektor kolom L , maka matriks X yang dihasilkan oleh *outer-product* tersebut adalah $L \times 11$. Pada contoh ini, $L = 7$. Perhatikan bahwa penulisan pada MATLAB bersifat *case-sensitive*, jadi variabel x dan X adalah berbeda.

2.7.3. Vektorisasi operasi logika

Adalah mungkin untuk menvektorisasi program yang berisi pernyataan kondisional `if`, `else`. Fungsi `clip` pada A.2 merupakan contoh yang baik untuk mendemonstrasikan jenis vektorisasi ini. *Loopfor* pada fungsi ini berisi sebuah pengujian logikal dan tidak tampak sebagai kandidat yang cocok untuk operasi vektor. Namun, operator logikal dan relasional pada MATLAB, seperti lebih besar dari, dapat diaplikasikan pada matriks. Sebagai contoh, pengujian “lebih besar dari” pada matriks 3×3 akan mengembalikan sebuah matriks 3×3 yang hanya berisi 1 dan 0.

```
>> x = [ 1  2 -3;  3 -2  1;  4  0 -1]    % membuat matriks pengujian  
  
      1   2  -3  
      3  -2   1  
      4   0  -1
```



```
>> mx = x > 0 % menguji kondisi lebih besar
```

```
mx = [ 1  1  0  
      1  0  1  
      1  0  0]
```

```
>> y = mx.* x % perkalian pointwise dengan matriks masking mx
```

```
y = [ 1  2  0  
     3  0  1  
     4  0  0]
```

Elemen nol menunjukkan kondisi yang salah, dan elemen 1 menunjukkan kondisi yang benar. Jadi, ketika kita melakukan perkalian pointwise antara x dan matriks masking mx , kita memperoleh suatu matriks di mana semua elemen negatif diset ke nol. Perhatikan bahwa dua pernyataan terakhir memproses keseluruhan matriks tanpa menggunakan loop for. Karena saturasi yang dilakukan pada clip.m mensyaratkan kita mengubah nilai-nilai yang besar pada x , kita dapat mengimplementasikan keseluruhan loop for dengan tiga perkalian array. Hal ini mengantarkan kita ke operator saturasi tervektorisasi yang dapat bekerja pada matriks, maupun vektor sekaligus:

```
y = Limit*(x > Limit) - Limit*(x < -Limit) + x.*(abs(x) <= Limit);
```

Tiga matriks masking diperlukan untuk mewakili masing-masing kasus saturasi positif, saturasi negatif, dan tanpa aksi. Operasi penjumlahan berkoresponden dengan operator logika OR pada semua kasus ini. Jumlah operasi aritmetik yang harus dilakukan pada pernyataan ini adalah $3N$ perkalian dan $2N$ penjumlahan, di mana N adalah jumlah elemen pada x . Tampak proses ini memerlukan lebih banyak pekerjaan daripada loop pada clip.m bila kita hanya memperhitungkan operasi aritmetik. Namun, “harga” yang harus dibayar untuk interpretasi kode sangat mahal. Pernyataan tervektorisasi (vectorized) ini hanya diinterpretasikan sekali saja, sementara tiga pernyataan di dalam loop for harus direinterpretasikan sebanyak N kali. Bila Anda menggunakan etime untuk mencatat waktu kedua implementasi tersebut, maka versi vektorisasi akan jauh lebih cepat untuk vektor-vektor panjang.

2.7.4. Membuat sinyal *impulse*

Contoh sederhana lainnya diberikan oleh trik berikut untuk menghasilkan sebuah vektor sinyal *impulse*:

```
nn = [-10 : 25];  
impulse = (nn == 0);
```

Hasilnya dapat diplot dengan menggunakan `stem(nn, impulse)`. Fragmen kode ini sangat baik untuk melukiskan esensi dari rumus matematika yang mendefinisikan bahwa sinyal *impulse* hanya ada untuk $n = 0$.

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

2.7.5. Fungsi Find

Sebuah alternatif lain untuk *masking* adalah menggunakan fungsi `find`. Ini tidak berarti bahwa pendekatan ini lebih efisien, namun hanya berupa pendekatan lain. Fungsi `find` akan menentukan daftar indeks pada sebuah vektor di mana kondisinya adalah benar (persyaratan dipenuhi). Misalnya, `find(x > Limit)` akan mengembalikan daftar indeks di mana vektor tersebut lebih besar dari nilai `Limit`. Jadi, kita dapat melakukan saturasi sebagai berikut:

```
y = x
jkl = find(y > Limit)
y(jkl) = Limit*ones(size(jkl));
jkl = find(y < -Limit);
y(jkl) = -Limit*ones(size(jkl));
```

Fungsi `ones` diperlukan untuk menghasilkan sebuah vektor pada sisi kanan yang memiliki ukuran yang sama dengan jumlah elemen pada `jkl`. Pada versi 5, hal ini tidak diperlukan karena sebuah skalar yang di-*assign* ke vektor berarti di-*assign* pula untuk setiap elemen vektor.

2.7.6. Vektorisasi

Upaya “menghindari *loopfor*” tidak selalu mudah untuk diikuti karena hal ini berarti algoritma harus disusun kembali dalam bentuk vektor. Bila notasi matriks-vektor dimasukkan dalam program MATLAB, maka kode tersebut akan berjalan jauh lebih cepat. Bahkan, *loop* untuk pengujian logika dapat divektorisasi bila *mask* diciptakan untuk semua kondisi. Jadi sebuah tujuan yang lebih masuk akal adalah

Eliminasi semua *loopfor*

2.7.7. Gaya Pemrograman

Bila ada sebuah kalimat yang merangkum gaya pemrograman yang baik, maka bunyinya mungkin seperti berikut

Usahakan fungsi Anda pendek dan nama variabelnya panjang

Kalimat di atas sangat tepat untuk MATLAB. Setiap fungsi seharusnya memiliki satu tujuan khusus. Hal ini akan menghasilkan modul-modul yang singkat dan sederhana yang dapat di-link bersama dengan komposisi fungsional untuk menghasilkan operasi yang kompleks. Hindari kecenderungan untuk membuat fungsi super dengan banyak opsi dan keluaran.

MATLAB mendukung nama variabel hingga 32 variabel. Gunakan fitur ini untuk memberikan nama variabel yang deskriptif. Dengan cara ini, jumlah komentar-komentar yang “mengotori” kode Anda dapat dikurangi secara drastis. Komentar-komentar seharusnya hanya dibatasi untuk informasi help dan trik-trik yang digunakan pada kode tersebut.

3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN

Pelajari penggunaan perintah `fir1`, `fir2`, `filter`, `freqz` dan perintah-perintah lain yang berhubungan dengan topik praktikum pada Matlab.

Tugas pendahuluan :

1. Jelaskan keuntungan penggunaan filter digital secara umum!
2. Apa yang dimaksud dengan filter FIR?
3. Jelaskan fungsi `fir1` dan `fir2` yang terdapat pada Matlab! Apa perbedaan antara keduanya?
4. Apa yang dimaksud dengan koefisien filter (yang diperoleh dari fungsi desain filter dari Matlab)?

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software Matlab

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software Matlab sudah terinstal dalam komputer.

4.2.1. Percobaan membuat sinyal input filter berupa superposisi beberapa sinyal sinusoidal dengan frekuensi berbeda.

1. Pada Matlab, representasikan sinyal dalam vektor (matriks $1 \times N$, N panjang vektor). Kita akan merepresentasikan sumbu waktu dimana untuk $0 < t < 2\pi$, kita beri panjang vektor 100 (100 sampel) dengan perintah `>>i=1:100;`
2. Buat 3 sinyal sinusoidal pada frekuensi pencuplikan $f_s=16000$ Hz untuk masing-masing frekuensi sinyal $f_1=200$ Hz, $f_2=1000$ Hz, $f_3=5000$ Hz. Ketikan :
`>>sin1=sin(2*pi*i*f1/fs);sin2=sin(2*pi*i*f2/fs);sin3=sin(2*pi*i*f3/fs);`
3. Jumlahkan ketiga sinyal tersebut menjadi satu sinyal sinusoidal rusak dengan perintah
`>>sintot=(sin1+sin2+sin3)/3;`
4. Coba plot gambarnya dengan perintah `>>plot(sintot);`
5. Lihat juga respon frekuensinya dengan perintah `freqz`.
6. Kini Anda telah memiliki sinyal input untuk *filter* yang akan kita rancang

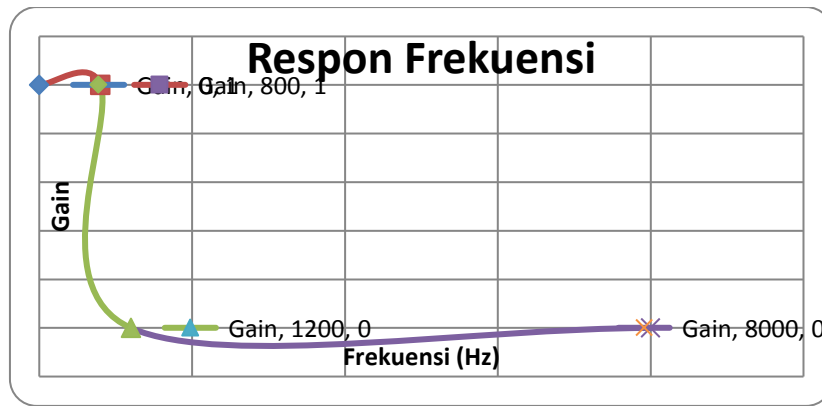
4.2.2. Percobaan desain dan simulasi filter FIR 1

Kita akan coba beberapa *filter* FIR dengan spesifikasi berikut (frekuensi boleh diubah-ubah)

- Filter FIR *low-pass* orde 32 dengan frekuensi *cut-off* 800 Hz
 - Filter FIR *band-pass* orde 32 dengan frekuensi *pass* 1000 – 3000 Hz
 - Filter FIR *high-pass* orde 32 dengan frekuensi *cut-off* 6000Hz
1. Rancang ketiga *filter* di atas, cari koefisien *filter*-nya dengan perintah yang sesuai (`fir1`). Catat masing-masing koefisien *filter*.
 2. Lihat frekuensi respon masing-masing *filter* dengan perintah `freqz`. Gambarkan hasilnya. Analisa pada frekuensi cut off.

4.2.3. Percobaan desain dan simulasi filter FIR 2

Kita akan coba mendesain *filter* FIR dengan metoda frekuensi sampling dengan respon frekuensi seperti pada Gambar 1. 2 Respon Frekuensi Filter



Gambar 1. 2 Respon Frekuensi Filter

1. Rancang *filter* di atas, cari koefisien *filter*-nya dengan perintah yang sesuai (`fir2`). Catat koefisien *filter*. Gunakan orde 16.
2. Lihat frekuensi respon *filter* dengan perintah `freqz`. Gambarkan hasilnya. Lakukan untuk orde lebih besar misalnya 128. Bandingkan dengan hasil sebelumnya.

4.2.4. Percobaan membuat m-file untuk melakukan pem-filter-an FIR

Pada bagian ini anda diminta untuk membuat m-file untuk melakukan pemfilteran FIR saja, untuk m-file anda tidak diperbolehkan memanggil fungsi internal MATLAB. Bandingkan hasilnya dengan percobaan dengan menggunakan perintah *filter* dari MATLAB. (Untuk kelancaran praktikum source code bisa dipersiapkan sebelum praktikum)

Catatan:

- Pada laporan m-file disertakan, dan berikan penjelasan algoritma yang anda gunakan untuk melakukan pem-filter-an FIR.

HINT: Koefisien filter FIR adalah merupakan respon impuls dari filter.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikum dan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai**.

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

CONTOH FORMAT LAPORAN

1.1. PERANCANGAN FILTER DENGAN MATLAB

Plot Sinyal Input

<i>Sinyal Input</i>	<i>Respon Frekuensi</i>
---------------------	-------------------------

1.2. DESAIN FILTER FIR

<i>Low-pass orde 31 f cut-off 800 Hz</i>	<i>Koefisien Filter</i>
<i>Respon Frekuensi Filter</i>	
<i>Band-pass orde 31 f cut-off 1000-3000 Hz</i>	<i>Koefisien Filter</i>
<i>Respon Frekuensi Filter</i>	
<i>High-pass orde 31 f cut-off 6000 Hz</i>	<i>Koefisien Filter</i>
<i>Respon Frekuensi Filter</i>	

PERCOBAAN II

SIMULASI FILTER FIR REALTIME

Dev-C++ akan digunakan secara ekstensif pada praktikum ini. Modul ini memberikan tinjauan mengenai filter FIR realtime dapat diimplementasikan. Pada modul ini, praktikan akan membuat dan melakukan simulasi untuk implementasi filter realtime dengan menggunakan Dev-C++. Pada modul ini juga dikenalkan isu numerik yang dihadapi ketika menggunakan prosesor DSP fixed point.

1. TUJUAN

1. Mempelajari bagaimana mengimplementasikan dan melakukan simulasi filter FIR realtime menggunakan Dev-C++
2. Mengetahui model bilangan fraksional untuk prosesor DSP fixed point.
3. Mengimplementasikan dan melakukan simulasi filter FIR realtime dengan bilangan fraksional menggunakan Dev-C++

2. DASAR TEORI

2.1. FILTER FIR REALTIME

Sistem realtime disebut juga dengan sistem waktu nyata. Sistem realtime harus dapat memberikan respon yang tepat dalam batasan waktu yang ditentukan. Realtime adalah metode realisasi, sehingga setiap tugas spesifik dapat dikerjakan pada waktu spesifik dengan siklus clock sistem.

Pada modul sebelumnya kita menggunakan filter FIR offline (non realtime). Terdapat perbedaan di antara algoritma sistem realtime dan algoritma sistem offline. Ide dasar dari algoritma realtime adalah hanya input saat ini dan yang telah lewat yang tersedia. Kita tidak memiliki akses kepada input yang akan datang berikutnya. Pada modul sebelumnya fungsi filter yang digunakan memiliki akses kepada keseluruhan sampel pada sinyal input.

Algoritma filter realtime diperlukan untuk implementasi filter realtime dengan hardware DSP. Untuk keperluan ini maka diperlukan simulasi algoritma filter realtime-like dan dibandingkan hasilnya dengan algoritma filter offline.

Sistem DSP adalah sistem waktu diskrit yang mengambil sampel sinyal input dan menghasilkan sampel sinyal output dengan frekuensi sampling tertentu yang konstan. Artinya pada sistem DSP realtime, pada rentang waktu antar sampling, haruslah dapat dilakukan proses

perhitungan sampel output yang merupakan hasil pengolahan sinyal digital (bisa berupa filter). Artinya proses komputasi pengolahan sinyal digital untuk tiap sampel haruslah dapat diselesaikan sebelum sampel berikutnya diambil.

Karakteristik simulasi realtime-like adalah sebagai berikut:

1. Loop untuk kesesuaian dengan kelakuan clock
2. Akses hanya kepada input yang telah ada pada array input
3. Update output setiap siklus loop

2.2. ISU NUMERIK

Kebanyakan prosesor DSP merupakan prosesor fixed point. Prosesor floating point jauh lebih rumit dari prosesor fixed point sehingga membutuhkan harga yang lebih mahal. Untuk membuat sistem yang cost effective maka prosesor DSP direalisasikan dengan prosesor fixed point. Prosesor fixed point secara natural set intruksi dasar matematikanya hanya mendukung operasi bilangan bulat (integer)

Memahami isu numerik adalah sangat penting untuk memperoleh performansi terbaik dari sebuah prosesor fixed-point. Masalahnya adalah kebanyakan sinyal dan juga koefisien bernilai bilangan real. Sehingga operasi pengolahan sinyal merupakan operasi perkalian dan penjumlahan bilangan real. Namun DSP yang kita gunakan merupakan prosesor fixed point.

Dari percobaan sebelumnya Anda telah memperoleh koefisien *filter* untuk melakukan pem-*filter*-an. Kita akan menggunakannya sebagai koefisien *filter* yang akan diterapkan di BF561. Nantinya kita akan mencoba program *filter* dalam bahasa pemograman C ataupun Assembly, jadi kita dapat menuliskan koefisien tersebut di program kita, seperti berikut (contoh)

```
int koef={a1,a2,a3,a4};
```

Perhatikan bahwa koefisien harus ditulis dalam format Q32 dan jangan lupa bahwa prosesor yang kita gunakan adalah prosesor *fixed-point* 32 bit sehingga setiap bilangan direpresentasikan dalam 32 bit dengan kisaran -2^{31} sampai $2^{31}-1$ atau -2147483648 – 2147483647 untuk -1 sampai 1.

Q31 adalah salah satu contoh format Q, yaitu penulisan bilangan untuk bilangan *fixed-point*. Pada BF561 yang merupakan prosesor *fixed-point* 32 bit, maka untuk N=32, digunakan format Q31. Format Q31 ini merepresentasikan fraksional 31 bit.

Pada format ini, MSB adalah *sign bit* yang diikuti oleh suatu titik imajiner, dan kemudian 31 bit atau mantisa yang dinormalisasi ke 1. Bilangan Q31 memiliki kisaran desimal antara -1 sampai 0.9999 (0x8000h sampai 0x7FFFh). Bilangan fraksional h dapat direpresentasikan

sebagai $h = \sum_{n=0}^N b_n 2^{-n}$ atau $h = -b_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + \dots + b_N 2^{-N}$, dimana h adalah bilangan fraksional dan b adalah biner 1 atau 0.

Konversi bilangan real ke bilangan fraksional dilakukan dengan

$$\text{koef} = \text{round}(2^{31} * \text{koef});$$

Untuk konversi bilangan fraksional ke bilangan real dilakukan dengan

$$\text{koef} = \text{koef} / 2^{31};$$

3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN

Pelajari proses pemfilteran dengan menggunakan buffering, circular buffer, dan operasi matematik dengan bilangan fraksional.

Tugas pendahuluan :

1. Bagaimana cara menggunakan koefisien filter agar dapat menghasilkan filter yang dapat memfilter sinyal masukan? (Jelaskan secara singkat dan melalui persamaan matematis)
2. Bagaimana cara mengimplementasikan hal tersebut pada program?
3. Buatlah flowchart program dari pertanyaan nomor 2 jika filter mempunyai 3 buah koefisien!
4. Apa yang dimaksud pemfilteran dengan buffer sirkular?

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software Matlab

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software Dev-C++ sudah terinstal dalam komputer.

Hal yang perlu diperhatikan ketika menjalankan simulasi adalah selalu buat file yang berbeda dengan file yang sedang dibuka oleh Microsoft Office Excel agar simulasi dapat menulis file. Karena file yang sedang dibuka tidak akan bisa ditulis oleh program simulasi.

4.2.1. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR

Pada bagian ini anda diminta untuk membuat simulasi pem-filter-an realtime FIR dengan Dev-C++. Fungsi internal filter menggunakan metoda non realtime, karena menghitung sinyal output dengan sinyal input yang lengkap. Buat perhitungan filter realtime FIR dengan menggunakan Dev-C++.

Ketentuan:

- Gunakan buffer input sepanjang orde filter (Contoh: jika orde filter adalah 16 maka buffer input adalah 16)
- Gunakan metoda loop sejumlah sampel sinyal input untuk simulasi clock
- Pada setiap loop (clock) mengindikasikan adanya sampel input yang baru. Ambil input baru dan tempatkan pada buffer input
- Pada setiap loop (clock) hitunglah sampel sinyal output yang dihasilkan pada loop tersebut, lalu tempatkan pada array output

Jalankan software Dev-C++. Pilih File > New Project. Kemudian pilih Jenis Console Application. Ganti nama project sesuai keinginan anda misalkan FIR. Pilih OK. Kemudian keluar jendela baru dan pilih folder untuk menempatkan project yang akan dibuat. Akan otomatis dibuat file source code main.cpp. Edit file ini dengan source code berikut:

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

// panjang tap filter
#define BUFFERLENGTH 9
#define LENGTH 100

#define PI 3.14159265

float koef_filter[BUFFERLENGTH] = {0.020041, 0.064976, 0.167492,
0.250847, 0.250847, 0.167492, 0.064976, 0.020041};
float buffer[BUFFERLENGTH];
float x[LENGTH];
float y[LENGTH];

int i, j;

float filter(float input) {
    float hasil;
    //buffering
    for(j = BUFFERLENGTH - 1; j > 0; j--) {
        buffer[j] = buffer[j-1];
    }
    //input disimpan di buffer 0
    buffer[0] = input;
```

```

        // perhitungan filter
        hasil = 0;
        for(j = 0; j < BUFFERLENGTH; j++) {
            hasil = hasil + buffer[j] * koef_filter[j];
        }

        // kembalikan hasil pemfilteran
        return hasil;
    }

int main(int argc, char *argv[])
{
    FILE *pFile;

    //siapkan input
    for(i=0; i < LENGTH; i++) {
        x[i] =
(sin(2*PI*200/16000*i)+sin(2*PI*1000/16000*i)+sin(2*PI*5000/16000*i)
) / 3;
    }

    //inisialisasi buffer dengan nol
    for(i=0; i < BUFFERLENGTH; i++) {
        buffer[i] = 0;
    }

    //loop mengambil input tiap sampel dan menaruhnya pada output
    for(i=0; i < LENGTH; i++) {
        y[i] = filter(x[i]);
    }

    //tuliskan output ke file
    pFile = fopen("output.txt", "w");
    for(i=0; i < LENGTH; i++) {
        fprintf(pFile,"%4.4f\n", y[i]);
    }
    fclose(pFile);

    system("PAUSE");
    return EXIT_SUCCESS;
}

```

Perhatikan ada bagian untuk menyiapkan input, bagian pemfilteran dan hasil output ditulis ke file dengan format text. Edit koefisien filter sesuai dengan filter yang didesain dengan MATLAB. Jalankan simulasi maka akan dihasilkan file text. Buka file text ini dengan Microsoft Office Excel. Lalu buat grafiknya.

Bandingkan hasil simulasi ini dengan fungsi internal filter.

4.2.2. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR dengan buffer sirkular

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan metode buffering sirkular. Dengan metoda ini maka setiap kali didapatkan sampel input, maka sampel-sampel terdahulu pada buffer tidak perlu digeser. Metoda buffering sirkular ini sangat sering digunakan karena operasi write pada memori biasanya membutuhkan waktu eksekusi yang cukup lama dibandingkan dengan operasi matematik biasa.

Edit source code dari bagian sebelumnya dengan mengubah bagian buffering dan perhitungan filter. Lakukan simulasi dan buatlah grafiknya.

Bandingkan hasil simulasi ini dengan hasil sebelumnya.

4.2.2. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR dengan bilangan fraksional

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan bilangan fraksional 32 bit. Semua koefisien filter direpresentasikan dengan bilangan fraksional. Buffer diisi dengan sampel input baru yang diisi dengan sampel input yang telah dikonversikan dari bilangan real menjadi bilangan fraksional. Modifikasi perkalian bilangan real menjadi perkalian bilangan fraksional. Simpan sampel sinyal output yang dihasilkan dengan terlebih dahulu mengkonversikan bilangan fraksional menjadi bilangan real.

Edit source code dari bagian sebelumnya dengan mengubah fungsi filter, juga koefisien dan buffer menjadi tipe integer. Tambahkan bagian konversi output menjadi float sebelum ditulis ke file. (Gunakan casting).

Bandingkan hasil simulasi ini dengan hasil sebelumnya.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikum dan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai**.

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN III

PENGUNAAN VISUAL DSP++ 5.0

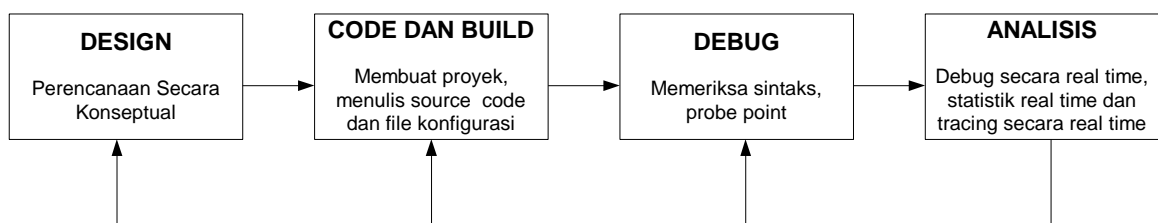
1. TUJUAN

1. Praktikan mengerti dan mampu menjelaskan fungsi-fungsi *tool-tool* yang terdapat VisualDSP++.
2. Praktikan mampu melakukan *debugging* dengan VisualDSP++.

2. DASAR TEORI

Untuk pemrograman Blackfin digunakan *software* VDSP++ (VisualDSP++) versi 5.0. Perangkat lunak ini digunakan untuk mengembangkan proyek berisi *source code* yang nantinya akan dimasukkan ke dalam prosesor Blackfin dalam bentuk *object code*. VDSP++ ini mampu untuk *men-debug*, *monitoring* memori pada prosesor DSP serta mampu melakukan analisis secara *real time*.

Terdapat fase dalam penggunaan *VisualDSP++* yang ditunjukkan oleh blok diagram pada gambar berikut :



Beberapa *tool* penting pada VisualDSP++ :

2.1. VIEW

2.1.1 View nilai variabel pada kode:

Pada VisualDSP++ ini terdapat *tool* yang dapat memperlihatkan nilai-nilai variabel pada kode. Prosedurnya adalah sebagai berikut:

- Pada *Menu bar* klik : View → Debug Windows → Expressions

Lalu muncul:

Device 0:a: Expressions			
Name	Type	Size	Value
U	fract16[]	0x00000004	{...}
[0]	short	0x00000002	0x8000
[1]	short	0x00000002	0x8000

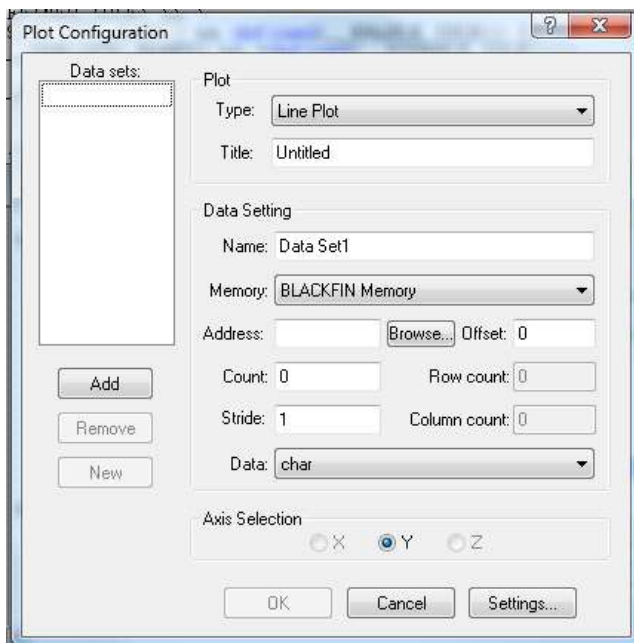
- Klik pada bagian yang kosong pada kolom *Name* lalu masukan nama variabel yang hendak dicari namanya. Setelah itu, tipe, ukuran dan nilai dari variabel tersebut akan muncul pada kolom lainnya. Sebagai catatan, nilai tidak dapat dilihat apabila program sedang dalam mode *Run*.

2.1.2. View grafik waktu dan frekuensi:

Pada VisualDSP++ ini terdapat *tool* yang dapat memperlihatkan grafik waktu dan frekuensi dari sinyal-sinyal. Prosedur:

- Pada *Menu bar* klik : View → Debug Windows → Plot → New...

Lalu muncul :



- Pilih *Type* grafik yang hendak ditampilkan
- Berikan judul dan nama yang sesuai untuk grafik yang hendak ditampilkan.

- Pilih variabel yang hendak ditinjau nilainya dengan menekan tombol *Browse...*
- Masukkan nilai *Count* lebih dari 0 dan pilih tipe *Data:* yang bersesuaian
- Tekan tombol *Add* pada sebelah kiri, lalu tekan tombol *OK*.

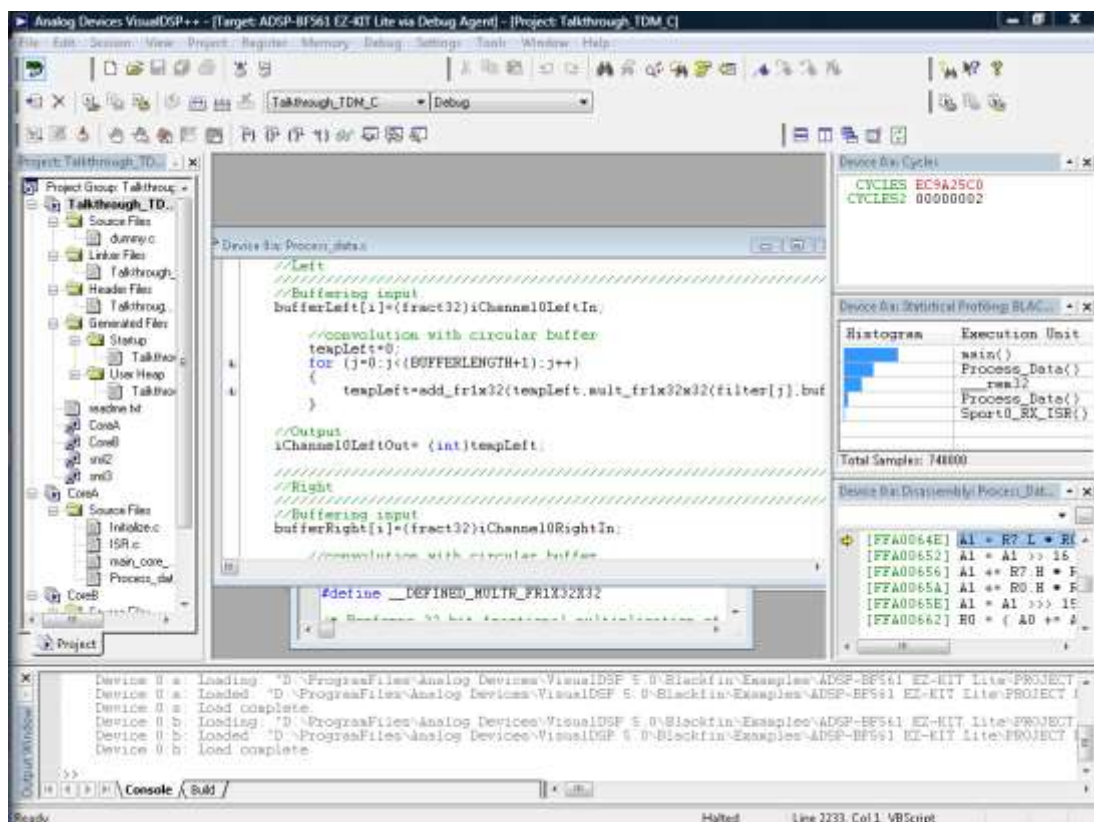
3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN

Tugas Pendahuluan :

1. Tuliskan dan jelaskan tools yang terdapat pada VisualDSP++ !
2. Buatlah flowchart program pemfilteran yang memanfaatkan buffer sirkular!
3. Mengapa konvolusi secara on-line diimplementasikan dengan menggunakan buffer sirkular?

Persiapan Praktikum :

Untuk menjalankan VisualDSP++, Double klik ikon “VisualDSP++ Environment”. Dengan melakukan ini maka akan muncul *workspace* yang berisikan Menu bar, tool bar, Project window, Console/Build window dan code window seperti ditampilkan berikut ini:



4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software Visual DSP++ 5.0

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software Visual DSP++ 5.0 sudah terinstal dalam komputer.

4.2.1. Percobaan meneruskan sinyal masukan dari input sebagai keluaran pada port output

1. Hubungkan kabel plug(male) dari sinyal masukan pada jack(female) dari board BF561EZ pada kanal 0.
2. Hubungkan kabel plug(male) dari sinyal keluaran pada jack(female) dari board BF561EZ juga pada kanal 0, selanjutnya hubungkan ujung lain dari kabel tersebut pada speaker.
3. Buka proyek yang bernama Talkthrough dari kumpulan contoh program. Buka proyek (klik `File` → `Open` → `Project Group...`) cari file bernama `Talkthrough_TDM_C.dpg`, tersimpan di `folder c:\Program Files\Analog Devices\VisualDSP 5.0\Blackfin\Examples\ASDP-BF561 EZ-KIT Lite\Audio Codec Talkthrough - TDM (C)`.
4. Untuk melihat source code dari proyek ini, buka `Process_data.c` pada bagian `CoreA` → `SourceFiles` (dengan jalan klik tanda (+) pada tulisan `CoreA` dan `Source Files`) dari `Project window`.
5. Pada VisualDSP, proses `Compile`, `Build` dan `Load` dapat dilakukan bersamaan. Pada *menu bar* klik `Project` → `Build Project`. Atau dapat pula dilakukan dengan cara menekan tombol `F7` pada keyboard.
6. Untuk menjalankan program yang telah di-build, pada *menu bar* klik `Debug` → `Run`. Atau dapat pula dilakukan dengan cara menekan tombol `F5` pada keyboard.
7. Lakukan pengamatan pada sinyal masukan dan keluaran. Akan terdengar bahwa sinyal keluaran sama seperti sinyal masukan / sinyal keluaran merupakan sinyal masukan yang diteruskan saja.

4.2.2. Percobaan meneruskan sinyal keluaran sebagai hasil amplifikasi sinyal masukan.

Apabila pada percobaan sebelumnya kita hanya meneruskan sinyal masukan sebagai sinyal keluaran saja, maka pada percobaan kali ini kita akan mengubah-ubah amplitude dari sinyal masukan. Percobaan ini bertujuan untuk memberikan gambaran mudah mengenai pengolahan-pengolahan sinyal secara sederhana yang mungkin dilakukan dalam board Blackfin BF561EZ.

1. Tetap hubungkan kabel plug(male) dari sinyal masukan pada jack(female) dari board BF561EZ pada kanal 0 serta kabel plug(male) dari sinyal keluaran pada jack(female) dari board BF561EZ juga pada kanal 0. Hubungkan pula ujung lain dari kabel tersebut pada speaker.
2. Buka proyek yang bernama Talkthrough dari kumpulan contoh program. Buka proyek (klik File → Open → Project Group...) cari file bernama Talkthrough_TDM_C.dpg, tersimpan di *folder* c:\Program Files\Analog Devices\VisualDSP 5.0\Blackfin\Examples\ASDP-BF561 EZ-KIT Lite\Audio Codec Talkthrough - TDM (C).
3. Buka Process_data.c pada bagian CoreA → SourceFiles (dengan jalan klik tanda (+) pada tulisan CoreA dan Source Files) dari Project window.
4. Amati terdapat code tertulis sebagai berikut:

```
iChannel0LeftOut = iChannel0LeftIn;  
  
iChannel0RightOut = iChannel0RightIn;  
  
iChannel1LeftOut = iChannel1LeftIn;  
  
iChannel1RightOut = iChannel1RightIn;
```

Keempat baris kode diatas menunjukkan bahwa setiap sinyal masukan diteruskan begitu saja sebagai sinyal keluaran. Baris pertama menunjukkan bahwa kanal 0 kiri masukan diteruskan pada kanal 0 kiri keluaran. Baris kedua menunjukkan bahwa kanal 0 kanan diteruskan pada kanal 0 kanan keluaran, dan seterusnya.

5. Ubah baris pertama dan kedua menjadi seperti berikut :

```
iChannel0LeftOut = 2*iChannel0LeftIn;  
  
iChannel0RightOut = 0.25*iChannel0RightIn;
```

6. Pada *menu bar* klik Project → Build Project. Atau dapat pula dilakukan dengan cara menekan tombol F7 pada keyboard.

7. Untuk menjalankan program yang telah di-build, pada *menu bar* klik Debug → Run. Atau dapat pula dilakukan dengan cara menekan tombol F5 pada keyboard.
8. Lakukan pengamatan pada sinyal masukan dan keluaran. Amati perubahan yang terjadi antara sinyal masukan dan keluaran baik pada kanal 0 kanan dan kiri.

4.2.3. Percobaan FIR dengan Circular Buffer sederhana

Pada percobaan sebelumnya telah dilakukan modifikasi terhadap amplituda sinyal masukan. Pengolahan tersebut dapat dianggap sebagai pengolahan sinyal yang sangat sederhana. Namun pada umumnya proses pengubahan sinyal masukan menjadi sinyal keluaran adalah melalui proses pemfilteran secara konvolutif. Pada percobaan kali ini akan dilakukan percobaan dengan menggunakan filter FIR yang koefisiennya kita tentukan secara sembarang. Perlu diperhatikan bahwa proses konvolusi yang dilakukan adalah menggunakan circular buffer karena proses pemfilteran harus dilakukan secara *on-line*.

1. Tetap hubungkan kabel plug(male) dari sinyal masukan pada jack(female) dari board BF561EZ pada kanal 0 serta kabel plug(male) dari sinyal keluaran pada jack(female) dari board BF561EZ juga pada kanal 0. Hubungkan pula ujung lain dari kabel tersebut pada speaker.
2. Buka proyek yang bernama Talkthrough dari kumpulan contoh program. Buka proyek (klik File→ Open → Project Group...) cari file bernama Talkthrough_TDM_C.dpg, tersimpan di *folder*: \Program Files\Analog Devices\VisualDSP 5.0\Blackfin\Examples\ ASDP-BF561 EZ-KIT Lite\Audio Codec Talkthrough - TDM (C).
3. Buka Process_data.c pada bagian CoreA→SourceFiles (dengan jalan klik tanda (+) pada tulisan CoreA dan Source Files) dari Project window.
4. Ketik inisialisasi sebagai berikut:

```
#define BUFFERLENGTH 32

int
bufferLeft[BUFFERLENGTH],bufferRight[BUFFERLENGTH]={0,0,0};

int tempLeft, tempRight = 0;

int i,j=0;

////////////////////////////////////
///

//Filter Coefficients (from MATLAB):

int filter[BUFFERLENGTH]={ -4.0920e+006,
-4.8192e+006, -6.0388e+006, -7.0440e+006,
```

```

-6.5343e+006, -2.8071e+006, 5.9122e+006,
2.1061e+007, 4.3334e+007, 7.2381e+007,
1.0666e+008, 1.4354e+008, 1.7958e+008,
2.1099e+008, 2.3425e+008, 2.4663e+008,
2.4663e+008, 2.3425e+008, 2.1099e+008,
1.7958e+008, 1.4354e+008, 1.0666e+008,
7.2381e+007, 4.3334e+007, 2.1061e+007,
5.9122e+006, -2.8071e+006, -6.5343e+006,
-7.0440e+006, -6.0388e+006, -4.8192e+006,
-4.0920e+006};

////////////////////////////////////
///

```

5. Lalu pada bagian `void Process_Data(void)`, masukan kode-kode berikut ini:

```

////////////////////////////////////
///

//channel0

////////////////////////////////////
///

    //////////////////////////////////
    //

        //Left

        //////////////////////////////////
        //

            //Buffering input

            bufferLeft[i] = iChannel0LeftIn;

            //convolution with circular buffer

            tempLeft=0;

            for (j = 0; j < BUFFERLENGTH; j++)

            {

tempLeft += (filter[j] >> 16) * (bufferLeft[(i + j) %
BUFFERLENGTH] >> 15);

```

```

        }

//Output

iChannel0LeftOut = tempLeft;

////////////////////////////////////
// //Right
////////////////////////////////////
//

//Buffering input

bufferRight[i]= iChannel0RightIn;

    //convolution with circular buffer

    tempRight=0;

    for (j = 0; j < BUFFERLENGTH; j++)

    {

tempRight += (filter[j] >> 16) * (bufferRight[(i + j) %
BUFFERLENGTH] >> 15);

    }

//Output

iChannel0RightOut=tempRight;

////////////////////////////////////
//

//decreasing i (circular)

////////////////////////////////////
//

i=(i+BUFFERLENGTH-1)%BUFFERLENGTH;

////////////////////////////////////
//

//channel1

////////////////////////////////////
//

////////////////////////////////////

//Left

```

```

////////////////////////////////////
iChannel1LeftOut = iChannel1LeftIn;

////////////////////////////////////

//right

////////////////////////////////////

iChannel1RightOut = iChannel1RightIn;

```

6. Pada *menu bar* klik Project → Build Project. Atau dapat pula dilakukan dengan cara menekan tombol F7 pada keyboard.
7. Untuk menjalankan program yang telah di-build, pada *menu bar* klik Debug → Run. Atau dapat pula dilakukan dengan cara menekan tombol F5 pada keyboard.
8. Lakukan pengamatan pada sinyal masukan dan keluaran. Proses pemfilteran apa yang terjadi di dalam board BF561EZ?

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikum dan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai**.

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN IV

DESAIN FILTER DAN IMPLEMENTASI ALGORITMA DSP

1. TUJUAN

1. Praktikan mampu menjelaskan bagian-bagian penting dari Blackfin BF561EZ beserta fungsinya
2. Praktikan mampu melakukan *trouble-shooting* BF561EZ
3. Mengimplementasikan pem-*filter*-an FIR untuk berbagai macam *filter* di BF561EZ
4. Melakukan verifikasi filter FIR hasil implementasi

2. DASAR TEORI

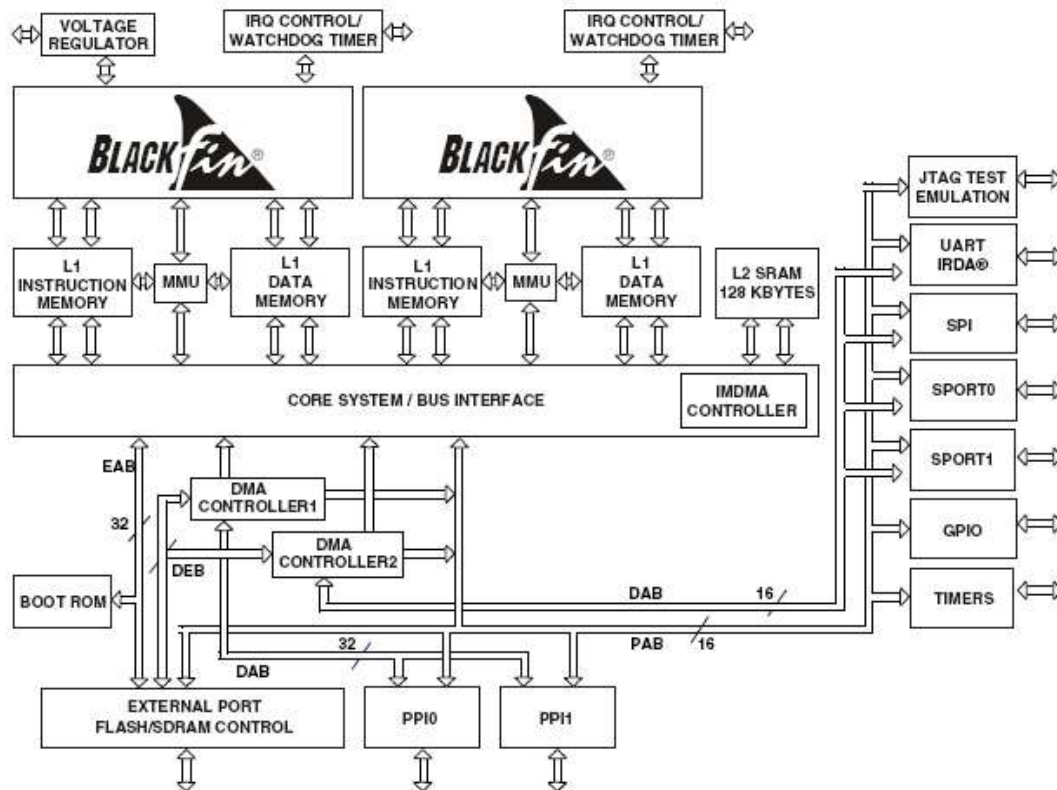
Blackfin BF561EZ adalah sebuah *development board* produksi dari Analog Devices yang digunakan untuk penelitian dan pengembangan aplikasi menggunakan Prosesor ADSP BF561. BF561EZ ini terdiri dari bermacam-macam *peripheral* yang digunakan untuk pengembangan.

BF561EZ mempunyai fitur :

- Prosesor ADSP-BF561 Blackfin
- 64 MB (16 M x 16-bit x 2) SDRAM
- 8 MB (4 M x 16-bit) FLASH memory
- AD1836 multichannel 96 kHz audio codec
- 3 RCA jack untuk composite (CVBS), differential component (YUV) atau S video (Y/C) input
- RCA jack untuk input/output audio stereo
- 10-bit video decoder ADV7183A
- NTSC/PAL video encoder ADV7179

Pada Gambar 1 ditampilkan blok diagram dari Blackfin BF561EZ. ADSP-BF561 merupakan anggota dari keluarga prosesor Blackfin yang memang ditargetkan untuk konsumen aplikasi-

aplikasi multimedia. Di dalam perangkat ini terdapat dua core prosesor Blackfin yang independen satu sama lain yang menawarkan performa tinggi serta konsumsi daya yang rendah dengan tetap menjaga kemudahan penggunaan serta kompatibilitas. Arsitektur ini mengombinasikan mesin pengolah sinyal dual-MAC, kemampuan single-instruction multiple-data (SIMD) yang fleksibel dan fitur-fitur multimedia ke dalam arsitektur *single-instruction-set*.



Gambar 1: Blok Diagram Blackfin BF561EZ

Produk-produk Blackfin memiliki fitur *Dynamic Power Management*, yang merupakan kemampuan untuk memvariasikan tegangan dan frekuensi operasional, demi optimasi konsumsi daya disesuaikan dengan tugas yang dijalankan.

2.1. PERIFERAL-PERIFERAL ADSP-BF561

Sistem ADSP-BF561 memiliki periferal-periferal sebagai berikut:

- Antarmuka Periferal Parallel / *Parallel Peripheral Interfaces* (PPIs)
- Port Serial / *Serial Ports* (SPORTs)
- Antarmuka Periferal Serial / *Serial Peripheral Interface* (SPI)
- Timer multi-guna / *General-Purpose Timers*
- Universal Asynchronous Receiver Transmitter (UART)

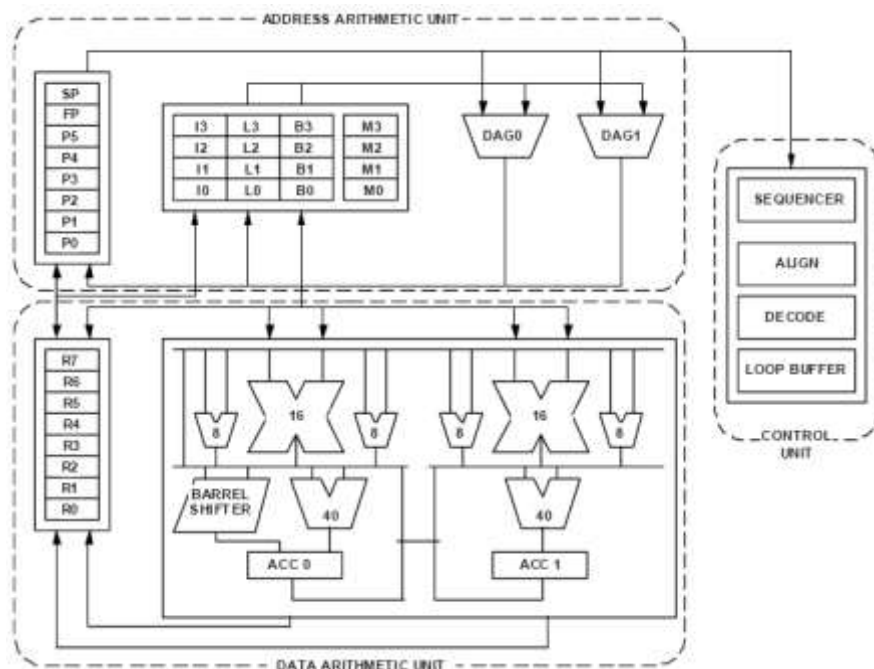
- Watchdog Timers
- I/O multi-guna / *General Purpose I/O* (Programmable Flags)

Periferal-periferal tersebut di atas terhubung dengan inti melalui beberapa bus ber-*bandwidth* besar, seperti ditampilkan pada Gambar 1.

Semua periferal kecuali I/O multi-guna dan Timer, didukung oleh struktur DMA yang fleksibel, termasuk di antaranya adalah dua buah pengatur DMA (DMA1 dan DMA2) dan sebuah pengatur internal memory DMA (IMDMA). Pada masing-masing dari DMA1 dan DMA2, terdapat dua belas kanal periferal DMA yang dapat deprogram dan dua aliran memori DMA terpisah, ditujukan untuk transfer data antar ruang memori DSP, yang mencakup SDRAM dan memori *asynchronous*. Beberapa bus on-chip menyediakan bandwidth yang cukup untuk menjaga agar inti prosesor berjalan meskipun terdapat pula aktivitas pada semua periferal-periferal on-chip dan periferal-periferal eksternal.

2.2. ARSITEKTUR INTI (CORE) ADSP-BF561

ADSP-BF561 memiliki dua inti Blackfin identik dan setiap inti mengandung dua 16-bit multiplier, dua 40-bit accumulator, dua 40-bit arithmetic logic unit (ALU), empat 8-bit video ALU dan sebuah 40-bit shifter bersamaan dengan unit-unit fungsional dari setiap inti, seperti ditampilkan pada Gambar 2. Unit-unit komputasional memproses data 8-, 16-, atau 32-bit dari register.



Gambar 2. Arsitektur setiap inti BF-561

File register komputasi mengandung delapan register 32-bit. Saat menjalankan operasi-operasi komputasi pada data operand 16-bit, file register beroperasi sebagai 16 register 16-bit yang independen. Semua operand untuk operasi-operasi komputasi berasal dari file register multi-port dan field-field instruksi yang konstan.

Setiap MAC dapat menjalankan sebuah 16 x 16 bit perkalian setiap siklusnya, dengan akumulasi ke sebuah hasil 40-bit. Format-format signed dan unsigned, pembulatan dan saturasi didukung di dalamnya.

ALU menjalankan set-set traditional aritmatikadan operasi-operasi logika pada data 16-bit atau 32-bit. Beberapa instruksi-instruksi khusus tergabung di dalamnya untuk mempercepat berbagai tugas pengolahan sinyal. Ini mencakup operasi-operasi bit seperti *field extract* dan *population count*, perkalian modulo 2^{32} , pembagian primitive, saturasi dan pembulatan, dan deteksi sign/eksponen. Set dari instruksi-instruksi video mencakup *bytealignment* dan operasi *packing*, penjumlahan 16-bit dan 8-bit dengan *clipping*, operasi rata-rata *8-bit*, dan operasi-operasi pengurangan / nilai absolute/ akumulasi 8-bit. Selain itu tersedia pula instruksi-instruksi perbandingan dan pencarian vektor. Untuk beberapa instruksi, dua operasi 16-bit ALU dapat dijalankan bersamaan pada pasangan register (16-bit paruh tengah atas dan 16-bit paruh tengah bawah dari register komputasi). Juga dengan menggunakan ALU kedua, operasi quad 16-bit juga dimungkinkan.

Shifter 40-bit dapat menyimpan data dan juga menjalankan operasi pergeseran, rotasi, normalisasi dan ekstraksi.

Sebuah pegurut program mengendalikan aliran dari eksekusi instruksi termasuk instruksi alignment dan decoding. Untuk pengendalian aliran program, pegurut mendukung PC-relative dan conditional jump tidak langsung (dengan prediksi cabang statis), dan panggilan-panggilan subrutin. Perangkat keras disediakan untuk mendukung *zero-overhead looping*.

Unit alamat aritmatika menyediakan dua alamat untuk pengambilan secara simultan dari memori. Unit ini mengandung file register multi-port yang terdiri dari empat set 32-bit, yaitu register Index, register Modify, register Length dan register Base (untuk circular buffer), dan delapan register pointer 32-bit tambahan.

Produk-produk Blackfin mendukung arsitektur Harvard termodifikasi dalam kombinasinya dengan struktur memori hierarki. Memori level 1 (L1) biasanya beroperasi pada kecepatan prosesor maksimum tanpa latensi. Pada level L1, memori instruksi memegang hanya instruksi-instruksi saja dan memori data memegang data, sedangkan sebuah memori data *scratchpad* terdedikasi menyimpan informasi mengenai stack dan variabel-variabel lokal.

Selain itu tersedia pula blok-blok memori L1, yang dapat dikonfigurasi sebagai campuran antara SRAM dan cache. Memory Management Unit (MMU) menyediakan proteksi memori untuk pekerjaan-pekerjaan individual yang mampu beroperasi pada inti dan dapat melindungi register sistem dari akses yang tidak diinginkan.

Inti ganda pada ADSP-BF561 saling membagi sistem memori *on-chip* L2, yang mampu menyediakan akses SRAM kecepatan tinggi dengan hambatan yang cukup besar. Memori L2

merupakan suatu kesatuan memori instruksi dan data serta dapat menyimpan berbagai campuran data dan kode yang dibutuhkan oleh desain sistem.

Arsitektur menyediakan tiga mode operasi: User, Supervisor, dan Emulation. Mode User memiliki akses terbatas untuk sebuah subset dari sumber daya sistem sehingga menyediakan lingkungan perangkat lunak yang terlindungi. Mode Supervisor dan Emulation memiliki akses yang tak terbatas pada sumber daya inti.

Set instruks Blackfin telah dioptimasi sedemikian sehingga op-codes 16-bit mewakili instruksi-instruksi yang paling sering digunakan. Instruksi-instruksi DSP yang kompleks dikodekan menjadi op-codes 32-bit sebagai instruksi-instruksi multifungsi. Produk-produk Blackfin mendukung kemampuan multi-isu yang terbatas, di mana instruks 32-bit dapat diisukan secara parallel bersamaan dengan instruksi-instruksi 16-bit. Ini memungkinkan seorang programmer untuk menggunakan banyak sumber daya inti dalam satu siklus instruksi.

Bahasa assembly ADSP-BF561 menggunakan sebuah sintaks aljabar. Arsitektur juga diptimaskan untuk digunakan dengan sebuah *compiler* C.

2.3. DASAR TEORI PEMFILTERAN

Sinyal di dunia nyata ada dalam bentuk analog dan selalu merupakan sinyal komposit antara bagian yang diketahui dan yang tidak diketahui. Misalnya, suara di jalur telepon adalah gabungan dari suara *speaker* di ujung dan derau. Pendengar di ujung harus mem-*filter* suara dominan agar pembicaraan dapat berlangsung. Contoh ini menunjukkan bahwa telepon mem-*filter* hampir semua derau kanal dan selanjutnya terserah pengguna untuk mem-*filter* dan mengartikan suara. Karenanya, pem-*filter*-an adalah teknik pemrosesan sinyal yang mendasar untuk mengekstrak sinyal yang diperlukan dari bentuk aslinya.

Semua teknik pem-*filter*-an analog –*low-pass*, *high-pass*, *band-pass*, dan *band-stop*– dapat diimplementasikan di domain digital bila sinyalnya di-*sampling* dengan tepat. Sampel sinyal ini dikirim melalui struktur *filter* digital untuk melakukan pem-*filter*-an.

Filter digital diklasifikasi menjadi *filter* FIR (Finite Impulse Response) dan IIR (Infinite Impulse Response). Masing-masing *filter* ini dapat melakukan hal yang serupa dengan *filter* analog.

input analog \rightarrow $h(t)$ \rightarrow output analog

input analog \rightarrow A/D \rightarrow $h(n)$ \rightarrow D/A \rightarrow output analog

Filter analog mengambil input analog dan menghasilkan output analog. *Filter* digital, dengan adanya perangkat pencuplikan dan konverter, melakukan hal yang sama dengan *filter* analog.

Pada kasus pem-*filter*-an digital, fungsi transfernya akan berupa model *filter* FIR ataupun IIR. Sinyal keluarannya dapat ditulis sebagai persamaan perbedaan agar model *filter* dapat diimplementasikan ke *hardware* digital.

Filter FIR adalah sistem waktu diskrit yang respon impulsnya terbatas:

$$h[n]=(h_0, h_1, h_2, \dots, h_{N-1})$$

Fungsi transfer $H(z)$ berupa polinom dalam z^{-1} , yaitu

$$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}+\dots+h_{N-1}z^{-(N-1)}$$

Karenanya *filter* ini memiliki $N-1$ zero dan tidak memiliki pole.

$$\text{Persamaan perbedaan } \textit{filter} \text{ FIR tidak rekursif, } y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$


Keuntungan filter FIR :

- Selalu stabil (karena fungsi transfer tidak memiliki *pole*)
- Dapat dirancang untuk memiliki fasa linier
- Lebih mudah untuk diimplementasikan ke *hardware*
- Respon transien memiliki durasi yang terbatas

Tetapi *filter* ini juga memiliki beberapa kelemahan yaitu membutuhkan banyak orde untuk memperoleh respons *filter* yang semakin baik sehingga otomatis *delay*-nya pun lebih lama.

 Coba pelajari apa keuntungan yang diperoleh dengan bekerja dengan fasa linier?

Ada dua metode untuk merancang *filter* FIR, yaitu metode Windowing dan metode Multiband dengan Band transisi.

 Coba pelajari keduanya !

2.4. PENYIMPANAN KOEFISIEN FILTER DAN SINYAL INPUT

Dari percobaan sebelumnya Anda telah memperoleh koefisien *filter* untuk melakukan pem-*filter*-an. Kita akan menggunakannya sebagai koefisien *filter* yang akan diterapkan di BF561. Nantinya kita akan mencoba program *filter* dalam bahasa pemrograman C ataupun Assembly, jadi kita dapat menuliskan koefisien tersebut di program kita, seperti berikut (contoh)

```
int koef={a1,a2,a3,a4};
```

Tetapi biasanya akan lebih mudah bila kita menyimpan koefisien tersebut di dalam *file*. Hal ini memudahkan apabila kita ingin mengganti-ganti koefisien kita dengan cepat.

Perhatikan bahwa koefisien harus ditulis dalam format Q31 dan jangan lupa bahwa prosesor yang kita gunakan adalah prosesor *fixed-point* 32 bit sehingga setiap bilangan direpresentasikan dalam 32 bit dengan kisaran -2^{31} sampai $2^{31}-1$ atau -2147483648 – 2147483647 untuk -1 sampai 1 .

Q31 adalah salah satu contoh format Q, yaitu penulisan bilangan untuk bilangan *fixed-point*. Pada BF561 yang merupakan prosesor *fixed-point* 32 bit, maka untuk N=32, digunakan format Q31. Format Q31 ini merepresentasikan fraksional 31 bit.

Pada format ini, MSB adalah *sign bit* yang diikuti oleh suatu titik imajiner, dan kemudian 31 bit atau mantisa yang dinormalisasi ke 1. Bilangan Q31 memiliki kisaran desimal antara -1 sampai 0.9999 (0x8000h sampai 0x7FFFh). Bilangan fraksional h dapat direpresentasikan

sebagai $h = \sum_{n=0}^N b_n 2^{-n}$ atau $h = -b_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + \dots + b_N 2^{-N}$, dimana h adalah bilangan fraksional dan b adalah biner 1 atau 0.

1. Buat skrip dalam Matlab yang mengubah ketiga koefisien *filter* dan koefisien sinyal input menjadi format Q31 sebelum menyimpannya ke *file* dalam bentuk di atas! Misal (contoh untuk koefisien filter fir 31 titik):

```
koef=fir1(30,800/24000);  
  
koef=round(2^31*koef);  
  
fid=fopen('koef.dat','w+');  
  
count=fprintf(fid,'%d ',koef);  
  
fclose(fid)
```

2. Jalankan skrip yang Anda buat.
3. Skrip pada contoh di atas akan menghasilkan file koef.dat pada folder tempat file skrip berada. File koef.dat ini berisi koefisien-koefisien dalam format Q31.
4. Setelah Anda menjalankan keempat skrip tersebut, maka Anda akan memperoleh empat buah file *.dat yang berisi koefisien-koefisien dalam format Q31. Sekarang Anda siap untuk melakukan pem-*filter*-an FIR.

2.5. WHITE NOISE

White noise adalah sinyal acak (*random*) dengan spektrum daya flat. Artinya sinyal white noise memiliki daya yang sama pada bandwidth yang tetap pada setiap frekuensi tengah. White noise merupakan nama yang diturunkan dari cahaya putih yang spektrum daya dari cahayanya terdistribusi pada rentang band cahaya tampak. Bandwidth dari white noise terbatas pada prakteknya oleh mekanisme pembangkitan noise, media transmisi. Sinyal random dapat dikatakan white noise jika memiliki spektrum yang flat pada bandwidth terlebar yang mungkin dapat dilewatkan pada sebuah medium.

3. TUGAS PENDAHULUAN

1. Jelaskan perbedaan antara filter FIR dengan filter IIR!

2. Apa yang dimaksud dengan fasa linier hasil pemfilteran FIR ? Jelaskan keuntungan yang diperoleh dari fasa linier pada hasil pemfilteran FIR!
3. Berikan penjelasan singkat mengenai metode windowing dan multiband dengan transition band) pada design filter FIR di MATLAB!
4. Gambarkan workflow langkah-langkah percobaan beserta hasil yang diharapkan pada percobaan kedua.

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software MATLAB dan Visual DSP++ 5.0
3. Kit Blackfin BF561EZ, adaftor, kabel RCA, converter RCA, dan speaker aktif.

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software MATLAB dan Visual DSP++ 5.0 sudah terinstal dalam komputer.

4.2.1. PERCOBAAN IMPLEMENTASI FILTER FIR : LOW-PASS FILTER

1. Dengan cara yang sudah dibahas pada modul dua dan metode pengambilan koefisien yang telah dibahas pada subbab 4 di atas FIR buat filter tipe low-pass orde 31 dengan frekuensi cut-off 800 Hz dengan MATLAB.
2. Tampilkan VisualDSP++ dan siapkan program yang digunakan pada modul tiga.
3. Buka `Process_data.c` pada bagian `CoreA` → `SourceFiles` (dengan jalan klik tanda (+) pada tulisan `CoreA` dan `Source Files`) dari Project window
4. Perhatikan bagian:

```
//Filter Coefficients (from MATLAB):
int filter[BUFFERLENGTH]={...};
////////////////////////////////////
```

Perhatikan bahwa nilai-nilai yang dimasukkan di atas merupakan koefisien-koefisien dari filter FIR.

5. Ganti koefisien-koefisien tersebut dengan koefisien low-pass filter yang telah didapatkan dalam MATLAB.
6. Pada menu bar klik `Project` → `Build Project`. Atau dapat pula dilakukan dengan cara menekan tombol F7 pada keyboard.
7. Untuk menjalankan program yang telah di-build, pada menu bar klik `Debug` → `Run`. Atau dapat pula dilakukan dengan cara menekan tombol F5 pada keyboard.
8. Lakukan pengamatan pada sinyal masukan dan keluaran. Perhatikan efek pemfilteran (low-pass filtering) yang telah diterapkan.
9. Uji filter yang telah dibuat dengan menggunakan White noise sebagai input. White noise mempunyai daya sama pada semua komponen frekuensi. White noise ini bisa dihasilkan dengan program Audacity.
10. Rekam keluaran filter dengan program Audacity. Periksa spektrum sinyal hasil rekaman. Gunakan hasil ini sebagai konfirmasi apakah filter Low Pass telah berfungsi sesuai dengan yang diinginkan. Seharusnya dihasilkan sinyal dengan spektrum dengan daya yang sama pada rentang frekuensi pass band nya.

4.2.2. PERCOBAAN IMPLEMENTASI FILTER FIR: BAND-PASS FILTER

1. Dengan cara yang sudah dibahas pada modul dua dan metode pengambilan koefisien yang telah dibahas pada subbab 4 di atas FIR buat filter tipe band-pass orde 31 dengan frekuensi pass 3000 – 9000 Hz dengan MATLAB.
2. Tampilkan VisualDSP++ dan siapkan program yang digunakan pada modul tiga.
3. Buka `Process_data.c` pada bagian `CoreA` → `SourceFiles` (dengan jalan klik tanda (+) pada tulisan `CoreA` dan `Source Files`) dari `Project window`

4. Perhatikan bagian:

```
//Filter Coefficients (from MATLAB):
int filter[BUFFERLENGTH]={...};

////////////////////////////////////
```

Perhatikan bahwa nilai-nilai yang dimasukkan di atas merupakan koefisien-koefisien dari filter FIR.

5. Ganti koefisien-koefisien tersebut dengan koefisien band-pass filter yang telah didapatkan dalam MATLAB.

6. Pada menu bar klik `Project` → `Build Project`. Atau dapat pula dilakukan dengan cara menekan tombol `F7` pada keyboard.
7. Untuk menjalankan program yang telah di-build, pada menu bar klik `Debug` → `Run`. Atau dapat pula dilakukan dengan cara menekan tombol `F5` pada keyboard.
8. Lakukan pengamatan pada sinyal masukan dan keluaran. Perhatikan efek pemfilteran (band-pass filtering) yang telah diterapkan.
9. Uji filter yang telah dibuat dengan menggunakan White noise sebagai input. White noise mempunyai daya sama pada semua komponen frekuensi. White noise ini bisa dihasilkan dengan program Audacity.
10. Rekam keluaran filter dengan program Audacity. Periksa spektrum sinyal hasil rekaman. Gunakan hasil ini sebagai konfirmasi apakah filter Band Pass telah berfungsi sesuai dengan yang diinginkan. Seharusnya dihasilkan sinyal dengan spektrum dengan daya yang sama pada rentang frekuensi pass band nya.

4.2.3. PERCOBAAN IMPLEMENTASI FILTER FIR: HIGH PASS FILTER

1. Dengan cara yang sudah dibahas pada modul dua dan metode pengambilan koefisien yang telah dibahas pada subbab 4 di atas FIR buat filter tipe high-pass orde 31 dengan frekuensi cut-off 6000Hz dengan MATLAB.
2. Tampilkan VisualDSP++ dan siapkan program yang digunakan pada modul tiga.
3. Buka `Process_data.c` pada bagian `CoreA` → `SourceFiles` (dengan jalan klik tanda (+) pada tulisan `CoreA` dan `Source Files`) dari Project window
4. Perhatikan bagian:

```
//Filter Coefficients (from MATLAB):
int filter[BUFFERLENGTH]={...};

////////////////////////////////////
////////
```

Perhatikan bahwa nilai-nilai yang dimasukkan di atas merupakan koefisien-koefisien dari filter FIR.

5. Ganti koefisien-koefisien tersebut dengan koefisien high-pass filter yang telah didapatkan dalam MATLAB.

6. Pada menu bar klik `Project` → `Build Project`. Atau dapat pula dilakukan dengan cara menekan tombol `F7` pada keyboard.
7. Untuk menjalankan program yang telah di-build, pada menu bar klik `Debug` → `Run`. Atau dapat pula dilakukan dengan cara menekan tombol `F5` pada keyboard.
8. Lakukan pengamatan pada sinyal masukan dan keluaran. Perhatikan efek pemfilteran (high-pass filtering) yang telah diterapkan.
9. Uji filter yang telah dibuat dengan menggunakan White noise sebagai input. White noise mempunyai daya sama pada semua komponen frekuensi. White noise ini bisa dihasilkan dengan program Audacity.
10. Rekam keluaran filter dengan program Audacity. Periksa spektrum sinyal hasil rekaman. Gunakan hasil ini sebagai konfirmasi apakah filter High Pass telah berfungsi sesuai dengan yang diinginkan. Seharusnya dihasilkan sinyal dengan spektrum dengan daya yang sama pada rentang frekuensi pass band nya.

Tugas-tugas yang harus dibuat dalam laporan :

- Jelaskan penggunaan format bilangan Q32 pada prosesor fixed-point DSP Blackfin BF561-EZ
- Penjelasan keuntungan menggunakan circular buffer untuk implementasi filter FIR.
- Penjelasan mengenai hasil pemfilteran low-pass, band-pass dan high-pass dengan menggunakan DSP.
- Tuliskan apa yang anda dapat simpulkan dari praktikum ini.
- Saran untuk praktikum modul ini.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikum dan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai**.

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN V

IMPLEMENTASI ALGORITMA DSP LANJUTAN

1. TUJUAN

1. Praktikan memahami teknik implementasi algoritma pada DSP Blackfin BF561EZ
2. Praktikan mampu untuk mengimplementasikan berbagai macam algoritma pada DSP Blackfin BF561EZ

2. DASAR TEORI

2.1. SCRAMBLER

Dalam dunia telekomunikasi, scrambler merupakan sebuah sistem yang mampu menginversi / mengkodekan sinyal pesan pada sebuah transmitter sehingga pesan tidak dapat dimengerti oleh pihak penerima yang tidak memiliki perangkat *descrambler*. Teknik ini serupa dengan encoding hanya saja apabila teknik encoding diterapkan pada domain digital, maka scrambler diterapkan pada domain analog. Teknik scrambling ini dapat dilakukan dengan berbagai cara misalnya dengan mengubah besaran besaran seperti amplituda pada sinyal asli ataupun dengan serangkaian proses rumit seperti yang diterapkan pada satelit, radio relay dan modem PSTN sehingga pesan tidak dapat dimengerti lagi tanpa melalui proses descrambler. Namun pada praktikum kali ini, akan dilakukan proses scrambler yang paling sederhana pada sinyal masukan, yaitu dengan jalan mengubah secara periodik polaritas sinyal-sinyal masukan.

2.2. EFEK PELEBARAN STEREO

Efek pelebaran stereo ini merupakan salah satu dari serangkaian efek audio 3D. Sedangkan efek-efek audio 3D merupakan kumpulan dari efek-efek suara yang bertujuan untuk melebarkan image stereo yang dihasilkan oleh dua loudspeaker atau headphone stereo. Atau dapat juga berupa efek untuk menciptakan ilusi persepsi mengenai keberadaan sumber-sumber suara dalam ruang tiga dimensi, misalnya di belakang, di atas atau di bawah pendengar.

Efek pelebaran stereo sendiri dapat dihasilkan dengan memanipulasi hubungan antara sinyal tengah C dan sinyal sisi S. Kedua sinyal tersebut memiliki rumusan : $C=(L+R)/2$; $S=(L-R)/2$. Bagian positif dari sinyal sisi S kemudian dijumlahkan dengan sinyal tengah pada kanal kiri, sedangkan bagian yang fasanya berkebalikan dijumlahkan dengan sinyal tengah pada kanal

kanan. Adapun terkadang sejumlah delay (20-100ms) dapat diberikan juga kepada sinyal terinversi untuk memberikan efek reverberasi.

3. TUGAS PENDAHULUAN

1. Apa itu efek scrambler dan pelebaran stereo? Jelaskan tujuannya!
2. Bagaimana pengaruh suara yang terkena efek scrambler dan pelebaran stereo?
3. Sebutkan contoh aplikasi efek scrambler dan pelebaran stereo pada kehidupan sehari-hari.
4. Jelaskan secara singkat cara mengimplementasikan efek scrambler dan pelebaran stereo dengan BF561EZ (bukan menuliskan source code).

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software MATLAB dan Visual DSP++ 5.0
3. Kit Blackfin BF561EZ, adaptor, kabel RCA, converter RCA, dan speaker aktif.

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software MATLAB dan Visual DSP++ 5.0 sudah terinstal dalam komputer.

4.2.1. Implementasi Scrambler

1. Tampilkan VisualDSP++ dan buka kembali project Talkthrough.
2. Buka `Process_data.c` pada bagian CoreA → SourceFiles (dengan jalan klik tanda (+) pada tulisan CoreA dan Source Files) dari Project window
3. Pada inisiasi kode `Process_data.c`, tambahkan library perhitungan matematika dan juga inisiasi variabel yang diperlukan :

```
//inisiasi  
#include "Talkthrough.h"
```

```
#include "math.h"
```

```
int i, j = 0;
```

4. Pada bagian `Void Process_Data(Void)`, ganti kode-kode yang ada dengan kode-kode sebagai berikut:

```
//amplitude polarizer
```

```
j=(i+1)%2;
```

```
i=j;
```

```
//Channel0
```

```
iChannel0LeftOut = (pow((-1),j))*iChannel0LeftIn;
```

```
iChannel0RightOut = (pow((-1),j))*iChannel0RightIn;
```

```
//Channel1
```

```
iChannel1LeftOut = (pow((-1),j))*iChannel1LeftIn;
```

```
iChannel1RightOut = (pow((-1),j))*iChannel1RightIn;
```

5. Pada menu bar klik `Project` → `Build Project`. Atau dapat pula dilakukan dengan cara menekan tombol `F7` pada keyboard.
6. Untuk menjalankan program yang telah di-build, pada menu bar klik `Debug` → `Run`. Atau dapat pula dilakukan dengan cara menekan tombol `F5` pada keyboard.
7. Lakukan pengamatan pada sinyal masukan dan keluaran, terutama pada domain frekuensi. Perhatikan efek Scrambling yang telah diterapkan.
8. Lakukan **analisis pada laporan praktikum** mengenai hubungan teknik scrambling yang telah dilakukan dengan modulasi sinyal masukan.

4.2.2. Implementasi Efek Pelebaran Stereo

1. Tampilkan VisualDSP++ dan buka kembali project Talkthrough.
2. Buka `Process_data.c` pada bagian `CoreA` → `SourceFiles` (dengan jalan klik tanda (+) pada tulisan `CoreA` dan `Source Files`) dari `Project window`
3. Pada bagian `Void Process_Data(Void)`, ganti kode-kode yang ada dengan kode-kode sebagai berikut:

```
width = 10
```

```

norm = (width + 1) / 2;
left = (float)iChannel0LeftIn;
right = (float)iChannel0RightIn;

C=(left+right)/2;
S=(left-right)/2;

Left = (C + S * width) / norm;
right = (C - S * width) / norm;

iChannel0LeftOut=(int)left;
iChannel0RightOut=(int)right;

iChannel1LeftOut = iChannel0LeftIn;
iChannel1RightOut = iChannel0RightIn;

```

4. Pada menu bar klik **Project** → **Build Project**. Atau dapat pula dilakukan dengan cara menekan tombol F7 pada keyboard.
5. Untuk menjalankan program yang telah di-build, pada menu bar klik **Debug** → **Run**. Atau dapat pula dilakukan dengan cara menekan tombol F5 pada keyboard.
6. Lakukan pengamatan pada sinyal masukan dan keluaran. Perhatikan efek pelebaran stereo yang telah diterapkan.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikum dan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai**.

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

PERCOBAAN VI

DESAIN DAN IMPLEMENTASI FILTER IIR PADA BLACKFIN DSP

1. TUJUAN

1. Praktikan memahami dan mampu melakukan desain filter IIR
2. Praktikan memahami algoritma filter IIR realtime
3. Praktikan mampu untuk mengimplementasikan algoritma filter IIR realtime pada DSP Blackfin BF561EZ

2. TUGAS BELAJAR DAN TUGAS PENDAHULUAN

Sebelum mengikuti praktikum, ada beberapa hal yang harus dipelajari oleh praktikan untuk kelancaran jalannya praktikum.

1. Pelajari Filter IIR.
2. Buatlah simulasi filter IIR realtime dengan menggunakan MATLAB. Verifikasi filter IIR realtime ini dengan membandingkannya dengan fungsi filter dari MATLAB.
3. Pelajari apa yang dimaksud dengan white noise, bagaimana spektrum dari sinyal white noise bisa digunakan untuk pengujian sistem.

Tugas Pendahuluan :

1. Salah satu metode mendesain IIR Filter dari Analog Filter adalah metode Transformasi Bilinear. Jelaskan secara singkat metode tsb dan berikan ilustrasi !
2. Jelaskan deskripsi dan parameter pada syntax MATLAB berikut
 $[n,Wn] = \text{buttord}(Wp,Ws,Rp,Rs)!$
3. Jelaskan apa yang dimaksud dengan white noise dan sertakan spektrum frekuensinya! Mengapa pengujian filter banyak dilakukan dengan white noise?
4. Jelaskan apa yang dimaksud dengan filter IIR.

3. PERCOBAAN

Diinginkan sebuah Band Pass Filter IIR yang mempunyai spesifikasi sebagai berikut : dapat meloloskan komponen sinyal berfrekuensi 2000 Hz sampai 6000 Hz (*passband frequency*), atenuasi yang terjadi pada frekuensi 400 Hz dan 14000 Hz minimal 20 dB, dan *ripple passband* maksimal 1 dB.

Filter ini akan diimplementasikan dengan board DSP EZ-Lite Kit BF-561. Diketahui sampling rate dari board ini adalah 48000 Hz. Lakukan langkah-langkah berikut pada proyek percobaan ini:

1. Desain Filter Band Pass ini dengan filter IIR dengan pendekatan Transformasi Bilinier dengan filter Butterworth dengan menggunakan MATLAB.
2. Buat source code program untuk Filter Band Pass IIR ini untuk EZ-Lite Kit BF-561 dengan memodifikasi program Talkthrough_TDM_C pada modul 3 atau program filter FIR pada modul 4
3. Uji filter yang telah dibuat dengan menggunakan White noise sebagai input. White noise mempunyai daya sama pada semua komponen frekuensi. White noise ini bisa dihasilkan dengan program Audacity
4. Rekam keluaran filter dengan program Audacity. Periksa spektrum sinyal hasil rekaman. Gunakan hasil ini sebagai konfirmasi apakah filter Band Pass telah berfungsi sesuai dengan yang diinginkan. Seharusnya dihasilkan sinyal dengan spektrum dengan daya yang sama pada rentang frekuensi pass band nya.

...