



دانشگاه تهران

پردیس دانشکده های دانشکده فنی

دانشکده مهندسی برق و کامپیووتر

دستور کار آزمایشگاه پردازش بی درنگ سیگنال های دیجیتال

ویراست ۳,۱

تیر

۱۳۹۹

کلیه حقوق مادی و معنوی این اثر متعلق به آزمایشگاه سیستم‌های پیشرفته مخابرات سیار، دانشکده مهندسی برق و کامپیوتر، پردیس دانشکده های فنی، دانشگاه تهران می باشد و هر گونه کپی برداری از آن ممنوع بوده و پیگرد قانونی دارد.

فهرست مطالب

۱	پیشگفتار	۰
۷	۱ معرفی بستر آزمایشگاهی	۱
۲۱	۲ آزمایش اول: آشنایی با نرم افزار Code Composer Studio	۲
۲۲	۲-۱ هدف	۱-۲
۲۲	۲-۲ مقدمه	۲-۲
۲۳	۳-۱ آزمایش	۳-۲
۲۳	۳-۲ تجهیزات مورد استفاده	۱-۳-۲
۲۳	۳-۲ قسمت اول: ایجاد یک پروژه در نرم افزار CCS	۲-۳-۲
۳۰	۳-۲ قسمت دوم: ابزارهای خطایابی	۳-۳-۲
۳۷	۴-۲ قسمت سوم: GEL File	۴-۳-۲
۳۸	۵-۳-۲ قسمت چهارم: ورود و خروج داده از فایل	۵-۳-۲
۴۱	۶-۳-۲ قسمت پنجم: فایل memory map (اختیاری)	۶-۳-۲
۴۱	۷-۳-۲ قسمت ششم: استک (اختیاری)	۷-۳-۲
۴۳	۴-۲ ضمایم	۴-۲
۴۳	۱-۴-۲ ساختار کلی فایل اسمبلي	۱-۴-۲
۴۵	۲-۴-۲ فایل linker command	۲-۴-۲
۴۸	۳ آزمایش دوم: نمونه برداری و تولید سیگنال آنالوگ در DSK6713	۳
۴۹	۱-۳ هدف	۱-۳

۴۹	مقدمه	۲-۳
۵۱	توابع مورد استفاده در آزمایشگاه برای خواندن و نوشتن از کدک	۳-۳
۵۳	آزمایش	۴-۳
۵۳	تجهیزات مورد استفاده	۱-۴-۳
۵۳	قسمت اول: نمونه برداری و تولید سیگنال آنالوگ در بورد DSK 6713	۲-۴-۳
۵۵	قسمت دوم: تولید سیگنال با استفاده از وقفه پورت سریال	۳-۴-۳
۵۶	قسمت سوم: تولید سیگنال سینوسی با polling	۴-۴-۳
۵۸	تمرین	۵-۳
۵۹	ضمایم	۶-۳
۵۹	استفاده از توابع CSL	۱-۶-۳
۶۱	پورت سریال McBSP	۲-۶-۳
۶۳	مشخصات کدک AIC23	۳-۶-۳
۶۶	وقفه	۴-۶-۳
۷۳	مراحل خواندن و نوشتן در کدک بورد C6713 DSK	۵-۶-۳

۴ آزمایش سوم: طراحی و پیاده سازی فیلتر FIR و استفاده از توابع پردازش

۷۶	سیگنال TI	
۷۷	هدف	۱-۴
۷۷	مقدمه	۲-۴
۷۷	تئوری	۳-۴
۸۲	فیلتر کردن با استفاده از توابع فیلتر TI	۴-۴
۸۴	آزمایش	۵-۴
۸۴	تجهیزات مورد استفاده	۱-۵-۴
۸۴	قسمت اول: حذف تون از سیگنال صوتی	۲-۵-۴
۸۵	قسمت دوم: طراحی و پیاده سازی فیلتر FIR به کمک بافر سیرکولار	۳-۵-۴
۸۷	قسمت دوم: استفاده از توابع ریاضی بهینه شده TI	۴-۵-۴

۸۹.....	ضمایم.....	۶-۴
۸۹.....	برگه اطلاعاتی تابع DSPF_sp_fir_gen.....	۱-۶-۴
۸۹.....	لیست توابع کتابخانه پردازش سیگنال TI.....	۲-۶-۴
۵ آزمایش چهارم: طراحی و پیاده سازی فیلترهای IIR.....		
۹۳.....	هدف.....	۱-۵
۹۳.....	مقدمه.....	۲-۵
۹۳.....	تئوری.....	۳-۵
۹۴.....	تحقیق فیلتر IIR.....	۱-۳-۵
۹۹.....	آزمایش.....	۴-۵
۹۹.....	تجهیزات مورد استفاده.....	۱-۴-۵
۹۹.....	قسمت اول: پیاده سازی فیلتر IIR.....	۲-۴-۵
۱۰۰.....	قسمت دوم: سیستم تولید پژواک.....	۳-۴-۵
۱۰۱.....	قسمت سوم: مولد سیگنال سینوسی (اختیاری).....	۴-۴-۵
۶ آزمایش پنجم: تبدیل فوریه (FFT) و تخمین طیف.....		
۱۰۴.....	هدف.....	۱-۶
۱۰۴.....	مقدمه.....	۲-۶
۱۰۴.....	تئوری.....	۳-۶
۱۰۸.....	آزمایش.....	۴-۶
۱۰۸.....	تجهیزات مورد استفاده.....	۱-۴-۶
۱۰۸.....	قسمت اول: FFT.....	۲-۴-۶
۱۰۹.....	قسمت دوم: اسپکتروم آنالایزر.....	۳-۴-۶
۱۱۲.....	قسمت سوم: استفاده از توابع پردازش سیگنال TI (اختیاری).....	۴-۴-۶
۷ آزمایش ششم: ممیز ثابت (fixed-point).....		
۱۱۳.....	آزمایشگاه سیستمهای پیشرفته مخابرات سیار - دانشگاه تهران	آزمایشگاه آموزشی پردازش بی درنگ سیگنالهای دیجیتال

۱۱۴.....	هدف.....	۱-۷
۱۱۴.....	مقدمه.....	۲-۷
۱۱۵.....	تئوری.....	۳-۷
۱۱۵.....	نمایش اعداد.....	۱-۳-۷
۱۱۶.....	محاسبات صحیح.....	۲-۳-۷
۱۱۷.....	خطای کوانتیزاسیون.....	۳-۳-۷
۱۱۸.....	خطای محاسبات.....	۴-۳-۷
۱۱۹.....	آزمایش.....	۴-۷
۱۱۹.....	تجهیزات مورد استفاده.....	۱-۴-۷
۱۱۹.....	قسمت اول: پیاده سازیتابع کسینوس به صورت fixed-point	۲-۴-۷
۱۲۲.....	قسمت دوم: پیاده سازیتابع رادیکال به صورت fixed-point	۳-۴-۷
۱۲۳.....	قسمت سوم: پیاده سازی فیلتر IIR به صورت fixed-point (اختیاری)	۴-۴-۷
۱۲۵.....	ضمیمه.....	۵-۷
۱۲۵.....	ابزار MATLAB نرم افزار Fixed-point	۱-۵-۷

۱۲۶.....	آزمایش هفتم: ارتباط MATLAB با CCS و پردازنده.....	۸
۱۲۷.....	هدف.....	۱-۸
۱۲۷.....	مقدمه.....	۲-۸
۱۲۸.....	تئوری.....	۳-۸
۱۲۸.....	نرم افزار Embedded IDE link	۱-۳-۸
۱۲۹.....	نرم افزار Target Support packag	۲-۳-۸
۱۳۰.....	آزمایش.....	۴-۸
۱۳۰.....	تجهیزات مورد استفاده.....	۱-۴-۸
۱۳۱.....	قسمت اول: ارتباط MATLAB با CCS	۲-۴-۸
۱۳۳.....	قسمت دوم: پردازنده در حلقه (PIL)	۳-۴-۸
۱۳۸.....	قسمت چهارم: ایجاد پروژه کامل از MATLAB برای CCS (اختیاری)	۴-۴-۸

۱۴۲.....	ضمایم	۵-۸
۱۴۳.....	ضمایم	۹
۱۴۳.....	لیست دستورات پردازنده‌های سری C6000	۱-۹
۱۴۵.....	لیست دستورات اعشاری پردازنده‌های سری C67X	۲-۹
۱۴۷.....	مراجع	۱۰

فهرست شکلها

..... ۱۰	شکل ۱-۱ تصویر بورد TMS320C6713 DSK
..... ۱۰	شکل ۲-۱ بلوک دیاگرام بورد TMS320C6713 DSK
..... ۱۱	شکل ۳-۱ بلوک دیاگرام پردازنده TMS320C6713
..... ۱۸	شکل ۴-۱ تنظیم compatibility mode برای استفاده از نرم افزار CCS v3.3 در ویندوز ۷
..... ۱۹	شکل ۵-۱ تصویر نرم افزار Scope
..... ۲۴	شکل ۱-۲ شمای نرم افزار Code Composer Studio Setup
..... ۲۴	شکل ۲-۲ انتخاب بورد مورد نظر در CCS Setup
..... ۲۴	شکل ۳-۲ نمای نرم افزار Code Composer Studio
..... ۲۵	شکل ۴-۲ پنجره ایجاد پروژه جدید
..... ۲۵	شکل ۵-۲ کد اسembly برای ذخیره تعدادی داده در حافظه
..... ۲۶	شکل ۶-۲ فایل ساده c
..... ۲۶	شکل ۷-۲ فایل linker command برای آزمایش اول
..... ۲۸	شکل ۸-۲ تنظیمات کامپایلر در قسمت Build Options
..... ۲۹	شکل ۹-۲ نمای پنجره های مختلف در ارتباط با پروژه lab1
..... ۳۰	شکل ۱۰-۲ پنجره Memory و پنجره تنظیمات آن
..... ۳۱	شکل ۱۱-۲ پنجره های Graph و Graph property Dialog متناظر با آن
..... ۳۱	شکل ۱۲-۲ C، تخصیص آدرس به اشاره گر
..... ۳۳	شکل ۱۳-۲ C تغییر یافته برای تعیین تعداد کلاک اجرای یک تابع با استفاده از قابلیتهای نرم افزار CCS
..... ۳۴	شکل ۱۴-۲ کد تابع sum_ به زبان اسembly
..... ۳۵	شکل ۱۵-۲ C با فراخوانی تابع اسembly
..... ۳۵	شکل ۱۶-۲ تغییر در فایلهای initmem.asm و main.c برای دسترسی به آدرس شروع محل قرار گرفتن داده ها در حافظه
..... ۳۸	شکل ۱۷-۲ مکان gel فایل
..... ۳۹	شکل ۱۸-۲ پنجره Breakpoint Manager

۳۹ شکل ۱۹-۲ انتخاب عملکرد break point
۳۹ شکل ۲۰-۲ پنجره پارامترها برای خواندن از فایل با استفاده از قابلیت break point
۳۹ شکل ۲۱-۲ پنجره پارامترها برای نوشتمن قسمتی از حافظه در یک فایل با استفاده از قابلیت break point
۴۰ شکل ۲۲-۲ یک نمونه فایل متنداده برای خوانده شدن توسط قابلیت break point
۴۰ شکل ۲۳-۲ کد نمونه برای ایجاد یک فایل از MATLAB
۴۱ شکل ۲۴-۲ کد تمرین ۲
۴۹ شکل ۱-۳ پردازش دیجیتال سیگنال آنالوگ
۶۳ شکل ۳-۳ بلوک دیاگرام کلی فرستنده پورت McBSP
۶۳ شکل ۴-۳ بلوک دیاگرام کلی گیرنده پورت McBSP
۶۵ شکل ۵-۳ بلوک دیاگرام کدک TLV320AIC23
۶۶ شکل ۶-۳ اولویت وقفه های CPU
۶۷ شکل ۷-۳ لیست منابع وقفه و نحوه عملکرد وقفه
۶۷ شکل ۸-۳ رجیسترها کنترلی وقفه
۷۹ شکل ۹-۳ جدول IST و یک Inerrupt Service Fetch Packet (ISFP)
۷۹ شکل ۱۰-۳ Interrupt Service Table Pointer (ISTP)
۷۰ شکل ۱۱-۳ مثال، تغییر مکان جدول IST در حافظه
۷۳ شکل ۱۲-۳ کد اسembly برای تعریف IST
۷۳ شکل ۱۳-۳ تابع برای انجام برخی موارد مرتبط با وقفه
۷۳ شکل ۱۴-۳ تابع ISR
۷۹ شکل ۱۵-۴ تحقق Type I direct form FIR برای فیلتر
۸۰ شکل ۲-۴ شمای ابزار fdatool
۸۱ شکل ۳-۴ قسمتی از کد برای فیلتر FIR بر مبنای تحقق مستقیم
۸۲ شکل ۴-۴ قسمتی از کد برای فیلتر FIR بر مبنای تحقق مستقیم و با پیاده سازی بر مبنای ایده سیرکولار بافر با کد C
۸۳ شکل ۵-۴ محل وارد کردن آدرس فایل های header
۸۵ شکل ۶-۴ مشخصات فیلتر پایین گذر با فرکانس نمونه برداری 24KHz

..... ۹۵	شکل ۱-۵ ساختار تحقق مستقیم نوع ۱ برای فیلتر IIR
..... ۹۶	شکل ۲-۵ ساختار تحقق مستقیم نوع ۲ برای فیلتر IIR
..... ۹۷	شکل ۳-۵ ساختار تحقق مستقیم نوع ۲ ترانسپوز برای فیلتر IIR
..... ۹۷	شکل ۴-۵ ساختار تحقق سری فیلتر IIR
..... ۹۸	شکل ۵-۵ تحقق فیلتر IIR مرتبه چهار با دو بخش با تحقق مستقیم نوع ۲
..... ۹۸	شکل ۶-۵ ساختار تحقق موازی فیلتر IIR
..... ۹۹	شکل ۷-۵ قسمتی از کد C برای پیاده سازی فیلتر IIR بر مبنای تحقق مستقیم نوع ۲ پیاده سازی شده است
..... ۱۰۰	شکل ۸-۵ پنجره Export ابزار fdatool
..... ۱۰۱	شکل ۹-۵ بلوک دیاگرام سیستم تولید اکو
..... ۱۰۶	شکل ۱-۶ اندازه تبدیل فوریه پنجره مستطیلی به طول $N=10$
..... ۱۱۲	شکل ۲-۶ تصویر طیف سیگنالهای سینوسی و مربعی با فرکانس تکرار 500 Hz
..... ۱۱۵	شکل ۱-۷ تقسیم بندهای بیتهای نمایش عدد حقیقی ۳۲ بیتی
..... ۱۲۰	شکل ۲-۷ پیاده سازی تابع کسینوس به روش اول و به صورت محاسبات ممیز شناور
..... ۱۲۱	شکل ۳-۷ پیاده سازی تابع کسینوس به روش دوم و به صورت محاسبات ممیز شناور
..... ۱۲۱	شکل ۴-۷ پیاده سازی تابع کسینوس به صورت محاسبات ممیز ثابت
..... ۱۲۹	شکل ۲-۸ ترکیب نرم افزارهای MATLAB، Simulink و Real-Time Workshop و Embedded IDE برای ایجاد محیطی جهت تولید، خطایابی و تست کد برای پردازنده‌های سیگنال (DSP) و میکروکنترلرهای (MCU)
..... ۱۳۰	شکل ۳-۸ ترکیب Target Support و Embedded IDE Link و Real-Time Workshop و Simulink برای طراحی، شبیه سازی و پیاده سازی الگوریتمهای پردازش سیگنال بر روی پردازنده‌های مختلف
..... ۱۳۳	شکل ۴-۸ بلوک دیاگرام شبیه سازی برای تخمینگر طیف در simulink
..... ۱۳۶	شکل ۵-۸ مدل سیمولینک برای شروع برای پیاده سازی تخمینگر طیف بر روی پردازنده به صورت PIL
..... ۱۳۶	شکل ۶-۸ تنظیم پارامترهای بلوک Target Preference بر مبنای بورد مورد استفاده
..... ۱۳۷	شکل ۷-۸ زیر مدل بلوکی جهت اجرا بر روی پردازنده

شکل ۸-۸ پیاده سازی پردازنده در حلقه برای تخمین گر طیف و مقایسه نتیجه اجرا بر روی پردازنده و بر روی PC.....	۱۳۷
شکل ۹-۸ بلوک دیاگرام یک سیستم حذف نویز برای پیاده سازی بر روی پردازنده.....	۱۳۹
شکل ۱۰-۸ محل برخی بلوکهای موجود در کتابخانه Target Support Package برای کار با بورد C6713 DSK.....	۱۳۹
شکل ۱۱-۸ بلوک Target Support Package از کتابخانه Target Prefence.....	۱۴۰
شکل ۱۲-۸ تنظیم پارامترهای بلوک ADC برای بورد C6713 DSK.....	۱۴۰
شکل ۱۳-۸ تنظیم پارامترهای بلوک سوئیچ برای بورد C6713 DSK.....	۱۴۰

فهرست جدولها

عنوان	صفحه
جدول ۱-۰ لیست آزمایش‌های آزمایشگاه پردازش بی‌درنگ سیگنال‌های دیجیتال به همراه اهداف مرتبط با هر کدام	۴
جدول ۲-۰ زمان بندی آزمایشگاه پردازش بی‌درنگ سیگنال‌های دیجیتال	۶
جدول ۱-۱ دستورات محاسبات صحیح مشترک بین پردازنده‌های خانواده‌های C62x و C67x	۱۳
جدول ۱-۲ دستورات محاسبات اعشاری برای پردازنده‌های C67x	۱۴
جدول ۱-۳ مودهای آدرس دهی	۱۵
جدول ۱-۴ نقشه حافظه برای TMS320C6713	۱۵
جدول ۱-۵ انواع داده در پردازنده‌های سری C6000	۱۷
جدول ۱-۶ مقایسه تعداد کلاکها برای build های مختلف	۳۶
جدول ۱-۷ نرخهای نمونه برداری مجاز برای کدک AIC23	۶۳
جدول ۱-۸ لیست توابع کتابخانه TI برای پردازنده‌های TMS320C67X	۸۹
جدول ۱-۹ نتیجه تست پیاده سازی‌های مختلف برای تابع کسینوس	۱۲۲
جدول ۱-۱۰ تعداد کلاک برای اجرای پیاده سازی‌های مختلف برای تابع کسینوس بر روی دو پردازنده	۱۲۲
جدول ۱-۱۱ نتیجه تست پیاده سازی تابع رادیکال به صورت fixed-point	۱۲۳
جدول ۱-۱۲ تعداد کلاک لازم برای اجرای تابع رادیکال بر روی پردازنده TMS320C6416	۱۲۳
جدول ۱-۱۳ لیست آرگومانهای ورودی برای تابع quantizer در MATLAB	۱۲۵
جدول ۱-۱۴ لیست متودهای موجود برای شیئ ticcs برای ارتباط با نرم افزار CCS در MATLAB	۱۴۲

۰ پیشگفتار

درس پردازش سیگنالهای دیجیتال یکی از درس‌های مهم و پرکاربرد است که در زمینه‌های گوناگون مهندسی و گرایشهای مختلف رشته مهندسی برق به خصوص مخابرات به کار می‌آید. به این منظور داشتن آزمایشگاهی در کنار این درس می‌تواند ضمن عمق بخشیدن به یادگیری مباحث این زمینه، دانشجویان را برای رویارویی با چالشهای عملی و پیاده سازی این مبحث علمی آشنا کند. به این ترتیب طراحی آزمایشگاه آموزشی پردازش بی‌رنگ سیگنالهای دیجیتال در چارچوب طرح "تأسیس آزمایشگاه سیستمهای پیشرفته مخابرات سیار و دوره‌های آموزشی مربوطه" با پشتیبانی شرکت ارتباطات سیار ایران (همراه اول) در خردادماه ۱۳۸۹ از سوی کمیته راهبری طرح در دستور کار قرار گرفت. این طرح در همراه اول از سوی آقایان مهندس صدوqi (مدیرعامل)، دکتر نیکوفر (قائم مقام مدیرعامل) و دکتر سید موسوی (مدیرکل دفتر تحقیقات) هدایت شده است. اعضای کمیته راهبری طرح در دانشگاه تهران شامل آقایان دکتر کمره‌ای (رئیس طرح)، دکتر جبه دار (مدیر طرح)، دکتر راشد محصل، دکتر نادر اصفهانی، دکتر شاه‌آبادی، دکتر فرجی دانا و دکتر لاهوتی (مدیر اجرایی طرح) بوده‌اند. همکاران طرح شامل آقایان دکتر جلیل سیفعلی هرسینی، مهندس محمد امیررحمت، مهندس امین خان‌سفید، مهندس حمیدرضا ارجمندی، مهندس بهزاد اسدی، مهندس افشین عبدالی، مهندس سیاوش قوامی، مهندس رضا پارسه، مهندس احسان ولی و خانم‌ها مهندس حکیمه پورمهدی و مهندس هما حسینمردی بوده‌اند. پروژه طراحی مفاد آزمایشگاه آموزشی پردازش بی‌رنگ سیگنالهای دیجیتال به مدیریت آقای دکتر لاهوتی و با مشارکت آقایان مهندس امین خان‌سفید و مهندس افشین عبدالی به عنوان همکاران اصلی به انجام رسیده است.

برای آماده سازی دستور کار این آزمایشگاه ابتدا مطالعه‌ای تطبیقی بر روی آزمایشگاه‌های پردازش سیگنال در دانشگاه‌های مختلف انجام شد، همچنین تعدادی از کتابهای مرتبط با این مبحث بررسی گردیدند که عنوان تعدادی از آنها در قسمت مرجع این دستور کار آمده است. سپس دستور کاری که پیش رو دارید با توجه به نیازهای خود تهیه گردید.

آزمایشگاه پردازش سیگنال در دانشگاههای مختلف دنیا با رویکردهای متنوعی ارائه می‌گردد. برخی در کنار درس کلاسی به صورت تمرینهای کامپیوتری منسجم که عموماً با نرم‌افزار MATLAB انجام می‌شوند ارائه می‌گردد و برخی دیگر با استفاده از ابزارهای فیزیکی پردازش سیگنال، آزمایشگاه را پایه گذاری نموده‌اند، با ابزارهایی نظیر پردازنده‌های دیجیتال شرکت Texas Instrument یا پردازنده‌های دیجیتال شرکت Analog Devices. باز برای استفاده از این پردازنده‌ها روش‌های مختلفی پیش گرفته شده است. بعضی از نرم افزار MATLAB و Simulink استفاده می‌کنند و با استفاده از بلوکهای Simulink برای ارتباط با بورد سخت افزاری، الگوریتم پردازشی خود را بر روی پردازنده پیاده می‌کنند. تعدادی از آزمایشگاهها نیز بر پایه پیاده سازی الگوریتم پردازشی بر روی بوردهای شرکت TI با برنامه نویسی این پردازنده‌ها به زبان C و اسمنبلی پردازنده مورد نظر، در محیط نرم افزاری به نام CCS پایه گذاری شده‌اند. در این آزمایشگاه رویکرد اصلی پیاده سازی الگوریتم‌های پردازش سیگنال بر روی بوردهای شرکت TI انتخاب شد. به این ترتیب دانشجویان ضمن رویارویی با چالشهای پیاده سازی الگوریتم‌های پردازش سیگنال با مباحث تئوری درس نیز بیشتر و عمیق‌تر آشنا می‌شوند. همچنین با توجه به آنکه پردازنده‌های شرکت TI از پرفروش‌ترین و پر مصرف‌ترین پردازنده‌های سیگنالهای دیجیتال در دنیا می‌باشند، در ضمن این آزمایشگاه، این رویکرد دانشجویان را برای کار عملی و ورود به بازار کار آمده می‌کند.

در این آزمایشگاه کامپیوتر، نرم افزار MATLAB و نرم افزار Code Composer Studio مورد استفاده قرار می‌گیرند. نرم افزار اخیر برای برنامه نویسی پردازنده‌های شرکت TI توسط این شرکت به بازار عرضه شده است. همچنین بسترهای سخت افزاری بورد DSK TMS320C6713 که حاوی یک fixed-point پردازنده TMS320C6416 DSK است در این آزمایشگاه می‌باشد و بورد TMS320C6416 DSK که حاوی یک پردازنده floating-point است در این آزمایشگاه در دسترس می‌باشند. وجود ابزارهای فانکشن ژنراتور و اسیلوسکوپ بر جنبه های عملی این آزمایشگاه افزده است.

در این آزمایشگاه دانشجویان با مباحث مربوط به ساختار داخلی پردازنده‌های دیجیتال در حد مختصر اما به اندازه‌ای که بتوانند با این پردازنده‌ها کار کنند آشنا می‌شوند. در فصل اول مقدماتی جهت

مطالعه و کسب اطلاعات پیش نیاز برای کار با بورد و پردازنده و تجهیزات مورد استفاده در آزمایشگاه ارائه می‌شود. در آزمایش اول ابزار برنامه نویسی و خطایابی CCS معرفی می‌شود. در آزمایش دوم دانشجویان با مباحث سخت افزاری نظری و قفه و پورت سریال آشنا می‌شوند و همچنین مباحث مرتبط با برنامه نویسی پردازنده‌های TI ارائه می‌گردد. در قسمت عملی این بخش دانشجویان با نمونه برداری و بازسازی سیگنال در بورد DSK 6713 آشنا می‌شوند و همچنین یک فانکشن ژنراتور باند صوتی می‌سازند. یکی از مباحث مهم پردازش سیگنال فیلترینگ است. در طی دو آزمایش دانشجویان به پیاده FIR سازی فیلتر FIR و IIR می‌پردازند. در مبحث فیلتر FIR دانشجویان بعد از آنکه الگوریتم فیلترینگ را با دو روش عادی و روشی بر مبنای ایده سیرکولار بافر پیاده نمودند، می‌آموزنند چگونه از توابع کتابخانه پردازش سیگنال TI که با توجه به ساختار داخلی پردازنده‌های این شرکت به صورت بهینه نوشته شده‌اند استفاده کنند. در فصل مربوط به پیاده سازی فیلتر IIR دانشجویان یک فیلتر IIR طراحی می‌کنند و سپس آن را بر روی پردازنده پیاده می‌کنند. در خلال این آزمایشها لازم است برخی دستورهای MATLAB نیز به کار گرفته شود که خود آموزنده هستند. در ادامه در آزمایش بعدی یک اسپکتروم آنالایزر دیجیتال در محیط نرم افزار CCS با استفاده از بورد پردازنده می‌سازند. دانشجویان ضمن کار در این آزمایشگاه با چالشهایی مثل پیاده سازی بی‌درنگ الگوریتمها، که مرتبط با پیاده سازی با پیچیدگی محاسباتی کم، می‌باشد، رو برو می‌گردد. در آزمایش بعدی بحث مهم پیاده سازی الگوریتمها به صورت ممیز ثابت مطرح است. محاسبات توابع حقیقی به صورت fixed-point و پیاده سازی الگوریتم فیلترینگ به صورت fixed-point در این آزمایش مطرح هستند. در آزمایش بعدی چگونگی برقرار کردن لینک ارتباطی نرم افزار MATLAB با نرم افزار CCS و تولید کد و ایجاد پروژه‌های پردازنده در حلقه بررسی می‌گردد. لیست آزمایشها این آزمایشگاه به همراه اهداف مرتبط چه از نظر آموزش ابزار پیاده سازی و چه از نظر اهداف تئوری در جدول ۱-۰ آمده‌اند.

بعد از انجام این آزمایشها دانشجویان قادر هستند به انجام پروژه‌های کاربردی مختلف به صورت مستقل بپردازند.

امید است این آزمایشگاه موجب یادگیری مباحث پردازش سیگنال با رویکرد پیاده سازی

الگوریتمهای مربوطه و عمق بخشنیدن بیشتر به آموخته‌های دانشجویان گردد و در آماده سازی ایشان در کارهای عملی و صنعتی کمک کند.

جدول ۱-۰ لیست آزمایشهای آزمایشگاه پردازش بی‌رنگ سیگنالهای دیجیتال به همراه اهداف مرتبط با هر کدام

نام آزمایش	اهداف مرتبط با پیاده‌سازی	اهداف مرتبط با مباحث تئوری
معرفی بستر آزمایشگاهی	<ul style="list-style-type: none"> ● آشنایی با پردازنده‌های DSP ● تجهیزات مورد استفاده در آزمایشگاه ● معرفی بورد TMS320C6713 DSK و نحوه استفاده از آن ● معرفی پردازنده TMS320C6713 ● لیست دستورات اسembلی پردازنده ● روش‌های آدرس دهی ● ارائه اطلاعات عمومی شامل نقشه حافظه و دسترسی به آن، کتابخانه <code>BSL</code>، انواع داده، <code>#pragma</code> 	<ul style="list-style-type: none"> ● آشنایی با پردازنده‌های ● تجهیزات مورد استفاده در آزمایشگاه ● معرفی بورد TMS320C6713 DSK و نحوه استفاده از آن ● معرفی پردازنده TMS320C6713 ● لیست دستورات اسembلی پردازنده ● روش‌های آدرس دهی ● ارائه اطلاعات عمومی شامل نقشه حافظه و دسترسی به آن، کتابخانه <code>BSL</code>، انواع داده، <code>#pragma</code>
آشنایی با نرم افزار Code Composer Studio (CCS)	<ul style="list-style-type: none"> ● مشاهده محتوا حافظه و رجیسترها ● Breakpoint ● خواندن و نوشتمن از فایل با استفاده از breakpoint ● نمایش گرافیکی داده‌ها ● مشاهده مقادیر متغیرها ● تخصیص اشاره گر به آدرس مشخصی از حافظه ● تعیین تعداد کلاک الگوریتم ● بهینه سازی کد در CCS ● ساختار فایل اسembلی ● linker command ● فایل memory map ● فایل (stack) 	<ul style="list-style-type: none"> ● آشنایی با قابلیتهای نرم افزار CCS شامل: ● ایجاد پروژه ● ابزارهای خطایابی ○ مشاهده محتوا حافظه و رجیسترها ○ Breakpoint ○ خواندن و نوشتمن از فایل با استفاده از breakpoint ○ نمایش گرافیکی داده‌ها ○ مشاهده مقادیر متغیرها ● تخصیص اشاره گر به آدرس مشخصی از حافظه ● تعیین تعداد کلاک الگوریتم ● بهینه سازی کد در CCS ● ساختار فایل اسembلی ● linker command ● فایل memory map ● فایل (stack)
نمونه برداری و تولید سیگنال آنالوگ در بورد	<ul style="list-style-type: none"> ● نمونه برداری و تولید سیگنال ● تداخل طیفی (aliasing) 	<ul style="list-style-type: none"> ● Chip support library (CSL) ● پورت سریال McBSP

	<ul style="list-style-type: none"> • کدک AIC23 • وقفه • توابع مربوط به خواندن و نوشتن از کدک GEL • تولید انواع سیگنال (با روش محاسبه نمونه سیگنال، با روش خواندن از جدول) 	C6713 DSK
<ul style="list-style-type: none"> • معرفی فیلتر FIR • طراحی فیلتر FIR با تعیین مکان صفر تابع انتقال 	<ul style="list-style-type: none"> • سیرکولاو بافر • پیاده سازی فیلتر FIR با کد C • استفاده از توابع پردازش سیگنال TI • استفاده ازتابع DSPF_sp_fir_gen() • تست برنامه در CCS و مقایسه نتیجه با نتایج در MATLAB • ابزار fdatool • رسم طیف سیگنال در MATLAB 	طراحی و پیاده سازی فیلترهای FIR
<ul style="list-style-type: none"> • معرفی فیلتر IIR • ساختارهای تحقق فیلتر IIR 	<ul style="list-style-type: none"> • پیاده سازی فیلتر IIR با کد C • پیاده سازی سیستم تولید پژواک • پیاده سازی مولد سیگنال سینوسی دیجیتال 	طراحی و پیاده سازی فیلترهای IIR
<ul style="list-style-type: none"> • تبدیل فوریه گسسته زمان • آشنایی با تناوبنگار و تخمین طیف 	<ul style="list-style-type: none"> • پیاده سازی یک اسپکتروم آنالایزر دیجیتال 	تبدیل فوریه(FFT) و تخمین طیف
<ul style="list-style-type: none"> • محاسبات صحیح (ممیز ثابت) • محاسبه تابع کسینوس و رادیکال به صورت ممیز ثابت 	<ul style="list-style-type: none"> • برنامه نویسی ممیز ثابت • پیاده سازی فیلتر IIR به صورت ممیز ثابت 	ممیز ثابت
<ul style="list-style-type: none"> • پیاده سازی آشکار سازی DTMF سیگنال • تخمین گر طیف • حذف نویز 	<ul style="list-style-type: none"> • ارتباط CCS با MATLAB • تولید کد برای پردازنده با استفاده از MATLAB • پیاده سازی سخت افزار در حلقه با Embedded IDE Link Software • (MATLAB) • Target Support Package (MATLAB) 	ارتباط MATLAB با نرم افزار CCS و پردازنده

در ویرایش دوم این دستور کار با توجه به زمانی که در اختیار دانشجویان قرار می‌گیرد قسمتهایی از آزمایشها به صورت اختیاری در نظر گرفته شده‌اند که انجام آنها بعد از انجام قسمتهای دیگر در صورت داشتن زمان به صورت اختیاری ممکن است. به این ترتیب جدول ۲-۰ جلسات این آزمایشگاه و وظیفه دانشجویان در هر قسمت را نشان می‌دهد. همچنین در این ویرایش بخش‌هایی از متن ویراستاری و اصلاح شده‌اند.

جدول ۲-۰ زمان بندی آزمایشگاه پردازش بی‌رنگ سیگنالهای دیجیتال

فعالیت	جلسه
معرفی آزمایشگاه و تجهیزات و پردازنده DSP آزمایش اول - قسمت اول	۱
آزمایش اول - قسمتهای دوم تا چهارم	۲
آزمایش دوم - قسمتهای اول و دوم	۳
آزمایش سوم - قسمت اول	۴
آزمایش سوم - قسمتهای دوم و سوم	۵
آزمایش چهارم - قسمتهای اول و دوم	۶
آزمایش پنجم - قسمتهای اول و دوم	۷
آزمایش ششم - قسمتهای اول و دوم	۸
آزمایش هفتم - قسمتهای اول، سوم	۹

۱ معرفی بستر آزمایشگاهی

در این آزمایشگاه برای پیاده سازی الگوریتمهای پردازش سیگنال از بورد^۱ DSK6713 استفاده می‌شود. قلب این بورد پردازنده TMS320C6713 ساخت شرکت^۲ TI می‌باشد. در این بورد پردازنده C6713 با دنیای آنالوگ از طریق کدک استریو AIC23 ارتباط برقرار می‌کند و ارتباط این بورد با کامپیوتر از طریق پورت USB می‌باشد که در کامپیوتر بوسیله نرم افزار (CCS) Code Composer Studio (CCS) امکان برنامه‌نوشته شده به زبان C/C++ یا اسملی برای اجرا بر روی پردازنده کامپایل، لینک و ارسال الگوریتمهای مخصوص پردازنده‌های DSP را فراهم می‌نماید. پردازنده‌های DSP با ایجاد قابلیتهايی نظير روش‌های آدرس دهی سیرکولار و bit-reverse (که در زمینه پیاده سازی کارا بافر و محاسبه FFT کاربرد دارند) و ایجاد ساختارهای مناسب سخت افزاری و دستوری (نظير دستور^۳ MAC که در فیلترینگ سیگنال کاربرد دارد) بستری ارزان و با مصرف توان کم برای پردازش سیگنال‌های دیجیتال فراهم کرده‌اند که کاربردهای متنوعی از جمله استفاده در گوشی موبایل، مودم، دوربین دیجیتال، ... دارند. در حقیقت پردازنده‌های DSP معمولاً برای پردازش زمان حقیقی^۴ سیگنال به کار می‌روند.

پردازنده‌های سری C6000 پردازنده‌هایی برای کاربردهایی که محاسبات زیاد دارند مناسب می‌باشند. در این سری هم پردازنده‌های floating-point وجود دارد (C67x) و هم پردازنده‌های fixed-point (C62x, C64x). اولین پردازنده از این سری TMS320C6201 در سال ۱۹۹۷ معرفی شد.

ابزارهایی که در این آزمایشگاه مورد استفاده قرار می‌گیرند عبارتند از:

۱- بستر TMS320C6713 DSK شامل

a. بورد TMS320C6713 DSK

b. نرم افزار CCS

c. کابل USB برای اتصال بورد DSK به PC

d. آداپتور تغذیه +5V

¹ DSK: DSP Starter Kit

² Texas Instrument

³ Multiply and accumulate

⁴ Real-time

۲ - کامپیوتر PC

۳ - فانکشن ژنراتور مدل 33521A ساخت شرکت Agilent

۴ - اسیلوسکوپ مدل TDS2024B ساخت شرکت Tektronix

البته به جای این دو ابزار می‌توان از نرم افزارهایی روی PC و کارت صوتی PC استفاده کرد.

۵ - هدفون، میکروفون و کابلهای اتصال پورتهای Line in و Line out از PC به بورد و بر عکس.

شکل ۱-۱ تصویر بورد DSK و قسمتهای مختلف آن و شکل ۲-۱ بلوک دیاگرام این بورد را نشان

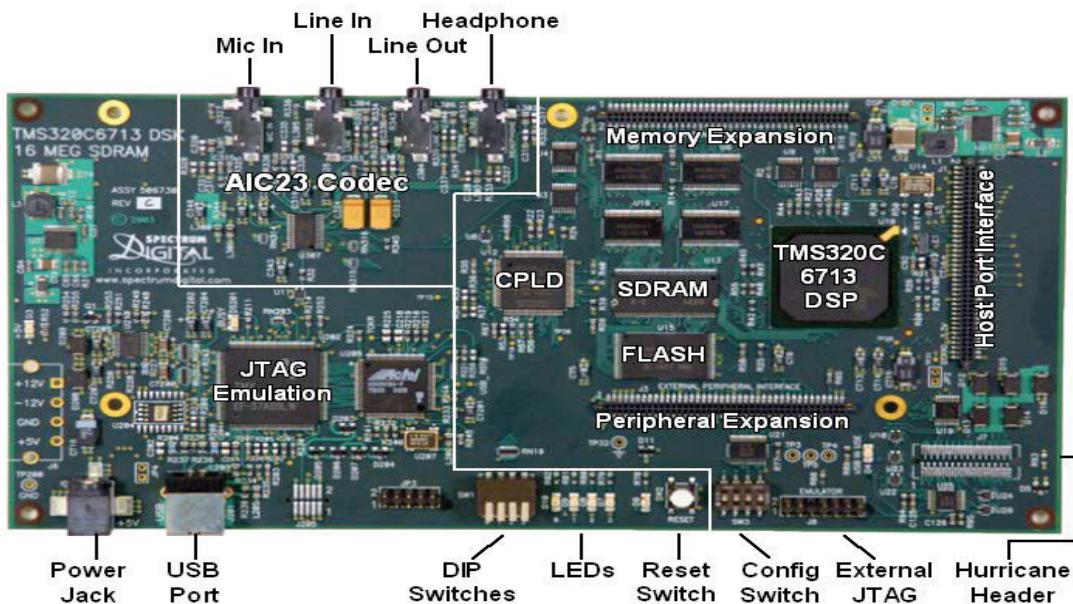
می‌دهند.

۶ - نرم افزار MATLAB

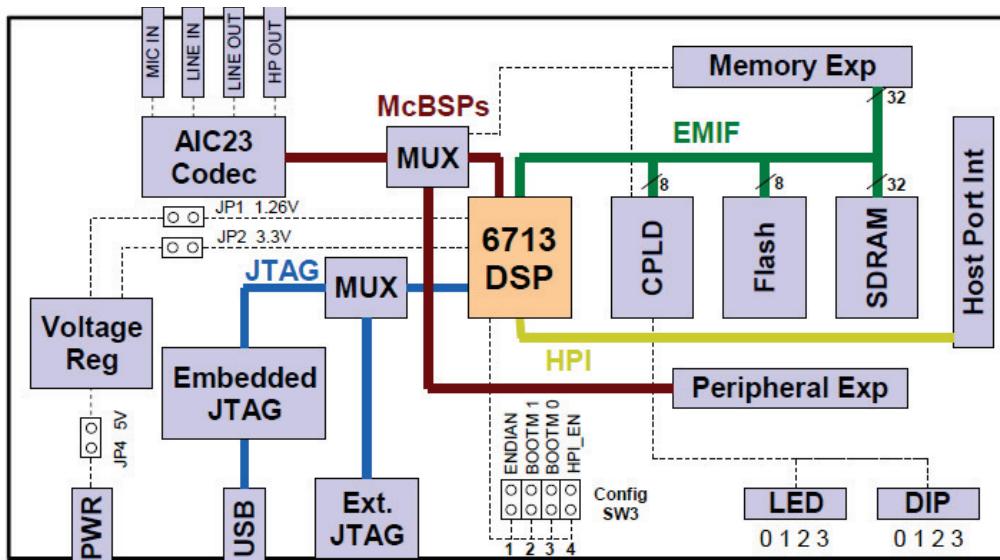
بورد DSK TMS320C6713 با ابعاد تقریبی ۸×۵ اینچ شامل قسمتهای زیر است:

- پردازنده TMS320C6713 : که ماکریم با فرکانس 225 MHz (هر سیکل دستور برابر ۰.4 ns) کار می‌کند.
- کدک AIC23 که به روش سیگما-دلتا قابلیت تبدیل سیگنال آنالوگ به سیگنال دیجیتال و بر عکس را با فرکانس‌های نمونه برداری مشخص از 8KHz تا 96 KHz فراهم می‌آورد.
- SDRAM^۱ 16 MB
- 512 KB حافظه فلاش
- دو کانکتور ۸۰ پین برای اتصال تجهیزات و حافظه خارجی
- دو کانکتور برای ورودی آنالوگ (MIC و Line In) و دو کانکتور برای خروجی آنالوگ (Headphone و Line Out)
- ۴ عدد DIP سوئیچ که وضعیت آنها به وسیله توابعی از کتابخانه dsk6713bsl.lib در برنامه قابل خواندن است.
- ۴ عدد LED که بوسیله توابعی از کتابخانه dsk6713bsl.lib در برنامه قابل کنترل هستند.
- JTAG Emulator

^۱ Synchronous Dynamic RAM



شکل ۱-۱ تصویر بورد TMS320C6713 DSK



شکل ۲-۱ بلوك دیاگرام بورد TMS320C6713 DSK

مراحل استفاده از بورد:

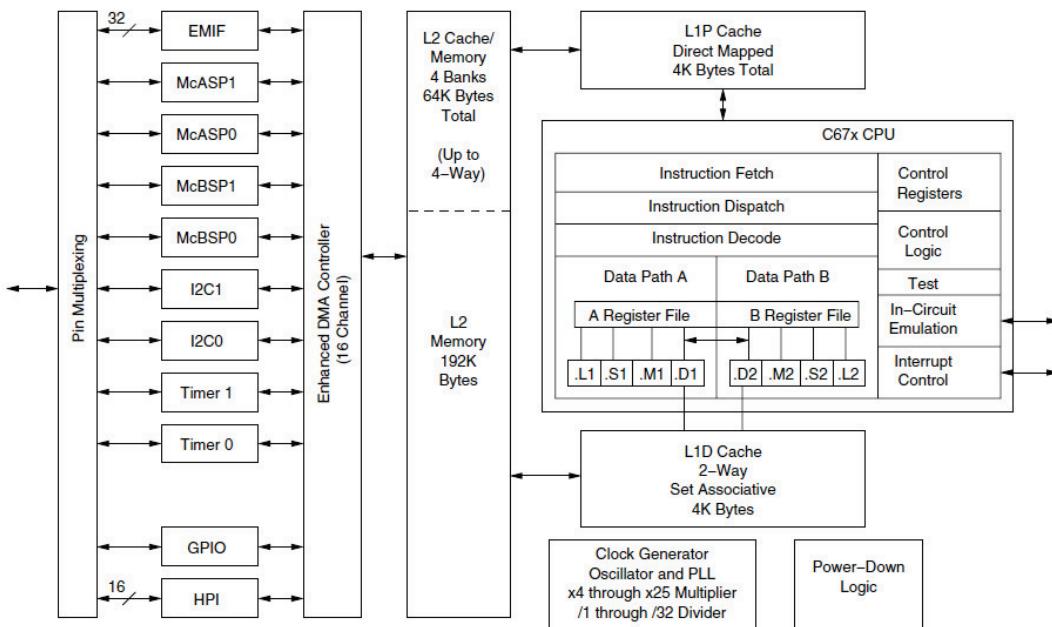
- ۱- کانکتورهای مربوط به MIC، Line In، Line Out، Headphone را به بورد وصل کنید.
- ۲- کابل USB را به بورد وصل کنید.
- ۳- سر دیگر کابل USB را به کامپیوتر وصل کنید.
- ۴- آداپتور تغذیه را به برق و سپس به بورد وصل کنید.
- ۵- نرم افزار CCS را روی PC اجرا کنید.

بعد از آنکه منبع تغذیه به بورد وصل شد، برنامه ای که در حافظه فلش بورد قرار دارد اجرا

می‌شود که قسمتهای مختلف بورد را تست می‌کند. اگر تستها موفقیت آمیز بود در نهایت چهار LED همزمان سه مرتبه چشمک می‌زنند و روشن باقی می‌مانند.

پردازنده TMS320C6713 یک پردازنده با کلمات^۱ ۳۲ بیتی است. به خاطر قابلیت انجام محاسبات اعشاری (floating-point) برای این پردازنده، در برنامه نویسی آن معمولاً مسئله سر ریز^۲ و رنج دینامیکی نمایش اعداد نگران کننده نخواهد بود در صورتی که اینها دو موضوع مهم در کار با پردازنده‌های fixed-point می‌باشند.

شکل ۱-۱ بلوک دیاگرام پردازنده TMS320C6713 را نشان می‌دهد. پردازنده TMS320C6713 دارای تجهیزات جانبی^۳ متعددی نظیر^۴ EDMA، پورت سریال (McBSP)، تایمر، I²C، HPI^۵ می‌باشد.



شکل ۱-۱ بلوک دیاگرام پردازنده TMS320C6713

ساختمان : CPU

¹ Word

² Over flow

³ Peripheral

⁴ Enhanced direct memory access

⁵ Host port interface

پردازنده C6713 دارای ساختار^۱ VLIW می‌باشد که TI آن را VelociTi CPU در پردازنده‌های سری C6000 دارای هشت واحد عملیاتی می‌باشد که به طور موازی کار می‌کنند و به طور یکسان به دو دسته A یا 1 و B یا 2 تقسیم می‌شوند. هر دسته شامل واحدهای زیر است:

M. واحد ضرب که قابلیت ضرب صحیح و اعشاری دارد.

L. برای عملیات منطقی و حسابی

S. برای پرش، محاسبات حسابی و دستکاری بیت^۲

D. برای محاسبات حسابی، عملیات ذخیره و بازیابی

به این ترتیب دستورات حسابی، منطقی و پرش در واحدهای S. و L. انجام می‌شوند و دستورات مربوط به انتقال داده در واحد D. انجام می‌شوند. این هشت واحد عملیاتی شامل چهار ALU صحیح و اعشاری (D.) و دو ALU صحیح (D.). و دو ضرب کننده صحیح و اعشاری (M.) می‌باشد.

همچنین پردازنده دارای ۳۲ رجیستر بیتی چند منظوره می‌باشد که به دو دسته مساوی A (A0-A15) و B (B0-B15) تقسیم می‌شوند که هر کدام مربوط به واحدهای عملیاتی دسته A و دسته B می‌شوند. هر دسته می‌تواند رجیسترها خود را در اختیار واحدهای عملیاتی دسته دیگر قرار دهد البته با این محدودیت که بین دو دسته A و B یک بار داده وجود دارد.

CPU هر بار هشت دستور ۳۲ بیتی را از حافظه می‌خواند که یک^۳ نامیده می‌شود. سپس این دستورات را برای اجرا به واحدهای عملیاتی می‌فرستد. دستورهایی که همزمان اجرا می‌شوند یک^۴ هستند.

دستورهای اسمبلي:

لیست دستورات اسمبلي برای محاسبات صحیح مرتبط با واحد های عملیاتی مختلف در جدول ۱-۱ آورده شده است. همچنین جدول ۲-۱ لیست دستورات اسمبلي برای محاسبات اعشاری متناظر با واحدهای عملیاتی مختلف پردازنده را نشان می‌دهد. در فصل ضمایم این لیستها به همراه توضیح

¹Very long instruction word

² Bit manipulation

³ Fetch packet

⁴ Execute packet

مختصری راجع به آنها تکرار می‌گردند. در این لیست دستورهایی که به^۱ DP ختم می‌شوند بر روی اعداد حقیقی با دقت مضاعف عمل می‌کنند و همین طور دستورهایی که به^۲ SP ختم می‌شوند به معنای محاسبه با آرگومانهای اعشاری ۳۲ بیتی هستند. جزئیات این دستورات در مراجع TMS320C6000 CPU TMS320C67x C67x+ DSP CPU and Instruction and Instruction Set Reference Guide [۴] و [۵] یافت می‌شوند.

جدول ۱-۱ دستورات محاسبات صحیح مشترک بین پردازنده‌های خانواده‌های C62X و C67x

.L unit	.M Unit	.S Unit	.D Unit
ABS	MPY	ADD	SET
ADD	MPYU	ADDK	SHL
ADDU	MPYUS	ADD2	SHR
AND	MPYSU	AND	SHRU
CMPEQ	MPYH	B disp	SSHLL
CMPGT	MPYHU	B IRP	SUB
CMPGTU	MPYHUS	B NRP	SUBU
CMPLT	MPYHSU	B reg	SUB2
CMPLTU	MPYHL	CLR	XOR
LMBD	MPYHLU	EXT	ZERO
MV	MPYHULS	EXTU	LDB (15-bit offset)
NEG	MPYHSLU	MV	LDBU (15-bit offset)
NORM	MPYLH	MVC	LDH (15-bit offset)
NOT	MPYLHU	MVK	LDHU (15-bit offset)
OR	MPYLUHS	MVKH	LDW (15-bit offset)
SADD	MPYLSHU	MVKLH	STB
SAT	SMPY	NEG	STH
SSUB	SMPYHL	NOT	STW
SUB	SMPYLH	OR	
SUBU	SMPYH		
SUBC			
XOR			
ZERO			

¹ Double precision² Single precision

جدول ۱-۲ دستورات محاسبات اعشاری برای پردازنده‌های C67x

.L Unit	.M Unit	.S Unit	.D Unit
ADDDP	MPYDP	ABSDP	ADDAD
ADDSP	MPYI	ABSSP	LDW
DPINT	MPYID	CMPEQDP	
DPSP	MPYSP	CMPEQSP	
DPTRUNC		CMPGTDP	
INTDP		CMPGTSP	
INTDPU		CMPLTDP	
INTSP		CMPLTSP	
INTSPU		RCPDP	
SPINT		RCPSP	
SPTRUNC		RSQRDP	
SUBDP		RSQRSP	
SUBSP		SPDP	

روشهای آدرس دهی:

برای مشخص کردن آدرس خانه ای از حافظه برای پردازنده‌های C67x روش خطی^۱ و سیرکولار وجود دارد. از همه رجیسترها برای آدرس دهی خطی می‌توان استفاده کرد، اما تنها رجیسترهاي B4-B7 و A4-A7 برای آدرس دهی سیرکولار قابل استفاده هستند. مودهای آدرس دهی مختلف در جدول ۱-۳ آورده شده‌اند. عملگر * شبیه عملگر اشاره گر در برنامه نویسی C نشان می‌دهد رجیستر بعد از آن حاوی آدرس حافظه می‌باشد. همچنین اگر از ++ بعد از نام رجیستر یا قبل از نام رجیستر استفاده شود به ترتیب به معنای افزایش مقدار رجیستر بعد از آدرس و قبل از استفاده از آدرس می‌باشد. به نحو مشابه علامت -- برای کم کردن مقدار رجیستر به کار می‌رود. اگر بعد از نام رجیستر داخل کروشه، عددی نوشته شود مثلاً [d] به معنای آفست می‌باشد.

بسته به آنکه دسترسی به حافظه به صورت بایتی، نیم کلمه (دو بايت) و یا کلمه (4 بايت) باشد، آفست با ضرب شدن در 1، 2 و یا 4 به مقدار رجیستر اضافه می‌شود یا از آن کم می‌شود.

¹Linear

جدول ۱-۳: مودهای آدرس دهنده

Addressing Type	No Modification of Address Register	Preincrement or Predecrement Address Register	Postincrement or Postdecrement Address Register
Register Indirect	*R	*++R	*R++
		*--R	*R--
Register Relative	*+R[ucst5]	*++R[ucst5]	*R++[ucst5]
	*-R[ucst5]	*--R[ucst5]	*R-[ucst5]
Register Relative with 15-bit Constant Offset	*+B14/B15[ucst15]	none	None
Base+INdex	*+R[offsetR]	*++R[offsetR]	*R++[offsetR]
	*-R[offsetR]	*--R[offsetR]	*R-[offsetR]
Notes:			
ucst5 = 5-bit unsigned integer constant			
ucst15 = 15-bit unsigned integer constant			
R = base register			
OffsetR = index register			

نقشه حافظه در پردازنده C6713

فضای آدرس دهنده در این پردازنده 32 بیتی می‌باشد($2^{32}=4GB$) که شامل چهار فضای آدرس خارجی CE0، CE1، CE2 و CE3 می‌باشد. وجود بلوکهای مختلف حافظه و باسهای متعدد به پردازنده اجازه می‌دهد در یک سیکل اجرای دستور^۱ دو دسترسی به حافظه داشته باشد (مثلاً دو Load موازی یا دو Write موازی در دو بلوک مختلف حافظه). پردازنده C6713 264 KB حافظه داخلی می‌باشد.

جدول ۱-۴: نقشه حافظه برای پردازنده C6713 و بورد DSK مربوطه را نشان می‌دهد.

جدول ۱-۴: نقشه حافظه برای TMS320C6713

Address	C67x Family Memory Type	DSK 6713
0x00000000	Internal Memory	Internal Memory
0x00030000	Reserved Space or Peripheral Regs	Reserved or Peripheral
0x80000000	EMIF CE0	SDRAM
0x90000000	EMIF CE1	Flash
0x90080000		CPLD
0xA0000000	EMIF CE2	Daughter Card

¹ Instruction Cycle

0xB0000000	EMIF CE3
------------	----------

کتابخانه^۱: BSL

برای استفاده از قابلیتهایی از بورد و پیکربندی آنها، نظیر دستیابی به کدک، خواندن وضعیت DIP سوئیچها، روشن و خاموش کردن LED های روی بورد، توابعی در فایل کتابخانه ای به نام dsk6713bsl.lib وجود دارند که می‌توان از آنها در کدهای C خود استفاده کرد. به عنوان مثال برای پیکربندی کدک از تابع DSK6713_AIC23_openCodec() استفاده نمود.

دسترسی به حافظه:

داده‌ها به صورت بایتی، نیم کلمه و کلمه قابل دسترسی هستند. بیت کم ارزش آدرس شروع داده‌ها به صورت نیم کلمه باید ۰ باشند. همچنین برای داده چهار بایتی (Word) باید دو بیت کم ارزش آدرس آنها ۰ باشد. اصطلاحاً گفته می‌شود کلمه (word) باید با مرز کلمه هم جهت^۲ شود. به عنوان مثال برای استفاده از دستور LDW باید آدرس شروع داده با مرز کلمه هم جهت شده باشد (مضرب ۴ باشد) در غیر این صورت داده اشتباه از حافظه برداشته می‌شود.

در صورتی که عددی چند بایت باشد و بایت قسمت کم ارزش آن در خانه با آدرس کوچکتر قرار گیرد گوییم از مدل Little Endian است و بر عکس در صورتی که بایت کم ارزش در حافظه با آدرس بیشتر قرار گیرد گوییم مدل Big Endian است.

: pragma راهنمای

در کدهای C راهنمای pragma به کامپایلر درباره نکات خاصی پیغام می‌فرستد. مثلاً

```
#pragma DATA_ALIGN (symbol, constant)
```

توانی از ۲ است و symbol نام متغیر می‌باشد. کامپایلر متغیر symbol را در حافظه در آدرسی مضرب constant قرار می‌دهد.

```
#pragma DATA_SECTION (symbol, "my_section")
```

^۱Board support library

^۲Align

در اینجا کامپایلر متغیر `symbol` را در بخش "my_section" از حافظه قرار می‌دهد. از این راهنما برای قرار دادن متغیر در حافظه خارجی می‌توان استفاده کرد.

انواع داده برای پردازنده‌های سری C6000

جدول ۵-۱ انواع مختلف داده را برای پردازنده‌های سری C6000 نشان می‌دهد که در برنامه C می‌توان استفاده نمود.

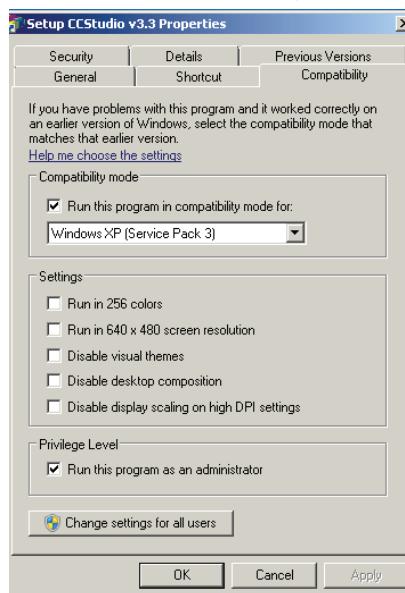
جدول ۵-۱ انواع داده در پردازنده‌های سری C6000

Name	Size (bits)	Type	Minimum	Maximum
short	16	Integer	-32768	32767
Int	32	Integer	-2147483648	2147483647
Long	40	Integer	-549755813888	549755813887
Pointer	32	Address	0000 0000h	FFFF FFFFh
IEEE float	32	Floating point	1.17549435e-38	3.40282347e+38
IEEE double	64	Floating point	2.225073855072014e-308	1.7976931348623157e+308

نصب نرم افزار Code Composer Studio

برای برنامه نویسی پردازنده‌های شرکت TI این شرکت یک نرم افزاری به نام code composer studio ارائه نموده است. در بسته بورد های آموزشی این شرکت نسخه‌ای محدود از نظر انواع پردازنده‌هایی که پشتیبانی می‌کند وجود دارد. در حال حاضر آخرین ویرایش این نرم افزار CCSv5.3 می‌باشد. نرم افزار همراه بسته بورد DSK C6713 نسخه CCSv3.1 است که این بورد آزمایشگاهی را پشتیبانی می‌نماید. در این آزمایشگاه نسخه CCSv3.3 استفاده می‌شود. نصب کردن آن کار سراستی می‌باشد و کافی است فایل نصب آن را تا انتها پیش رفت. بعد از نصب، دو آیکن از این نرم افزار روی desktop ویندوز ظاهر می‌شوند. یکی CCStudio v3.3 و دیگری Setup CCStudio v3.3 می‌باشد. برای برنامه نویسی برای یک پردازنده مورد نظر از TI ابتدا با کلیک کردن بر روی آیکن Setup CCStudio v3.3 در این برنامه پردازنده مورد نظر را با توجه به مشخصات برنامه ریزی شدن، انتخاب می‌کنید. در حقیقت setup بورد در این قسمت مشخص می‌شود. بعد با کلیک کردن بر روی آیکن CCStudio v3.3 وارد برنامه اصلی می‌شوید. در آزمایش اول با محیط این نرم افزار بیشتر آشنا می‌شوید. این نسخه برای سیستم عامل Windows XP است و در صورتی که این نسخه را روی سیستم عامل Windows 7 نصب می‌کنید، باید از قابلیت compatibility mode ویندوز 7 استفاده آزمایشگاه سیستمهای پیشرفته مخابرات سیار - دانشگاه سینگنالهای دیجیتال

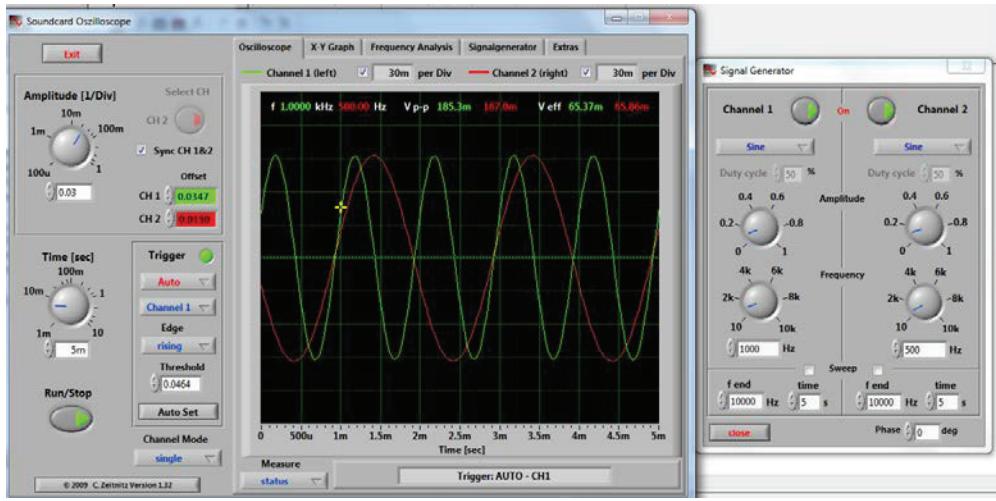
نمایید. برای این منظور بر روی آیکونهای مذکور کلیک راست بزنید و گزینه properties را انتخاب نمایید، سپس از پنجره ای که باز می‌شود برگه compatibility را انتخاب کنید و گزینه Run this program in compatibility mode for Windows XP را برای program in compatibility mode for در قسمت C6416 DSK وجود Setup CCStudio v3.3 بورد C6713 DSK یا بورد C6713 DSK به ندارد. بنابراین باید برای معرفی این بوردهای به نرم افزار CCS اقدامات بعدی انجام شود. در وب سایت این محصولات از شرکت سازنده (spectrum digital) zip فایلی شده حاوی اطلاعات مربوط به بورد می‌باشد. به عنوان مثال برای بورد DSK6713 از صفحه وب با آدرس Target Version 2 در سطر http://c6000.spectrumdigital.com/dsk6713 Zip فایل Content با عنوان DSK6713.zip را دانلود کنید. نام‌گذاری پوشش‌های این فایل متناظر با نام‌گذاری پوشش‌های مربوطه در محل نصب نرم افزار CCS می‌باشد. یعنی باید فایلهای درون هر DSK6713 را در پوشش متناظر در محل نصب CCS (به عنوان مثال C:\CCStudio_v3.3) کپی نمایید. همچنین در سایت شرکت سازنده (spectrum digital) فایلی حاوی کلیه درایورهای محصولات این شرکت برای نرم افزار CCS وجود دارد، که می‌توان آن را در سایت این شرکت یافت و برنامه setup آن را اجرا کرد تا همه کارها به طور خودکار انجام شود. در حال حاضر این فایل setupCCSPlatinum_v30330.zip نام دارد.



شکل ۱-۴ تنظیم compatibility mode برای استفاده از نرم افزار CCS v3.3 در ویندوز 7

نرم افزار Scope

در این آزمایشگاه می‌توانید از کارت صوتی کامپیوتر و نرم افزار Scope به عنوان سیگنال ژنراتور و اسیلوسکوپ استفاده کنید. شکل ۱-۵ تصویری از این نرم افزار نشان می‌دهد. اسیلوسکوپ و سیگنال ژنراتور در این نرم افزار دو کاناله می‌باشد. ورودی آنالوگ آن (اسیلوسکوپ) پورت Line In از کارت صوتی و خروجی آنالوگ آن (سیگنال ژنراتور) پورت Line Out از کارت صوتی است.



شکل ۱-۵ تصویر نرم افزار Scope

نحوه ارائه نتایج آزمایش:

با توجه به اینکه در این آزمایشگاه نتایج فعالیتها به صورت پروژه‌های در نرم افزار Code Composer Studio (CCS) ارائه می‌گردد، برای هماهنگی و نظم در ارائه نتایج انجام آزمایشها لازم است دانشجویان از الگوی یکسانی پیروی نمایند. به این ترتیب انتظار می‌رود کلیه فایلهای هر گروه در درایو اختصاصی (درایو Z در شبکه) همان گروه ذخیره گردند. برای هر آزمایش یک پوشه با عنوان Labi بسازید، که آ شماره آزمایش است، و سپس درون آن پوشه‌هایی با عنوان sectionj درست کنید در اینجا نیز عدد j شماره آزمایش را نشان می‌دهد و سپس در هر پوشه section CCS پروژه را ایجاد نمایید و در صورت لزوم فایلهای دیگر مثلًا فایلهای عکس از نتایج اجرا برنامه را ذخیره نمایید. به علاوه هر پروژه‌ای که برای CCS می‌سازید به طور مستقل در نظر بگیرید، یعنی فایلهایی که به آن اضافه می‌گردد مثل فایلهای C و اسembly و Linker command باید در پوشه همان پروژه قرار داشته باشند.

۲ آزمایش اول: آشنایی با نرم افزار Code Composer Studio

۱-۲ هدف

در این آزمایش قابلیتهای پایه نرم افزار CCS برای پیاده سازی و خطایابی برنامه‌ها برای پیاده سازی در پردازنده‌های شرکت TI معرفی می‌گردد.

۲-۲ مقدمه

برای برنامه نویسی پردازنده‌های DSP شرکت TI، الگوریتم مورد نظر به زبان C/C++ و یا اسembلی نوشته می‌شود. (CCS) نرم افزاری^۱ IDE برای برنامه نویسی و خطایابی^۲ برنامه‌ها برای پیاده سازی الگوریتمهای پردازش سیگنال بر روی پردازنده‌های DSP شرکت TI می‌باشد. این برنامه فراخوانی کامپایلر C و اسembلر و لینکر را انجام می‌دهد. همچنین از این نرم افزار برای بارگذاری^۳ برنامه بر روی بورد DSP و اجرای آن استفاده می‌شود. ویرایش نرم افزاری که این دستور کار بر اساس آن نوشته شده است CCSv3.3 می‌باشد. CCS یک محیط مدیریت فایل و پروژه در اختیار کاربر قرار می‌دهد. به علاوه با استفاده از ویرایشگر^۴ خود امکان نوشتند و ویرایش فایلهای C و اسembلی را به کاربر می‌دهد. همچنین ابزارهای خطایابی نظیر Break point، قابلیتهای نمایش گرافیکی داده‌ها و ابزارهای benchmark در این نرم افزار در نظر گرفته شده است. برای آشنایی بیشتر با نرم افزار CCS می‌توان به مراجع [۲] و [۳] TI CCS User's Guide و TI CCS Tutorial مراجعه کرد. در این آزمایش ضمن اجرای برنامه‌های به زبان C و اسembلی، تعدادی عدد را در حافظه ذخیره می‌کنیم. در ضمن این عملیات با نحوه ساخت پروژه در نرم افزار CCS و اجرای آن آشنا می‌شویم. همچنین با قابلیتهای خطایابی نظیر مشاهده مقادیر متغیرها و خانه‌های حافظه، نمایش گرافیکی داده‌ها و قابلیتهای benchmarking و بهینه سازی^۵ نرم افزار CCS آشنا می‌شویم.

¹ Integrated development environment

² Debugging

³ Load

⁴ Editor

⁵ Optimization

۳-۲ آزمایش

۱-۳-۲ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS

توجه: در این آزمایش برای قسمت اول و دوم یک پروژه CCS بسازید که در حین انجام آزمایش مرحله به مرحله تکمیل می‌شود. با توجه به توضیحات فصل اول این پروژه در زیرپوشه section1 از پوشه Lab1 ساخته شود. همچنین برای قسمت سوم یک پروژه جداگانه در پوشه section3 بسازید. در ادامه برای هر قسمت اختیاری یک پروژه جداگانه در زیر پوشه جداگانه بسازید.

۲-۳-۲ قسمت اول: ایجاد یک پروژه در نرم افزار CCS

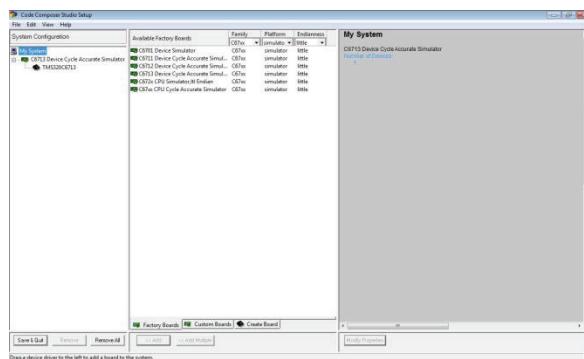
برای ایجاد یک فایل اجرایی بر روی پردازنده DSP به فایلهای مختلفی نظیر کدهای C/C++ (.c, .cpp) و کدهای اسembly (.asm, .cmd)، فایلهای linker command file (.as)، فایلهای (.h)، فایلهای کتابخانه (.lib) نیاز است. این فایلهای در قالب یک فایل پروژه (با پسوند .prj) جمع می‌گردند. در ادامه مراحل ایجاد یک پروژه در نرم افزار CCS ذکر می‌گردد.

۱- در این نرم افزار ابتدا باید بورد نظر را انتخاب کنیم. به این منظور بر روی آیکون Setup

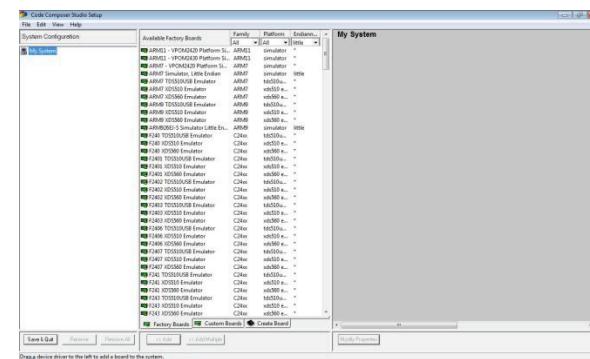


کلیک می‌کنیم تا برنامه آن باز شود (شکل ۱-۲).

از قسمت Family گزینه C67xx و از قسمت Simulator گزینه Platform را انتخاب می‌کنیم. بعد از اینکه به این نحو انتخاب خود را فیلتر نمودیم بر روی بورد Little گزینه C6713 Device Cycle Accurate Simulator را انتخاب می‌کنیم. دبل کلیک می‌کنیم تا به پنجره سمت چپ وارد شود (شکل ۲-۲)



شکل ۲-۲ انتخاب بورد مورد نظر در CCS Setup



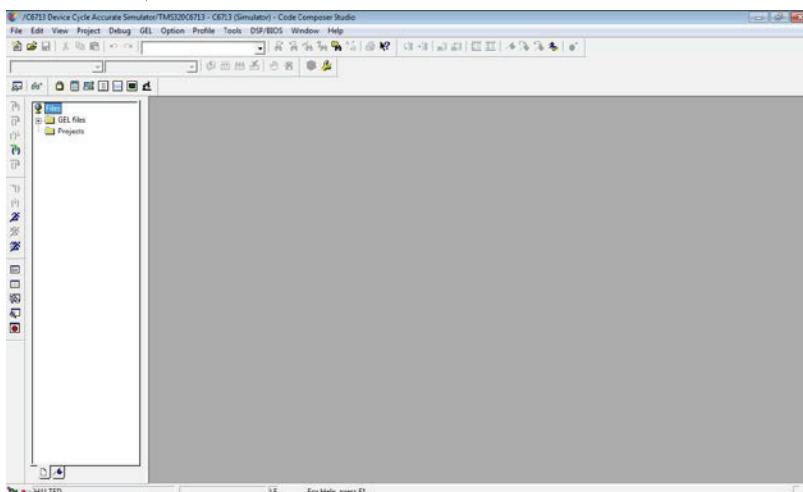
شکل ۱-۲ شمای نرم افزار Code Composer Studio Setup

و سپس گزینه Save & Quit را انتخاب می کنیم و در جواب

گزینه Yes را انتخاب می کنیم تا برنامه CCS اجرا گردد (شکل ۳-۲).

توجه: در اینجا یک شبیه ساز برای بورد انتخاب شده است. در صورتی که بخواهیم برنامه را بر روی بورد DSK C6713 پیاده نماییم، از ستون platform گزینه dsk را انتخاب کنید و بورد مورد نظر را انتخاب کنید. در فصل اول درباره معرفی C6713 DSK به CCS توضیح داده شده است.

توجه: ممکن است در نسخه هایی از نرم افزار CCS که همراه بورد C6713 DSK ارائه می شوند، بخش setup بورد وجود نداشته باشد (مرحله ۱) و مستقیماً برنامه CCS را باز کنیم. (از مرحله ۲ شروع کنیم.)

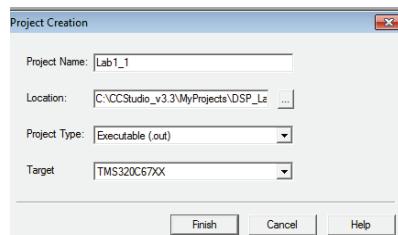


شکل ۳-۲ نمای نرم افزار Code Composer Studio

۲- در محیط نرم افزار CCS از منوی Project گزینه New را انتخاب کنید به این ترتیب پنجره ای

مشابه شکل ۲-۴ باز می شود که مشخصات پروژه، نظیر نام، محل ذخیره سازی آن، نوع خروجی،

و بورد مورد نظر را دریافت می کند.



شکل ۴-۲ پنجره ایجاد پروژه جدید

۳- ایجاد فایل برنامه^۱: نرم افزار CCS یک ویرایشگر برای ایجاد فایلهای لازم نظیر C و اسembly و linker command را ایجاد کرده است. برای ایجاد یک فایل جدید از منوی File گزینه New و Source File را انتخاب کنید. در اینجا در پنجره باز شده کد اسembly شکل ۵-۲ را وارد کنید و آن را با پسوند .asm ذخیره کنید (initmem.asm).

توجه داشته باشید در فایلهای اسembly دستورها و راهنمایها^۲ باید حتماً از ستون دوم به بعد نوشته شوند (فاصله ای بین لبه صفحه و شروع آنها باشد) زیرا کلماتی که از ستون اول شروع می‌شوند به عنوان Label (آدرس) شناخته می‌شوند.

```
.sect "mydata"
.short 0
.short 7
.short 10
.short 7
.short 0
.short -7
.short -10
.short -7
.short 0
.short 7
```

شکل ۵-۲ کد اسembly برای ذخیره تعدادی داده در حافظه

در این فایل sect. مشخص می‌کند که کدهای زیر آن در بخش mydata از حافظه قرار گیرد و راهنمای short. نشان می‌دهد عدد بعد از آن ۱۶ بیتی است. در فایل command مشخص می‌کنیم بخش mydata در چه آدرسی از حافظه قرار داشته باشد (شکل ۷-۲).

در ادامه یک فایل c. نیز ایجاد می‌کنیم (فایل c. و فایل C. و فایل C++ می‌باشد). به روش مشابه با ایجاد فایل اسembly، فایل main.c (شکل ۶-۲) را ایجاد و ذخیره کنید.

¹ Source file² Directive

```
#include <stdio.h>
main()
{
    printf("Begin\n");
    printf("End\n");
}
```

شکل ۶-۲ فایل ساده ۵

-۴ ایجاد فایل linker command : برای پیاده سازی برنامه بر روی پردازنده DSP این فایل اطلاعاتی به لینکر می دهد. در این فایل مشخص می کنیم بخش های مختلف کد برنامه و متغیرها در چه آدرسی از حافظه قرار گیرند. به این منظور یک فایل ایجاد کنید و کدهای شکل ۷-۲ را در آن وارد کنید و آن را با پسوند .cmd ذخیره نمایید. البته می توانید این فایل را از مسؤول آزمایشگاه نیز بگیرید. لازم به ذکر است عبارات بین * / و * / توضیح می باشند و وجود آنها ضروری نیست. برای آشنایی با محتوای فایل لینکر می توانید به بخش ضمایم این فصل و مرجع [۱۵] رجوع کنید.

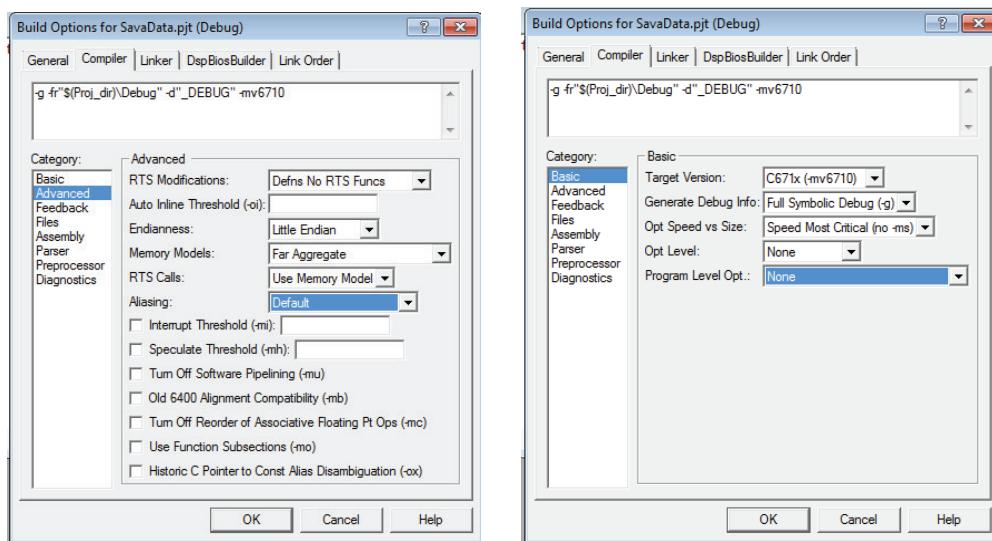
```
MEMORY
{
    IST : origin = 0x0,           len = 0x400
    IRAM : origin = 0x400,         len = 0x40000 /* 256 Kbytes */
    SDRAM: origin = 0x80000000, len = 0x1000000 /* 16 Mbytes SDRAM */
    FLASH: origin = 0x90000000, len = 0x40000 /* 256 Kbytes */
}
SECTIONS
{
    .vec:      { }> IST /* Interrupt vectors included */
    .text:     { }> IRAM /* Executable code */
    .const:    { }> IRAM /* Initialized constants */
    .bss:      { }> IRAM /* Global and static variables */
    .data:     { }> IRAM /* Data from .asm programs */
    .cinit:    { }> IRAM /* Tables for initializing */
                    /* variables and constants */
    .stack:    { }> IRAM /* Stack for local variables */
    .far:      { }> IRAM /* Global and static variables */
                    /* declared far */
    .sysmem:   { }> IRAM /* Used by malloc, etc. (heap) */
    .cio:      { }> IRAM /* Used for C I/O functions */
    .csldata:  { }> IRAM
    .switch:   { }> IRAM
    mydata    { }> SDRAM
}
```

شکل ۷-۲ فایل linker command برای آزمایش اول

۵- اضافه کردن فایلها به پروژه: برای اضافه کردن فایل‌های مورد نیاز به پروژه، از منوی Project گزینه ... Add Files To Projects را انتخاب کنید و فایل‌های initmem.asm و main.c و dsk6713.cmd را به پروژه اضافه کنید.

۶- اضافه کردن کتابخانه‌های لازم به پروژه: معمولاً run-time support library لازم است کتابخانه مناسب نیز به پروژه اضافه شود. نام این کتابخانه بسته به اینکه با کدام خانواده از پردازنده‌های شرکت TI کار می‌کنیم و همچنین بسته به برخی تنظیمات مثل مدل حافظه، متفاوت است. مکان این فایل کتابخانه‌ای در پوشه cgtools از پوشه خانواده مورد نظر می‌باشد. به عنوان مثال در اینجا که پردازنده مورد نظر 6713، عضو خانواده سری 6000 می‌باشد، نشانی این کتابخانه Far Aggregate در حالی که از مدل حافظه C:\CCStudio_v3.3\C6000\cgtools\lib می‌باشد. در قسمت Build Option استفاده کرده باشیم، کتابخانه مورد نظر little endianness و rts6700.lib نام دارد.

۷- انجام تنظیمات کامپایلر: برای انجام تنظیمات مربوط به کامپایلر از منوی Project گزینه Options ... را انتخاب کنید و سپس روی قسمت compiler کلیک کنید. با توجه به اینکه می‌خواهیم برنامه‌ها را بر روی پردازنده TMS320C6713 اجرا کنیم در دسته Basic و در بخش Target Version: عبارت 6710 (-mV 671x) را انتخاب کنید. بقیه پارامترها را به مقدار پیش فرض باقی می‌گذاریم. شکل ۸-۲ تنظیمات مربوط دسته‌های Basic و Advanced را نشان می‌دهد.



شکل ۸-۲ تنظیمات کامپایلر در قسمت Build Options

از تنظیمات کامپایلر می‌توان به تعیین مدل حافظه، تعیین سطح بهینه سازی، تعریف^۱ عبارات در قسمت **preprocessor**، تعیین آدرس فایلهای header مورد استفاده در پروژه، اشاره کرد.

-۸- انجام تنظیمات Linker: برای انجام تنظیمات مربوط به Linker از منوی Project گزینه **Linker Options ...** و سپس قسمت linker را انتخاب کنید. از پارامترهای تنظیم linker می‌توان به آدرس کتابخانه‌ها، آدرس شروع برنامه (Code Entry Point (-e))، طول استک (stack size) اشاره کرد. در اینجا مقادیر پارامترها را در مقادیر پیش فرض باقی بگذارید.

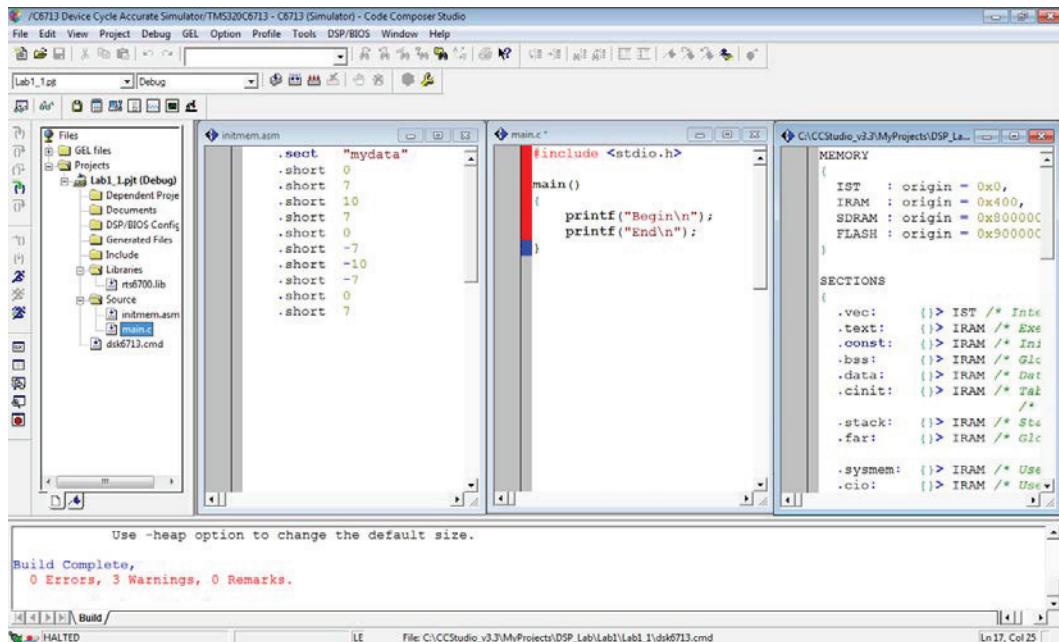
-۹- Build: منظور از build پروژه ترکیب دو عملیات compile و سپس link می‌باشد. بعد از اضافه کردن فایلهای لازم به پروژه و انجام تنظیمات لازم، باید پروژه Build شود. به این منظور از منوی Project گزینه Build را انتخاب کنید. اگر خطای وجود نداشته باشد، فایل اجرایی lab1_1.out ساخته می‌شود که در زیرپوشه Debug از پوشه پروژه می‌باشد. همچنین امکان استفاده از گزینه All Rebuild از منوی Project نیز وجود دارد، به این معنی که تنها فایلهایی که نسبت به آخرین Build تغییر کرده‌اند، Build می‌شوند.

-۱۰- در صورتی که از بورد C6713 DSK استفاده می‌کنید قبل از بارگذاری برنامه در حافظه پردازنده لازم است از منوی Connect گزینه Debug را انتخاب نمایید. در انتهای کار وقتی لازم داشته

^۱ define

باشد ارتباط بورد با نرم افزار CCS را قطع کنید از منوی Debug گزینه Disconnect را انتخاب کنید.

۱۱- بارگذاری^۱ برنامه بر روی DSP: بعد از آنکه پروژه بدون خطا Build شد. قبل از اجرای برنامه باید فایل اجرایی حاصل از مرحله Build را در حافظه پردازنده قرار بدهید. به این منظور از منوی File گزینه Load Program را انتخاب کنید و فایل Lab1_1.out را از درون زیرپوشش Project از پوشه پروژه مورد نظر برگزینید. شکل ۹-۲ تصویر نرم افزار CCS و پنل View را نشان می‌دهد.



شکل ۹-۲ نمای پنجره های مختلف در ارتباط با پروژه lab1

۱۲- اجرای برنامه: برای اجرای برنامه از منوی Debug گزینه Run را انتخاب کنید. به این ترتیب با اجرای این کدها اعداد نوشته شده در فایل اسembly در حافظه خارجی از آدرس END ۰x80000000 قرار می‌گیرند و حاصل اجرای فایل main.c که نوشتن عبارات BEGIN و cout می‌باشد در پنجره stdout نمایش داده می‌شود.

^۱ Load

۳-۳-۲ قسمت دوم: ابزارهای خطایابی

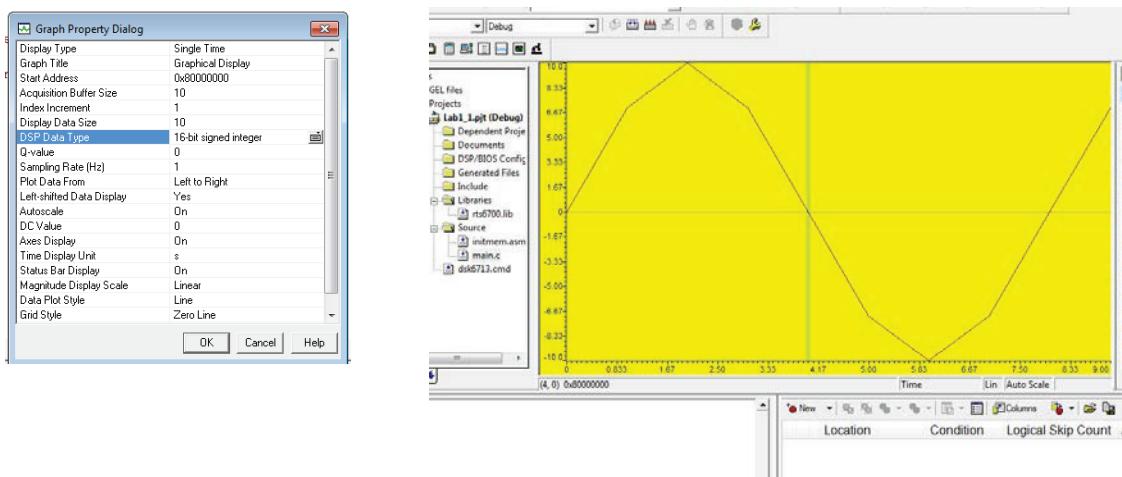
۱- مشاهده خانه های حافظه: برای مشاهده خانه های حافظه از منوی View گزینه Memory را انتخاب کنید. به این ترتیب در پنجره Memory در قسمت Address آدرس شروع 16-bit Signed Format mydata (0x80000000) را وارد کنید و همچنین در قسمت mydata عبارت Integer را انتخاب کنید (شکل ۱۰-۲).

نکته: می توانید برای مشاهده محتوای رجیسترهای پردازنده از منوی View گزینه های Register و سپس Core Register را انتخاب کنید.

Address	Value	Sign
0x80000000	0	7
0x80000004	10	7
0x80000008	0	-7
0x8000000C	-10	-7
0x80000010	0	7
0x80000014	0	0
0x80000018	0	0
0x8000001C	0	0
0x80000020	0	0
0x80000024	0	0
0x80000028	0	0
0x8000002C	0	0
0x80000030	0	0
0x80000034	0	0
0x80000038	0	0
0x8000003C	0	0
0x80000040	0	0

شکل ۱۰-۲ پنجره Memory و پنجره تنظیمات آن

۲- نمایش گرافیکی داده‌ها: قابلیت مفیدی که در نرم افزار CCS قرار داده شده است نمایش گرافیکی و پردازش مختصر (FFT) بر روی داده‌هاست. به این منظور از منوی View Graph Property Dialog گزینه Graph و سپس Time/Frequency... را انتخاب کنید تا پنجره Graph DSP و Display data size و Acquisition Buffer Size ، Start Address باز شود. قسمتهای Data Type را به ترتیب با 16-bit signed-integer (0x80000000، 10، 10) و OK گزینه را بزنید. پنجره Graph Property و پنجره Graph متناظر با آن در شکل ۱۱-۲ دیده می‌شود. برای ویرایش مشخصات پنجره Graph روی آن کلیک راست کنید و گزینه Properties را انتخاب نمایید.



شکل ۱۱-۲ پنجره های Graph و Graph property Dialog متناظر با آن

۳- تخصیص اشاره گر^۱ به آدرس مشخصی از حافظه: در این قسمت می خواهیم به مقادیر حافظه از درون کد C دسترسی داشته باشیم. این کار با استفاده از مفهوم اشاره گر امکان پذیر است. به این منظور باید حتماً آدرس حافظه به نوع اشاره گر type cast شود. برای این منظور فایل c main.c را مطابق شکل ۱۲-۲ تغییر دهید. سپس پروژه را Build و بعد فایل Lab1_1.out را load و سپس اجرا کنید. با اجرای این برنامه شما باید ۱۰ مقدار ۱۶ بیتی (short) حافظه از آدرس ۰x80000000 را در پنجره stdout مشاهده نمایید. همچنین در پنجره های مختلف CCS به جای آدرس حافظه می توانید از نام اشاره گر به آن یا اسم متغیر استفاده کنید.

```
#include <stdio.h>

main()
{
    int i;
    short *point;
    point=(short * ) 0x80000000;

    printf("Begin\n");
    for(i=0;i<10;i++)
        printf("[%d] = %d\n",i, point[i]);
    printf("End\n");
}
```

شکل ۱۲-۲ کد C، تخصیص آدرس به اشاره گر

۴- مشاهده مقادیر متغیرها: برای مشاهده مقدار یک متغیر در حین اجرای برنامه، می توان از Break و Watch Window استفاده کرد. برای ورود متغیر مورد نظر به Watch Window point موردنظر را انتخاب می کنیم و روی آن کلیک راست می کنیم و Add to Watch Window را

¹ Pointer

انتخاب می‌کنیم. متغیر point را وارد watch window نمایید. خواهید دید بعد از اجرای برنامه مقدار متغیر point برابر identifier not found: point است، زیرا متغیرهایی که درون توابع (از جمله تابع main()) تعریف می‌شوند، محلی هستند و در استک^۱ ذخیره می‌شوند. یعنی بعد از خروج از تابع مقدار آنها از بین می‌رود. برای مشاهده مقادیر متغیرهای محلی در محل مناسب در کد برنامه break point (کلید F9) قرار دهید و سپس برنامه را اجرا کنید. برنامه تا محل قرار گرفتن break point پیش می‌رود و در محل Break point توقف می‌کند. در این حالت مقادیر متغیرها قابل نمایش هستند.

با قرار دادن break point در مکان مناسب در برنامه main.c آن را اجرا کنید و مقدار متغیر point را مشاهده کنید.

خط int i; را از درون تابع main() خارج کنید و قبل از آن قرار دهید، به این ترتیب متغیر i به صورت سراسری^۲ تعریف می‌شود. برنامه را Build و سپس Load کنید. مقدار متغیر i را به مقدار متغیر n برابر 10 باشد.

هنگام توقف برنامه در یک Break point با زدن کلید F10 ، برنامه خط به خط اجرا می‌شود و با زدن کلید F11 دستور به دستور.

۵- تعیین تعداد کلاک لازم برای انجام یک تابع: در این قسمت یک تابع به کد فایل main.c اضافه می‌کنیم که تعداد N عدد را با هم جمع می‌زند. هدف از این قسمت تعیین میزان زمان اجرای تابع است. کد main.c را مطابق شکل ۱۳-۲ تغییر دهید.

```
#include <stdio.h>

int ret_sum(const short* array, int N)
{
    int count, sum;
    sum=0;
    for (count=0; count<N; count++)
        sum+=array[count];
    return (sum);
}
```

¹ Stack

² Global

```

main()
{
    int i, ret;
    short *point;
    point=(short *) 0x80000000;

    printf("Begin\n");
    for(i=0;i<10;i++)
        printf("[%d] = %d\n", i, point[i]);
    ret = ret_sum(point,10);
    printf("C program Sum = %d\n", ret);

    printf("End\n");
}

```

شکل ۱۳-۲ کد C تغییر یافته برای تعیین تعداد کلاک اجرای یکتابع با استفاده از قابلیت‌های نرم افزار CCS

جلوی خط تابع `ret_sum` در تابع `main()` یک `break point` قرار دهید. از منوی `Profile` گزینه `Clock` و سپس `Enable` را انتخاب کنید. بعد برای مشاهده آیکون کلاک از منوی `Profile` گزینه `Clock` و سپس `View` را انتخاب کنید. به این ترتیب آیکون یک ساعت زرد رنگ در نوار پایین نرم افزار CCS ظاهر می‌شود . برنامه را `load` و سپس اجرا نمایید. وقتی برنامه در اولین `Break point` توقف نمود روی آیکون ساعت کلیک کنید تا `reset` شود. سپس `F10` را بزنید. بعد از توقف برنامه، تعداد کلاک لازم کنار آیکون کلاک نوشته می‌شود.

نکته ۱: بعد از `Build` پروژه با انتخاب گزینه `Mixed Source/Asm` از منوی `View` کد اسembly معادل کد C قابل مشاهده است.

نکته ۲: برای ذخیره محیط کاری شامل `break points`، `graph`، `watch window` و غیره، می‌توانید از منوی `File` گزینه `Save Workspace...` و سپس `Workspace...` را انتخاب نمایید.

نکته ۳: بعد از اجرای برنامه روی پردازنده در صورتی که بخواهیم دوباره برنامه از اول اجرا شود می‌توان از منوی `Debug` گزینه `Restart` را انتخاب نمود یا اینکه دوباره برنامه را در پردازنده بار نمود.

۶- بهینه سازی کد: در این قسمت هدف مقایسه بین زمان اجرای تابع برای سطوح مختلف بهینه سازی است. کد شکل ۱۴-۲ را که به زبان اسembly نوشته شده است در یک فایل با نام `sum.asm` وارد کنید و به پروژه اضافه کنید. در زبان اسembly برای پردازنده‌های TI عبارات بعد از سمتیکلن ؛ توضیح می‌باشند. نام تابع و نام متغیرهای مشترک بین C و اسembly در کدهای اسembly با

شروع می‌شوند. آرگومان تابع اسمبلی `_sum` طبق قرارداد در رجیستر B4 برای آن ارسال می‌گردد و نتیجه تابع نیز در رجیستر A4 قرار داده می‌شود. در ادامه کد فایل main.c را نیز مطابق شکل ۱۵-۲ تغییر دهید. با توجه به مفاهیم فایل command برای قرار دادن کد در حافظه خارجی `SDRAM > IRAM`. را به `.text` تغییر دهید. به این ترتیب قادر خواهید بود اثر دسترسی به کد و داده در حافظه خارجی را با حضور در حافظه داخلی مقایسه کنید. در کد شکل ۱۳-۲ آدرس متغیر point برابر ۰x80000000 می‌باشد. در صورتی که مکان فایل اسمبلی را در حافظه عوض کنید این آدرس تغییر می‌کند. برای دسترسی به آدرس جدید راه حل‌های متفاوتی می‌توانید ارائه دهید، مثلاً می‌توانید از فایل memory map کمک بگیرید یا اینکه این فایل را در command file در آدرس مشخصی از حافظه داخلی قرار دهید. همچنین می‌توانید تغییری اندک در کدهای شکل ۱۳-۲ و شکل ۱۶-۲ به صورت شکل ۱۳-۲ ایجاد کنید. لازم به ذکر است که فایل memory map یک فایل در پوشه Debug از پروژه مورد نظر است که پسوند آن map باشد و محل قرار گیری قسمتهای مختلف کد و متغیرها و حجم فضای اشغال شده از حافظه را نشان می‌دهد.

```
.global _sum

._sum:
    ZERO .L1 A9          ; Sum register
    MV   .L1 B4,A2        ; initialize counter with passed
                           argument

loop: LDH   .D1 *A4++, A7 ; load value pointed by A4 into register
                           A7
    NOP   4
    ADD   .L1 A7,A9,A9      ; A9 += A7
    [A2] SUB   .L1 A2,1,A2      ; decrement counter
    [A2] B    .S1 loop       ; branch back to loop
    NOP   5

    MV   .L1 A9,A4        ; move result into return register
                           A4
    B    .S2 B3           ; branch back to address stored in
                           B3
    NOP   5
```

شکل ۱۴-۲ کد تابع `_sum` به زبان اسمبلی

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

```

extern sum();
int ret_sum(const short* array, int N)
{
    int count, sum;
    sum=0;
    for (count=0; count<N; count++)
        sum+=array[count];
    return(sum);
}

main()
{
    int i, ret;
    short *point;
    point=(short * ) 0x80000000;
    printf("Begin\n");
    for(i=0;i<10;i++)
        printf("[%d] = %d\n",i, point[i]);
    ret = ret_sum(point,10);
    printf("C program Sum = %d\n",ret);

    ret = sum(point,10);
    printf("Assembly program Sum = %d\n",ret);

    printf("End\n");
}

```

شکل ۱۵-۲ کد C با فراخوانی تابع اسembly

initmem.asm	main.c
<pre> .sect "mydata" .global _point _point .short 0 .short 7 ... </pre>	<pre> ... extern short point[]; main() { int i, ret; //short *point; //point=(short *) 0x80000000; ... </pre>

شکل ۱۶-۲ تغییر در فایلهای main.c و initmem.asm برای دسترسی به آدرس شروع محل قرار گرفتن داده‌ها در حافظه

سطح بهینه سازی را در قسمت Compile option از Build option تغییر دهید و بعد از Build و اجرای برنامه تعداد کلای لازم را در جدول ۱-۲ وارد کنید. هنگام قرار دادن داده در حافظه داخلی به کمک فایل command، در برنامه C آدرس شروع داده‌ها را باید تغییر دهید. برای اجرای مجدد یک برنامه در

پردازنده از منوی Debug گزینه Restart را انتخاب نمایید. همچنین همواره در پنجره stdout از صحت نتیجه توابع محاسبه کننده جمع اطمینان حاصل نمایید.

جدول ۱-۲ مقایسه تعداد کلاکها برای build های مختلف

Type of Build		Number of Cycles	
		Data in external memory	Data in internal memory
Code in external memory	C routine		
	Assembly routine		
Code in internal memory	C routine (No Opt.)		*****
	C routine (-o0)		*****
	C routine (-o1)		*****
	C routine (-o2)		*****
	C routine (-o3)		*****

لازم به ذکر است که در صورتی که بورد DSP در اختیار نباشد، نرم افزار CCS می‌تواند به عنوان شبیه‌ساز^۱ بورد، مورد استفاده قرار گیرد. به این منظور در نرم افزار CCS Setup simulator باید خانواده پردازنده مورد نظر را انتخاب کنید. البته احتمالاً برخی قابلیتها نظیر HPI و Timer در شبیه‌سازی پشتیبانی نشوند.

نکته: در صورتی که از بورد DSK استفاده می‌کنید، برای تعین تعداد کلاک اجرای کد، ظاهراً نرم افزار CCS به رویی که گفته شد صحیح عمل نمی‌کند! برای آنکه CCS در این حالت نیز عملکرد مطلوب داشته باشد باید بعد از آنکه کارهای قبلی را انجام دادیم از منوی profile گزینه clock و سپس setup را انتخاب کنید و بدون تغییر پارامترها دکمه OK را بزنید!!

¹ Simulator

۴-۳-۲. قسمت سوم: GEL File

با استفاده از Gel^۱ file می‌توان مقادیر متغیرها را در حالت برنامه در پردازنده در حال اجراست تغییر داد. برای این منظور در یک فایل جدید در نرم‌افزار CCS کدهای لازم را می‌نویسیم و فایل را با پسوند .gel ذخیره می‌کنیم و سپس از منوی Load GEL File گزینه ... را انتخاب می‌کنیم و این فایل را از این طریق به پروژه اضافه می‌کنیم.

فرمت یک تابع slider در gel file به صورت زیر است:

```
slider param_definition(minVal, maxVal, increment, PageIncrement,
paramName)
{
    statements
}
```

نام پنجره slider است. minVal و maxVal مقادیر مینیمم و ماکزیمم قابل تخصیص به متغیر می‌باشند. increment مقدار اضافه شده یا کم شده به متغیر paramName با استفاده از کلیدهای جهتی صفحه کلید می‌باشد و PageIncrement مقدار اضافه شده به این متغیر با کلیک بر روی پنجره slider می‌باشد.

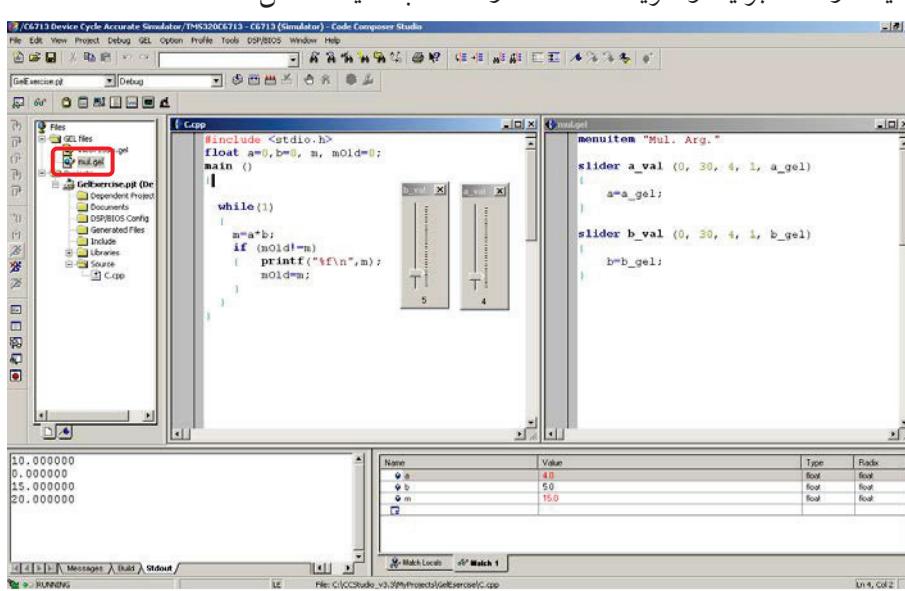
به عنوان مثال برای تغییر متغیر سراسری gain در کد برنامه C در حین اجرای برنامه بر روی پردازنده یک فایل gel با محتوای کد زیر با پروژه خود اضافه کنید.

```
menuitem "Gain"
slider Gain (0, 30, 4, 1, gain_parameter)
{
    gain=gain_parameter;
}
```

در مثال فوق menuitem نامی برای slider در منوی Gain در نرم‌افزار CCS قرار می‌دهد. برای دسترسی به اسلايدر ایجاد شده از منوی GEL، slider مربوطه را انتخاب می‌کنیم. به این ترتیب با

¹ General Extension Language

تغییر slider پنجره Gain می توانیم مقدار متغیر gain را تغییر دهیم. برای حذف یک Gel فایل از پروژه، از پنجره File View در محیط CCS روی آیکون Gel files کلیک کنید و روی Gel فایلی که می خواهید حذف کنید کلیک راست بزنید و گزینه remove را انتخاب کنید (شکل ۱۷-۲).



شکل ۱۷-۲ امکان gel فایل

پروژه ای جدید در CCS ایجاد کنید. در این قسمت برنامه ای به زبان C بنویسید و به پروژه خود اضافه کنید که حاصل ضرب دو عدد را حساب کند و با دستور printf() نمایش دهد. با استفاده از gel فایل دو اسلایدر بسازید که آرگومانهای عملیات ضرب را تعیین کنند. برای آنکه اجرای برنامه به صورت پیوسته ادامه داشته باشد، در تابع main() از یک while(1) استفاده کنید و عملیات حاصل ضرب را جز دستورات آن بگذارید. مقادیر متغیرهای سراسری آرگومان حاصل ضرب و نتیجه حاصل ضرب را در Watch Window مشاهده نمایید.

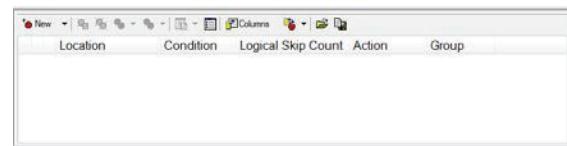
۳-۵-۵ قسمت چهارم: ورود و خروج داده از فایل

نرم افزار CCS قابلیت آن را دارد که حین اجرای برنامه داده از فایل بخواند و یا داده در فایل بنویسد. این قابلیت مفیدی برای تست کد نوشته شده با داده های معین می باشد. از این قابلیت در ویرایشهای قدیمی تر break point یاد می شده است، در حالی که در ویرایش ۳.3 این قابلیت در قالب امکانات CCS به point ارائه می گردد. برای استفاده از این قابلیت مراحل زیر را انجام دهید:

- ۱- برنامه را load کنید. سپس از منوی Breakpoints گزینه Debug را انتخاب کنید تا پنجره Breackpoint Manager باز شود(شکل ۱۸-۲).
- ۲- در محل مورد نظر در کد برنامه Breakpoint قرار دهید. برای این کار در ستون سمت حاشیه چپ خط مورد نظر کلیک کنید یا کرسر را در خط مورد نظر قرار دهید و کلید F9 را بزنید. به این ترتیب دایره قرمز رنگی در کنار خط مورد نظر قرار می‌گیرد.
- ۳- در پنجره Breakpoint Manager در سطر مربوط به Breakpoint مورد نظر روی قسمت مربوط به Action کلیک کنید. سپس در منوی باز شده (شکل ۱۹-۲) ، گزینه Write Data to File یا Read Data From File را انتخاب کنید.
- ۴- با انتخاب Action مربوط به Breakpoint، پنجره پارامترهای آن باز می‌شود (شکل ۲۰-۲ و شکل ۲۱-۲).



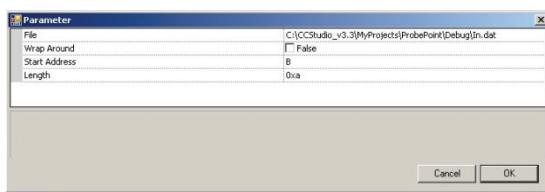
شکل ۱۹-۲ انتخاب عملکرد break point



شکل ۱۸-۲ پنجره Breakpoint Manager



شکل ۲۱-۲ پنجره پارامترها برای نوشتن قسمتی از حافظه در یک break point با استفاده از قابلیت file



شکل ۲۰-۲ پنجره پارامترها برای خواندن از فایل با استفاده از قابلیت break point

فایل خوانده شده یا نوشته شده فایل متنی می‌باشد و دارای فرمات خاصی است. خط اول آن از

پنج قسمت زیر تشکیل شده است:

1651	Format	Starting Address	PageNum	Length
------	--------	------------------	---------	--------

عدد 1651 یک عبارت ثابت است. Format بر حسب اینکه داده‌ها از کدام یک از انواع Starting Address باشد به ترتیب 1، 2، 3 و یا 4 هستند. float و long ، integer، hexadecimal

برای فایل نوشته شده توسط CCS آدرس شروع بلوک و شماره صفحه بلوک داده را نشان می‌دهد. قسمت Length تعداد بایت داده‌ها را نشان می‌دهد. خط هدر در مبنای ۱۶ نشان داده می‌شوند. برای خواندن داده‌های float از فایل، داده‌ها باید ممیزدار باشند، همچنین آدرس شروع و تعداد داده‌ها در فایل داده تعیین‌کننده نیستند و آدرسی (یا اسم متغیر) که در پنجره پارامترها (شکل ۲۱-۲) وارد می‌گردد ملاک عمل قرار می‌گیرد، البته به هر حال تعداد داده‌ها در فایل داده نباید ۰ باشد. یک نمونه کلی از فایل داده برای خوانده شدن توسط قابلیت break point را نشان می‌دهد.

```
1651 4      0      0      0x4
0.6715
-1.2075
0.7172
1.6302
0.4889
1.0347
0.7269
-0.3034
0.2939
0.1
```

شکل ۲۲-۲ یک نمونه فایل متنی داده برای خوانده شدن توسط قابلیت break point

راه حل دیگر برای وارد و خارج کردن داده از فایل، استفاده از توابع fopen و fprintf و fscanf در برنامه C می‌باشد. در صورت به کارگیری این توابع، فایل را بایست در پوشه debug از پوشه پروژه این کد قرار دارد.

اکنون به کمک دستور fprintf در نرم افزار MATLAB یک فایل متنی با پسوند txt. بسازید که حاوی ۱۰۰ نمونه از یک دوره تناوب سینکال سینوسی باشد. در این رابطه می‌توانید از کد شکل ۲۳-۲ استفاده نمایید. در برنامه C این داده‌ها را با قابلیت Breakpoint بخوانید و در پنجره گراف نمایش دهید. همچنین این داده‌ها را در ۲ ضرب کنید و در یک فایل بنویسید.

```
fid=fopen('sine_sample.txt','wt');
fprintf(fid,'1651 4 0 0 0x190 \n')
for i=0:99
    fprintf(fid,'%f \n',sin(2*pi/100*i))
end
fclose(fid)
```

شکل ۲۳-۲ کد نمونه برای ایجاد یک فایل از MATLAB

۶-۳-۶. قسمت پنجم: فایل memory map (اختیاری)

با استفاده از فایل memory map (یک فایل با پسوند .map) در پوشه debug می‌باشد که بعد از پروژه ساخته می‌شود، یک بار حجم فضای کد را وقتی از تابع printf() در کد استفاده شده است تعیین کنید و یک بار وقتی دستور printf() حذف شود (دستور printf() توضیح^۱ شود). به این ترتیب حجم کد برای تابع printf را تعیین کنید.

سؤال: به نظر شما حضور تابع printf در کدی که قرار است در پردازنده اجرا شود به چه معنایی است؟

۶-۳-۷. قسمت ششم: استک (اختیاری)

الف) پروژه ای جدید ایجاد کنید و کد شکل ۲۴-۲ را در یک فایل C وارد کنید و به پروژه اضافه کنید. نتیجه اجرای این کد بر روی پردازنده را بررسی کنید. حدس می‌زنید علت رفتار غیر عادی برنامه چیست؟

```
#include <stdio.h>

main()
{
int A[1200];
    int i;

    for (i=0; i<1200; i++)
        A[i]=i;
    printf("Hello :-) ");
}
```

شکل ۲۴-۲ کد تمرین ۲

ب) با قرار دادن خط int A[1200] ; قبل از تابع main() یا به عبارت دیگر تعریف متغیر A به صورت سراسری، یک بار دیگر نتیجه اجرای برنامه را بر روی پردازنده بررسی کنید.
پ) یک بار دیگر کد شکل ۲۴-۲ را بر روی پردازنده اجرا کنید، اما این بار طول استک را تغییر دهید. برای این منظور از منوی Debug گزینه... Linker و سپس Basic در قسمت

¹ Comment

Stack Size (-stack) مقدار 0x2000 را وارد کنید. آیا نتیجه اجرا برنامه معقول شد؟

نکته: در پردازنده‌های TI مانند تعداد دیگری از زبانهای برنامه نویسی، محل ذخیره متغیرهای محلی، آدرس بازگشت از توابع، وضعیت جاری پردازنده هنگام رخداد وقفه، درون استک است. در صورتی که فضای اختصاص یافته به استک پر شود، پردازنده بدون توجه به پر شدن استک داده‌ها را در حافظه در ادامه فضای استک می‌نویسد و به این ترتیب محتوای قسمتهای دیگری از حافظه را خراب می‌کند. جلوگیری از سرریز استک بر عهده برنامه نویس است. بنابراین در صورت لزوم می‌توان به روش قسمت پ طول استک را افزایش داد و از تعریف متغیرهای محلی بزرگ جلوگیری کرد. به طور پیش فرض طول استک 0x400 کلمه می‌باشد.

۲-۴. ضمایم

۱-۴. ساختار کلی فایل اسembly

گاهی اوقات برای داشتن کد برنامه بهینه از لحاظ زمان اجرا و حجم کد، لازم است الگوریتم مورد نظر یا قسمتی از آن را به زبان اسembly پردازنده مورد نظر بنویسیم. دستورات اسembly به ساختار پردازنده بستگی دارند. فرمت یک کد اسembly برای پردازنده‌های خانواده C6000 به صورت زیر است:

Label || [] Instruction Unit Operands ; Comments

Label: در صورت وجود، آدرس یا خانه حافظه که دستور یا داده در آن قرار دارد مشخص می‌کند.
B1 باید از اولین ستون شروع شود.

| : در صورت وجود، نشان می‌دهد این دستور همزمان با دستور قبلی اجرا می‌شود.

[] : این قسمت در صورت وجود، قابلیت اجرای شرطی دستور را فراهم می‌کند. در داخل براکت باید یکی از رجیسترها A1، A2، B0، B1 یا B2 را نوشت. مثلاً اگر [A2] نوشته شود به این معنی است که در صورت غیر صفر بودن رجیستر A2 آن خط اجرا گردد و یا [!A2] به معنی اجرای آن خط در صورت صفر بودن رجیستر A2 است.

Instruction: می‌تواند شامل یک راهنمای اسembler باشد یا یک دستور اجرایی. این بخش نمی‌تواند از ستون اول شروع شود.

Unit: این بخش دلخواه مشخص می‌کند دستور در کدام واحد عملیاتی پردازنده اجرا شود.

Operands: عملوند های دستور می‌باشد.

Comments: بعد از علامت ; آنچه نوشته شود به عنوان توضیح در نظر گرفته می‌شود. اگر توضیح از اولین ستون شروع شود می‌توان برای مشخص کردن آن از علامت * نیز استفاده کرد.

مثال:

```
ADD .L1 A3, A7, A7 ; add A3+A7-> A7
```

دستور فوق مقدار درون رجیستر A3 را با مقدار درون رجیستر A7 جمع می‌زند و حاصل را درون رجیستر A7 ذخیره می‌کند. واحد عملیاتی L1 برای این محاسبه به کار می‌رود. که البته تعیین واحد عملیاتی اختیاری است.

مثال:

```
MPY .M2 A7, B7, B6 ; multiply 16 LSBs of A7 and B7 ->B6
|| MPYH .M1 A7, B7, A6 ; multiply 16 MSBs of A7 and B7 -> A6
```

این مثال نشان می‌دهد می‌توان دو عدد ۱۶ بیتی را همزمان ضرب نمود.

مثال:

```
LDH .D2 *B2++, B7
```

دستور فوق ۱۶ بیت (نیم کلمه) از حافظه که آدرس آن در رجیستر B2 قرار دارد به رجیستر B7 منتقل می‌کند. سپس محتوای رجیستر B2، واحد افزایش می‌یابد تا به نیم کلمه بعدی در حافظه اشاره کند. دستور LDW یک کلمه ۳۲ بیتی را بار می‌کند.

در فصل اول و همچنین در قسمت ضمائم لیست دستورات اسمنلی برای پردازنده‌های سری C6000 و لیست دستورات floating-point برای پردازنده‌های سری C67x آورده شده است. برای آشنایی با دستورهای اسمنلی و نحوه استفاده از آنها می‌توان از TMS320C6000 CPU and Instruction Set Reference Guide [۴] استفاده کرد.

در کدهای اسembler گاهی از راهنمای اسembler^۱ استفاده می‌شود. راهنمای اسembler دستور نیستند بلکه پیغامی برای اسembler هستند. به عنوان مثال "my_buffer" در کد اسembler بخشی از کد یا داده را به نام sect مشخص می‌کند. همچنین مشخص می‌کند عدد بعد از آن ۱۶ بیتی است یا float. مشخص می‌کند عدد بعد از آن عدد حقیقی ۳۲ بیتی (32-bit IEEE single precision) می‌باشد.

به طور کلی نوشتمن کد اسembler برای پردازنده‌های C6000 با دست کار دشواری است زیرا در این پردازنده‌ها چندین واحد اجرایی وجود دارد همچنین وجود خط لوله^۲ چند سطحی و دستورات با زمان اجرای متفاوت بر این پیچیدگی می‌افزاید. ولی از طرف دیگر برخی قابلیتهای سخت افزاری نظیر سیرکولار بافر مستقیماً از طریق کد C در دسترس نیستند.

۲-۴ فایل linker command

در برنامه نویسی برای پردازنده‌های TI، به طور کلی قرار دادن قسمتهای مختلف کد و داده‌ها و متغیرها در حافظه بر عهده برنامه نویس است. این کار در دو مرحله انجام می‌شود در مرحله اول بخش‌های^۳ مختلف کد و متغیرها در برنامه نام‌گذاری می‌شوند و در مرحله دوم در فایلی با پسوند cmd. مشخص می‌شود که هر بخش در چه آدرسی از حافظه قرار بگیرد. فایل command باید به پروژه اضافه گردد. در فایلهای C بخش‌های مختلف به طور خودکار مشخص می‌شوند. مثلاً کد برنامه در بخشی به نام .text، متغیرها در بخشی به نام .bss، مقادیر ثابت در بخش const، استک در بخش stack، مقادیر اولیه متغیرها در بخش cinit. و... قرار می‌گیرند. البته بخش‌های دیگری هم وجود دارند. برای آگاهی از بخش‌های مختلف برنامه بعد از build پروژه فایل متند با پسوند map. را در CCS باز کنید. در این فایل نام بخش‌های مختلف و طول آنها در حافظه و آدرس قرار گرفتن آنها در حافظه مشخص شده است و از آن با عنوان فایل memory map یاد می‌شود.

¹ Assembler directive

²Pipeline

³ Sections

بخشها دو دسته هستند، دسته ای initialized هستند مثل بخش‌های کد برنامه و ثوابت، و دسته ای uninitialized هستند مثل بخش‌های مربوط به قرار گرفتن متغیرها. در فایلهای اسambilی برای مشخص کردن بخش‌های initialized از راهنمای sect. با ساختار زیر استفاده می‌شود.

.sect "نام بخش"

کدهای بعد از این خط در بخشی به اسم نام بخش قرار می‌گیرند. همچنین برای مشخص کردن بخش‌های uninitialized (تعریف متغیرها) از راهنمای usect. در فایلهای اسambilی استفاده می‌شود. ساختار تعریف متغیر در فایل اسambilی به صورت زیر است:

طول حافظه , "اسم دلخواه برای بخش" usect. آدرس شروع

_Array	.usect	"var"	20
--------	--------	-------	----

در مرحله دوم برای قرار دادن بخش‌های مختلف در آدرس‌های مختلف در حافظه در فایل command از ساختار زیر استفاده می‌کنیم.

MEMORY

{

 اسم جدید origin= ، آدرس شروع طول

 اسم جدید origin= ، آدرس شروع طول

 " " " "

}

SECTIONS

{

 اسم جدید < { } : اسم بخش در فایل C یا اسambilی

 اسم جدید < { } : اسم بخش در فایل C یا اسambilی

}

برای تخصیص بخشهای مخاطب مختلف حافظه در فایل command لازم است ابتدا برنامه نویس با ساختار حافظه^۱ پردازنده مورد نظر از طریق مطالعه برگه اطلاعاتی^۲ آن آشنا شود. ساختار حافظه پردازنده 6713 به طور خلاصه در جدول ۱-۴ نشان داده شده است. شکل ۷-۲ یک فایل command برای پردازنده C6713 را نشان می‌دهد.

یکی از بخشهای مهم، بخشی است که محل قرار گرفتن جدول سرویس وقفه^۳ را مشخص می‌کند. با این بخش در آزمایش دوم آشنا می‌شوید.

برای کسب اطلاعات بیشتر راجع به فایل command می‌توان به مرجع TMS320C6000 Assembly [۱۰] و [۱۵] Language Tools User's Guide رجوع کنید.

^۱ Memory map

^۲ Data sheet

^۳ Interrupt service table

۳ آزمایش دوم: نمونه برداری و تولید سیگنال آنالوگ در

DSK6713

۱-۳ هدف

در این آزمایش با مفاهیمی از پردازنده‌های TI نظری و قوه، پورت سریال، کدک AIC23، CSL، خواندن و نوشتن از کدک بورد DSK6713 آشنا می‌شویم. ضمناً با مفاهیم نمونه برداری و بازسازی سیگنال و همچنین نحوه پیاده سازی یک سیگنال ژنراتور دیجیتال آشنا می‌شویم.

۲-۳ مقدمه

طبق قضیه نایکوییست، اگر از سیگنال آنالوگ باند پایه به پهنهای باند w با نرخ حداقل $f_s = 2w$ نمونه برداری کنیم، می‌توان از روی نمونه‌های سیگنال، سیگنال اصلی را بازسازی نمود، به عبارت دیگر سیگنال آنالوگ و سیگنال نمونه‌برداری شده با هم معادل هستند. به این ترتیب برای پردازش دیجیتال سیگنال سه مرحله اصلی وجود دارد که عبارتند از نمونه برداری از سیگنال آنالوگ و کوانتیزاسیون نمونه‌ها، پردازش سیگنال دیجیتال و در نهایت تبدیل سیگنال دیجیتال به سیگنال آنالوگ (شکل ۱-۳).



شکل ۱-۳ پردازش دیجیتال سیگنال آنالوگ

در برد آموزشی DSK6713 از پردازنده دیجیتال TMS320C6713 استفاده می‌شود.^۱ ADC^۲ و DAC مورد استفاده در DSK6713، تراشه AIC23 ساخت شرکت TI است. این تراشه دارای قابلیتهای مختلفی می‌باشد که قبل از آنکه از آن استفاده شود، لازم است پیکربندی شود. در DSK6713 این کار توسط پردازنده TMS320C6713 از طریق پورت سریال شماره ۰ آن (McBSP 0) انجام می‌گیرد و توابع لازم برای این

^۱ Analog to digital convertor

^۲ Digital to analog convertor

تنظیمات در کتابخانه dsk6713bsl.lib که توسط شرکت سازنده برد DSK ارایه شده است، موجود می‌باشد. همچنین برای انتقال نمونه‌های کوانتیزه شده از AIC23 به پردازنده و برعکس، پورت سریال شماره ۱ پردازنده (McBSP 1) مورد استفاده قرار می‌گیرد. لازم به ذکر است که پورت سریال در این پردازنده قابلیتهای متنوعی دارد و لازم است برای کاربرد مور نیاز پیکربندی گردد. برای این کار باید رجیسترها مربوط به آن در پردازنده با مقادیر مشخصی، مقداردهی گردند که بدین منظور از توابع^۱ API موجود در کتابخانه CSL مربوط به پردازنده استفاده می‌گردد. این کتابخانه برای پردازنده TMS320C6713.lib و cs16713.lib نام دارد.

پردازنده DSP برای خواندن و نوشتمن از پورت سریال می‌تواند از روش‌های مختلفی استفاده کند که می‌توان به روش polling ، روش استفاده از وقفه و روش استفاده از DMA اشاره کرد. در روش polling پردازنده منتظر می‌ماند تا پورت سریال برای ارسال و یا دریافت آماده شود، سپس داده خود را برای ارسال به پورت سریال می‌فرستد و باز منتظر می‌مانند تا پورت سریال کار ارسال یا دریافت داده را تمام کند و دوباره برای ارسال و یا دریافت آماده شود و این روند تکرار می‌شود. در روش وقفه، پردازنده به انجام کارهای دیگر می‌پردازد. وقتی پورت سریال برای ارسال یا دریافت داده آماده شد وقفه‌ای به پردازنده می‌فرستد و پردازنده موقتاً انجام کارهای خود را رها می‌کند و در سرویس روتین وقفه، داده را برای پورت سریال ارسال و یا از آن دریافت می‌کند و سپس به ادامه انجام کارها بر می‌گردد.

در ادامه توضیحاتی برای خواندن و نوشتمن از کدک در این آزمایش و آزمایشهای بعدی آورده می‌شود. برای کار با کدک و خواندن و نوشتمن از آن از تعدادی تابع استفاده می‌کنیم. به عبارت دیگر یک الگوی ثابت در پروژه‌هایی که نیاز به خواندن و نوشتمن از کدک دارند به کار می‌بریم که توضیح آن آمده است و برای درک بهتر آن اطلاعات بخش ضمایم این فصل مفید هستند. در قسمت آزمایشها با انجام تعدادی آزمایش با استفاده از تعدادی تابع با نحوه خواندن و نوشتمن از کدک آشنا می‌شویم و با مفاهیمی از قبیل نمونه برداری، بازسازی سیگنال و تولید سیگنال به روش دیجیتال آشنا می‌شویم. در قسمت ضمایم این بخش، اطلاعات لازم برای درک بهتر عملکرد پردازنده و برنامه نویسی آن ارائه شده است. برای استفاده از ادوات جانبی پردازنده‌های TI در

¹ Application programming interface

² Chip support library

برنامه C از توابعی که در کتابخانه CSL می‌باشد کمک گرفته می‌شود که در بخش ضمایم مراحل استفاده از آنها توضیح داده شده است. سپس پورت سریال به عنوان یک تجهیز جانبی مهم به طور کاملاً اجمالی معرفی می‌گردد و به دنبال آن بخش ADC/DAC یا کدک AIC23 معرفی می‌شود. در ادامه با روش کار و برنامه ریزی وقفه‌ها در پردازنده‌های DSP سری 6000 شرکت TI آشنا می‌شویم و به عنوان نمونه وقفه INT15 را برای خواندن از پورت سریال 1 پردازنده برنامه ریزی می‌کنیم.

۳-۳. توابع مورد استفاده در آزمایشگاه برای خواندن و نوشتمن از کدک

برای خواندن از کدک و نوشتمن در آن ابتدا باید تنظیمات اولیه‌ای انجام داد و کدهای ثابتی در برنامه نوشته باشند. به این منظور در پروژه‌هایی که به تبادل داده با کدک نیاز است، از دو فایل با نامهای init.c و init.h استفاده می‌کنیم، به این ترتیب که این دو فایل را در پوشش پروژه خود کپی می‌کنیم و فایل "init.c" را به پروژه اضافه می‌کنیم. همچنین فایل "init.h" را به ابتدای برنامه (اول فایل c حاوی تابع main) اضافه می‌کنیم:

```
#include "init.h"
```

برای خواندن و نوشتمن از کدک می‌توانیم از یکی از دو روش polling یا وقفه استفاده کنیم که انتخاب هر کدام مستلزم تنظیماتی متناسب با خود است. توابع مورد نیاز برای این تنظیمات در فایل init.c پیاده‌سازی شده‌اند و فقط کافی است که با آرگومانهای مناسب فراخوانی شوند. در صورتی که بخواهیم از روش polling استفاده کنیم فقط در ابتدای تابع main بعد از قسمت تعریف متغیرها، تابع comm_poll(fsampling, src) را صدای می‌زنیم که فرکانس نمونه‌برداری و src ورودی کدک را تعیین می‌کنند. مقدار fsampling یکی از مقادیر DSK6713_AIC23_FREQ_NNKHZ می‌باشد که مقادیر مجاز NN در جدول ۱-۳ آمده است. مثلاً فرکانس نمونه‌برداری را برابر ۱۶ KHz تعیین کرد. src ورودی کدک را تعیین می‌کند که می‌تواند یکی از دو مقدار DSK6713_AIC23_INPUT_LINE (Line In) یا DSK6713_AIC23_INPUT_MIC (Microphone برد) باشد.

در صورتی که روش خواندن و نوشتمن با استفاده از وقفه پورت سریال را انتخاب کنیم، لازم است

کارهای زیر را انجام دهیم:

- فایل IST.asm (شکل ۱۲-۳) را به پروژه خود اضافه کنید. این فایل آدرس شروع توابع اینترپرها را مشخص می‌کند.
- در ابتدای تابع main() بعد از تعریف متغیرها و قبل از استفاده از کدک، تابع comm_intr(fsampling, src) را صدا بزنید.
- در کد C دستورات سرویس وقفه را در تابع سرویس روتین وقفه، شبیه کد زیر، در قسمت Statement بنویسید.

```
interrupt void serialPortRcvISR()
{
    Statements
}
```

گرچه که این تابع برای وقفه دریافت پورت سریال می‌باشد، اما در اینجا از آن هم برای نوشتن در کدک می‌توانیم استفاده کنیم و هم برای خواندن از آن.

همان طور که می‌دانیم کدک AIC23 یک کدک دو کاناله (استریو) است. در تنظیمات انجام شده برای آن در هر ارسال به آن یا هر دریافت از آن یک فریم ۳۲ بیتی برای آن فرستاده می‌شود یا از آن دریافت می‌شود که ۱۶ بیت کم ارزش آن نمونه سیگنال آنالوگ کانال راست و ۱۶ بیت پر ارزش آن نمونه سیگنال آنالوگ کانال چپ می‌باشد. به این منظور در فایل init.c شش تابع برای ارتباط با کدک و دنیای آنالوگ نوشته شده است. برای خواندن از کدک می‌توان یکی از توابع زیر را به کار گرفت:

```
Uint32 input_sample();
short input_left_sample();
short input_right_sample();
```

که طبیعتاً هیچ کدام ورودی نمی‌گیرند. تابع اول یک متغیر ۳۲ بیتی بر می‌گرداند که همان طور که انتظار داریم ۱۶ بیت کم ارزش نمونه سیگنال کانال راست کدک و ۱۶ بیت پر ارزش نمونه سیگنال کانال چپ کدک از پورتی که ابتدای برنامه تعیین کردیم (MIC یا LINE) می‌باشد. دو تابع بعدی به ترتیب برای خواندن از

کانالهای چپ و راست کدک به کار می‌روند.

به همین ترتیب سه تابع زیر برای نوشتن در کدک مورد استفاده قرار می‌گیرد که همزمان در پورت Line و Headphone out می‌نویسند.

```
void output_sample(int);
void output_left_sample(short);
void output_right_sample(short);
```

با توجه به نحوه تعریف شش تابع فوق در فایل init.c، برای استفاده از این توابع فرقی نمی‌کند از کدام روش polling یا وقفه برای خواندن/نوشتن از کدک استفاده کرده باشیم.

۳-۴ آزمایش

۳-۴-۱ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS، نرم افزار MATLAB، بورد DSK C6713، اسیلوسکوپ و سیگنال ژنراتور

۳-۴-۲ قسمت اول: نمونه برداری و تولید سیگنال آنالوگ در بورد DSK 6713

در این قسمت برنامه‌ای می‌نویسید که از سیگنال آنالوگ وردی نمونه‌برداری می‌کند و سیگنال بازسازی شده را در پورتهای آنالوگ headphone و line out می‌نویسد. برای این منظور کارهای زیر را انجام دهید:

۱- پورت Line In بورد DSK را به سیگنال ژنراتور وصل کنید.

برای جلوگیری از آسیب دیدن بورد DSK توجه کنید دامنه سیگنال ورودی از ۱V RMS بیشتر نشود.

۲- پورت Line Out بورد DSK را به اسیلوسکوپ وصل کنید.

در صورتی که از کارت صوتی کامپیوتر و نرم‌افزارهایی نظیر Scope به عنوان سیگنال ژنراتور و اسیلوسکوپ استفاده می‌کنید، پورت Line In بورد DSK را به پورت Line Out کامپیوتر، و پورت Line Out کارت صوتی کامپیوتر وصل کنید و برنامه Scope را روی کامپیوتر اجرا کنید. بورد را به پورت Line In کارت صوتی کامپیوتر وصل کنید و برنامه Scope را روی کامپیوتر اجرا کنید.

۳- پروژه ای در نرم افزار CCS ایجاد کنید. فایلهای کتابخانه‌ای لازم (lib) dsk6713bsl.lib و rts6700.lib و csl6713.lib را به پروژه خود اضافه کنید. آدرس دو کتابخانه آخر C:\CCStudio_v3.3\C6000\csl\lib و C:\CCStudio_v3.3\C6000\dsk6713\lib می‌باشد. این دو کتابخانه هنگام استفاده از فایل init.c مورد استفاده قرار می‌گیرند.

۴- در قسمت Build Option در زبانه Compiler در دسته Preprocessor، در بخش Path (i): آدرس هدر فایلهای dsk6713_aic23.h و dsk6713.h را وارد کنید که آدرس آنها C:\CCStudio_v3.3\C6000\dsk6713\include می‌باشد.

۵- در قسمت Memory در زبانه Compiler در دسته Advanced، در بخش Build Option گزینه Models: Far (mem_model:data:far) را انتخاب کنید.

۶- فایل init.c و init.h را در پوشه پروژه خود کپی کنید.

۷- فایل init.c را به پروژه اضافه کنید.

۸- در یک فایل C برنامه‌ای بنویسید که با روش polling سیگنال آنالوگ را از کانال چپ پورت Line In بورد DSK بخواند و در پورت خروجی Line Out بنویسد. فرکانس نمونه برداری را 16000Hz در نظر بگیرید. فایل C را در پوشه پروژه ذخیره نمایید و آن را به پروژه اضافه کنید.

۹- برنامه خود را build کنید و در صورتی که بدون خطا بود در پردازنده بار و سپس اجرا کنید.

۱۰- در سیگنال ژنراتور، سیگنال سینوسی با فرکانس 1KHz و دامنه 0.5V انتخاب کنید و کانکتور قرمز را به سیگنال ژنراتور وصل نمایید. سیگنال بازسازی شده را مشاهده نمایید. آیا همه چیز همان طور است که انتظار داریم؟

۱۱- فرکانس سیگنال ژنراتور را تا حدود فرکانس 20KHz زیاد کنید و سیگنال بازسازی شده را مشاهده کنید. سوال: تا چه فرکانس‌هایی این سیستم درست کار می‌کند؟ چرا؟

۱۲- شکل موج سیگنال ژنراتور را مربعی با فرکانس 100HZ و دامنه 5V انتخاب کنید. همان طور که مشاهده می‌کنید در لبه‌های پالس مربعی بازسازی شده نوساناتی وجود دارد. سوال: علت آن چیست؟

۱۲- فرکانس تکرار پالس مربعی سیگنال ژنراتور را به 2 KHz افزایش دهید. سوال: شکل موج

سیگنال بازسازی شده چگونه است؟ چرا؟

۱۴- در سیگنال ژنراتور سیگنال را نویز سفید تنظیم کنید. سیگنال بازسازی شده را روی اسیلوسکوپ مشاهده نمایید. سپس با استفاده از قابلیت نمایش طیف سیگنال در اسیلوسکوپ (گزینه FFT در زیربخش دکمه MATH MENU در اسیلوسکوپ) طیف سیگنال بازسازی شده را مشاهده نمایید. سوال: این پاسخ فرکانسی چه چیزی را نشان می‌دهد؟ فرکانس قطع آن چقدر است؟

۳-۴- قسمت دوم: تولید سیگنال با استفاده از وقفه پورت سریال

در این قسمت سیگنالهای سینوسی و مربعی با فرکانس ثابت با به کارگیری قابلیت وقفه پورت سریال تولید می‌کنیم. برای این کار از نمونه های سیگنال ذخیره شده در حافظه استفاده می‌کنیم. روش کار به این صورت است که اگر فرکانس نمونه برداری F_s باشد و N نمونه از دوره تناوب سیگنال را در حافظه داشته باشیم، با ارسال پیاپی این N نمونه یک سیگنال با فرکانس تکرار $\frac{F_s}{N}$ خواهیم داشت، که البته باید پهنای باند سیگنال از $\frac{F_s}{2}$ کمتر باشد.

۱- یک پروژه جدید در نرم افزار CCS ایجاد کنید و فایلهای لازم را به آن اضافه کنید. لازم به ذکر است که چون در این قسمت از روش وقفه پورت سریال استفاده می‌کنیم باید فایل IST.asm را نیز به پروژه اضافه کنید. فرکانس نمونه برداری $F_s = 16000$ را انتخاب کنید.

۲- در ابتدای تابع main() بعد از تعریف متغیرها تابع comm_intr(...) را صدا بزنید تا تنظیمات مربوط به وقفه پورت سریال را انجام دهد

۳- با استفاده از جدول زیر یک سیگنال سینوسی بر روی کanal چپ کدک ایجاد کنید.
`short table_sine[16]={0, 3827, 7071, 9239, 10000, 9239, 7071, 3827, 0, -3827, -7071, -9239, -10000, -9239, -7071, -3827};`

۴- سرویس روتین وقفه، می‌تواند شبیه شکل ۳- باشد.

```
interrupt void serialPortRcvISR()
```

```
{
    output_right_sample((short) table_sine[index++]);
    if (index==16) index=0;
}
```

شکل ۲-۳ سرویس روتین وقفه برای تولید سیگنال سینوسی

- ۵- با اجرای برنامه از صحت عملکرد آن اطمینان حاصل کنید.
- ۶- برنامه خود را برای تولید سیگنال مربعی با فرکانس 1 KHz تغییر دهید. تنها کافی است مقادیر table_sine را تغییر دهید.
- ۷- حال برنامه خود را به گونه‌ای تغییر دهید که بر روی کanal چپ سیگنال سینوسی و بر روی کanal راست، سیگنال مربعی تولید شده را بدهد.
- ۸- سؤال: برای تولید یک سیگنال با فرکانس تکرار f_0 با این روش چه محدودیتی بر روی سیگنال وجود دارد؟
- ۹- سؤال: آیا می‌توانید با این روش یک سیگنال سینوسی با فرکانس 6 KHz تولید کنید.
- ۱۰- عملکرد یک Arbitrary function generator نظیر مدل 33521A ساخت شرکت Agilent که در آزمایشگاه موجود است، شبیه روشی است که در این قسمت برای ساخت سیگنال به کار برده شد.
- سؤال: به نظر شما برای ساخت یک سیگنال sinc متناوب با فرکانس تکرار $\frac{F_s}{N}$ چه تعداد نمونه از سیگنال sinc لازم دارد؟ عدد مورد نظر خود را با تعداد نمونه که سیگنال ژنراتور آزمایشگاه برای تولید سیگنال sinc به کار می‌گیرد مقایسه نمایید.

۴-۴ قسمت سوم: تولید سیگنال سینوسی با polling

در این قسمت در پورتهای آنالوگ خروجی کدک سیگنال سینوسی با دامنه و فرکانس قابل تنظیم ایجاد می‌کنیم. برای نوشتتن در کدک از روش polling استفاده می‌کنیم.

برای تولید سیگنال سینوسی در کد $\sin()$ استفاده کنید. به این ترتیب لازم است هدر فایل math.h را در فایل C خود اضافه کنید.

هدف تولید فایل سینوسی $s(t) = A \sin(2\pi f_0 t)$ می‌باشد. نمونه‌های این سیگنال با فرکانس نمونه

برداری f_s یا زمان نمونه برداری $T_s = \frac{1}{f_s}$ تولید می‌شوند. یعنی:

$$s[n] = A \sin(2\pi f_0 n T_s) = A \sin\left(2\pi n \frac{f_0}{f_s}\right) = \sin(n\Delta)$$

که در آن $\Delta = 2\pi \frac{f_0}{f_s}$ می‌باشد. اگر $\theta[n] = \theta(nT) = n\Delta$ باشد،

آرگومان تابع سینوس در لحظه $n+1$ برابر است با $\theta[n+1] = (n+1)\Delta = \theta[n] + \Delta$. به این ترتیب نمونه سیگنال در لحظه $n+1$ از روی نمونه سیگنال در لحظه n به دست می‌آید. شکل ۳-۳ قسمتی از کد C برای تولید سیگنال سینوسی بر روی کانال راست است که لازم است آن را تکمیل کنید.

```
...
#define pi 3.14159265
short sample=0;
float Fs=16000.;
float f0=1000.;
float gain=15000;
float delta, twopi;
float angle=0;
float right;
void main()
{
    ...
    twopi=2.0*pi;
    while(1)
    {
        delta=twopi*f0/Fs;
        right=15000*sin(angle);
        sample= (short) right;
        output_right_sample(sample);
        angle += delta;
        if(angle>=twopi) angle-=twopi;
    }
}
```

شکل ۳-۳ قسمتی از کد C برای تولید سیگنال سینوسی بر روی کانال راست

سؤال: علت استفاده از ضریب 15000 برایتابع سینوس چیست؟

سؤال: علت استفاده از دستور if(angle>=twopi) angle-=twopi; چیست؟

نکته: همان طور که می‌دانید ورودی تابع output_sample از نوع int ، ۳۲ بیتی، می‌باشد، که ۱۶ بیت کم ارزش نمونه داده کانال راست و ۱۶ بیت با ارزش نمونه کانال چپ است. برای ترکیب دو متغیر short در یک متغیر int می‌توانید از ایده نوع داده union در زبان برنامه نویسی C استفاده کنید. شکل ۳-۳- نحوه استفاده از این ایده را نشان می‌دهد.

```
...
union {unsigned int uint;
        short      channel[2];
} AIC23_data;
...
main()
{
    ...
    AIC23_data.uint=0;
    AIC23_data.channel[LEFT]= (short) left;
    AIC23_data.channel[RIGHT]= (short) right;
    output_sample(AIC23_data.uint);
    ...
}
```

شکل ۴-۳ نحوه استفاده از ایده نوع داده union برای ترکیب دو داده ۱۶ بیتی در یک داده ۳۲ بیتی

۵-۳ تمرین

برنامه‌ای بنویسید که سیگنال باند پایه دلخواهی به پهنهای باند حداکثر ۳ KHz را دریافت کرده و به مدت ۰.۵ ثانیه آن را ضبط کند. سپس پس از وقفه‌ای به مدت ۱ ثانیه، آن را به خروجی ارسال کند.

۶-۳ ضمایم

۶-۳-۱ استفاده از توابع CSL^۱

برای کار با تجهیزات جانبی پردازنده‌های TI باید تجهیزات جانبی به نحو مطلوب پیکربندی گردد. یک روش برای این کار کد نوشتن اسمبلی و انتقال مقادیر مناسب به رجیسترها تجهیز جانبی مربوطه می‌باشد. TI برای C/C++ پیکربندی و کار با این تجهیزات تعدادی تابع در قالب کتابخانه CSL ارائه کرده است که در برنامه استفاده می‌شوند و با استفاده از آنها دیگر نیاز نیست با دستورهای اسمبلی رجیسترها پردازنده را برنامه ریزی نماییم. برای برنامه ریزی تجهیزات جانبی نظیر پورت سریال، تایмер، PLL و غیره از توابع مناسب آنها استفاده می‌شود. در ادامه مراحل استفاده از تابع CSL شرح داده شده است.

۱- در اولین خط برنامه C/C++ نوع چیپ تعریف شود. مثلاً برای سری 6713 می‌نویسیم:

```
#define CHIP_6713
```

۲- فایل کتابخانه‌ای مناسب را به پروژه اضافه می‌کنیم. فایلهای .lib. برای CSL ها همگی با کلمه `csl` شروع می‌شوند و در شاخه `csl` قرار دارند. مثلاً در سری 6713 محل فایلها در شاخه `C:\CCStudio_v3.3\CC6000\csl\lib` است و اسم فایل کتابخانه‌ای `csl6713.lib` می‌باشد. در صورتی که از مدل حافظه `large` یا `big endian` استفاده کرده باشیم به جای فایلهای معمولی فایلهای با پسوند `x` یا `e` را به پروژه اضافه می‌کنیم.

۳- کردن فایل header کلی یا عمومی به نام `csl.h` در خط دوم.

```
#include <csl.h>
```

۴- با توجه به نوع تجهیز جانبی که می‌خواهیم برنامه ریزی نماییم، یک فایل `csl` دیگر را نیز می‌کنیم. مثلاً برای پورت سریال می‌نویسیم:

```
#include <csl_mcbsp.h>
```

۵- در تابع `main()` پیش از استفاده از تابع `csl_init()` تابع `csl` را صدا می‌زنیم.

¹ Chip support library

۶- با استفاده از cs1 ها یک بخش به نام csldata. به وجود می آید که آن را به کمک فایل cmd. در آدرس مناسب در حافظه قرار می دهیم.

۷- بسته به آنکه از کدام یک از تجهیزات جانبی پردازنده استفاده می کنیم، یک متغیر از نوع structure که تعریف آن در فایل هدر مربوطه آمده است، تعریف می کنیم. این متغیر معمولاً با نام تجهیز جانبی و کلمه Config مشخص شده است و با مقادیر رجیسترهای کنترلی تجهیز جانبی مقدار دهی می شود. به عنوان مثال برای پورت سریال تعریف متغیر مربوطه به صورت زیر می باشد:

`MCBSP_Config myconfig={.....};`

`MCBSP_Config myconfig={0x1000, 0x0000, ...};`

مثال

برای تعیین مقادیر رجیسترهای تجهیز جانبی لازم است برگه اطلاعاتی آن تجهیز جانبی مطالعه گردد. این اطلاعات برای برخی تجهیزات جانبی در مرجع (TMS320C6000 DSP Peripherals Reference Guide) [V] یافت می شود.

۸- تعریف متغیری از نوع "اسم تجهیز جانبی" . به عنوان مثال برای پورت سریال متغیری به صورت زیر تعریف می کنیم.

`MCBSP_Handle hMcbsp;`

مثال

۹- فراخوانی تابع بازگشته تجهیز جانبی مربوطه. این تابع با نام "open()_اسم تجهیز جانبی" می باشد که خروجی آن از نوع متغیر تعریف شده در مرحله ۸ می باشد. در زیر نحوه استفاده از چنین تابعی برای پورت سریال آمده است. در اینجا اولین آرگومان تابع نشان می دهد پورت سریال ۰ باز شود و آرگومان دوم می گوید پورت سریال با مقادیر پیش فرض پیکربندی شود (در قسمت ۱۰ پیکربندی مورد نظر را اعمال می کنیم).

`hMcbsp=MCBSP_open(MCBSP_PORT0, MCBSP_OPEN_RESE);`

مثال

۱۰- فراخوانی تابع پیکربندی کننده مربوط به تجهیز جانبی به نام "config_اسم تجهیز جانبی" ، که ورودی آن متغیر مربوط به تجهیز جانبی مربوطه و همچنین متغیر حاوی مقادیر رجیسترهای تجهیز جانبی (که در قسمت ۷ تعریف شد) می باشد. مثلاً برای پورت سریال:

```
MCBSP_config(hMcbsp, &myconfig);
```

مثال

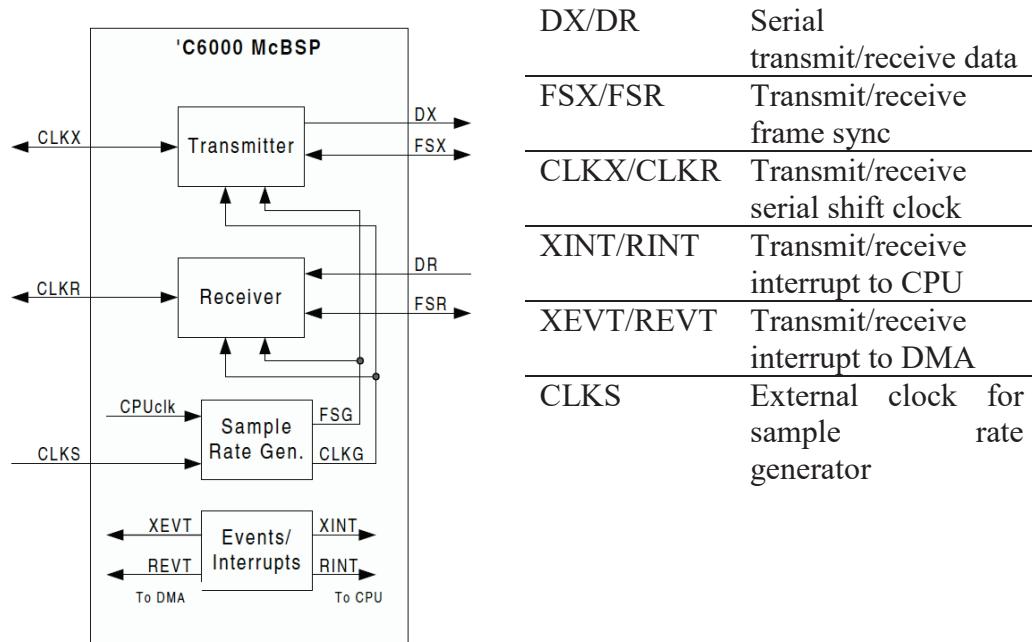
-۱۱ در صورت لزوم استفاده از توابع API (از کتابخانه CSL) برای کار با تجهیز جانبی مربوطه.

MCBSP_rrdy() برای پورت سریال توابع MCBSP_write() و MCBSP_read() یا () ممکن است مورد استفاده قرار بگیرند.

برای آشنایی با توابع CSL می‌توان در مرجع TMS320C6000 Chip Support Library API Library [۹] اطلاعات بیشتری پیدا کرد.

۲-۳-۲ پورت سریال^۱

یکی از تجهیزات جانبی پردازنده‌های شرکت TI پورت سریال (McBSP) می‌باشد. شکل ۲-۳ بلوک دیاگرام پورت سریال را برای پردازنده‌های خانواده C6000 نشان می‌دهد.



شکل ۲-۳ بلوک دیاگرام McBSP در خانواده 6000 از
پردازنده‌های TI

¹ Multi-Channel buffered serial port

بلوک دیاگرام کلی قسمت فرستنده پورت سریال در شکل ۳-۳ نشان داده شده است. CPU یا DMA یک کلمه ۳۲ بیتی در رجیستر^۱ DXR کپی می‌کند و سپس بیت XRDY را '۰' می‌کند. با لبه بالارونده FSX تعداد بیت قابل تنظیم به صورت سریال از رجیستر^۲ XSR خارج می‌شود. بعد از خارج شدن بیتها رجیستر XSR بیتها CPU به صورت موازی وارد XSR می‌شوند و بیت XRDY '۱' می‌شود. CPU می‌تواند با چک کردن بیت XRDY در صورت '۱' بودن آن کلمه بعدی را برای ارسال به پورت سریال بفرستد. همچنین در صورتی که فیلد XINTM از رجیستر^۳ SPCR برابر '۰۰' باشد وقتی XRDY از '۰' به '۱' تغییر وضعیت می‌دهد و قله XINT به CPU می‌رود به علاوه یک^۴ XEVT به DMA می‌فرستد.

بلوک دیاگرام کلی قسمت گیرنده پورت سریال در شکل ۴-۳ نشان داده شده است. وقتی^۵ FSR '۱' می‌شود بیتها دریافتی از پین DR پردازنده به صورت سریال وارد^۶ RSR می‌گردند. وقتی تعداد بیت مشخص و قابل تنظیم دریافت شد، ۳۲ بیت RSR به صورت موازی وارد^۷ RBR می‌شوند و سپس به رجیستر^۸ DRR بار می‌شوند. برداشتن تعداد بیت تنظیم شده از ۳۲ بیت DRR بر عهده برنامه نویس است. وقتی بیتها از RDR وارد DRR شدند، بیت RRDY '۱' می‌شود و در صورتی که فیلد RINTM در SPCR برابر '۰۰' باشد و RRDY از '۰' به '۱' تغییر وضعیت دهد پورت سریال، وقفه RINT را به CPU ارسال می‌کند و همچنین REVT نیز به EDMA می‌فرستد.

¹ Data transmit register

² Transmit shift register

³ Serial port control register

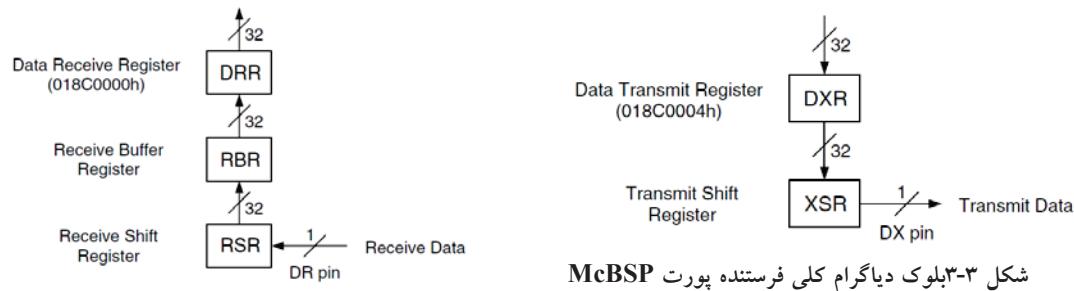
⁴ Transmit event

⁵ Receive frame synch

⁶ Receive shift register

⁷ Receive buffer register

⁸ Data receive register



شکل ۴-۳ بلوک دیاگرام کلی گیرنده پورت McBSP

شکل ۳-۳ بلوک دیاگرام کلی فرستنده پورت

برای تنظیم پارامترها و کار کردن با پورت سریال می‌توان از توابع CSL استفاده کرد، از جمله این توابع می‌توان به `(MCBSP_write()` و `(MCBSP_read()`) بخواندن از پورت سریال و نوشتن در آن، اشاره کرد. پورت سریال قابلیتهای بسیار متنوعی دارد که برای آشنایی بیشتر می‌توان به مرجع TMS320C6000 DSP [۸] Multichannel Buffered Serial (McBSP) Reference Guide مراجعه کرد.

۳-۶-۳ مشخصات کدک AIC23

نرخهای نمونه برداری که توسط کدک AIC23 پشتیبانی می‌شود در جدول ۱-۳ آمده است.

جدول ۱-۳ نرخهای نمونه برداری مجاز برای کدک AIC23

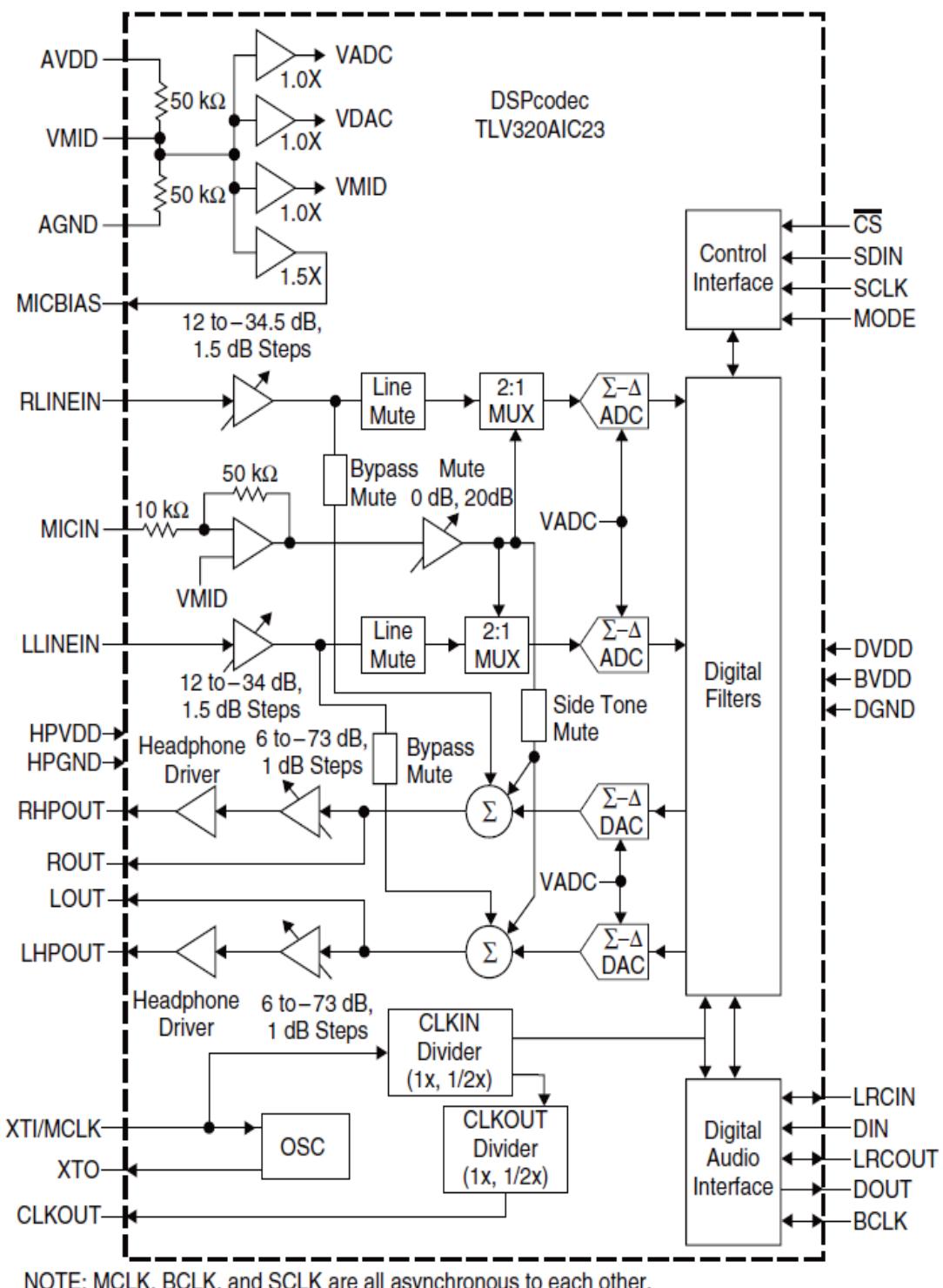
Sampling rate (KHz)	
8	44
16	48
24	96
32	

در ورودی ADC و خروجی DAC فیلترهای پایین گذر وجود دارند که در فرکانس $0.5f_s$ ، -6dB -بهره دارند. همچنین ماکریم سیگنال آنالوگ در ورودی و خروجی آن VRMS 1.0 می‌باشد. البته بین پورت وردي LINE IN بورد DSK و ورودی LINE IN از کدک روی بورد یک نصف کننده ولتاژ قرار دارد. در این کدک در مسیرهای LINE IN و HEADPHONE OUT بهره قابل تنظیم به ترتیب -34dB - 12dB - -73dB - -6dB قابل اعمال است.

برای تنظیم نرخ نمونه برداری از تابع `DSK6713_AIC23_setFreq(handle, freq_ID)` استفاده می‌شود. این تابع در کتابخانه `dsk6713bsl.lib` می‌باشد. برای استفاده از آن باید هدر فایلهای `dsk6713_aic23.h` و `dsk6713.h` در کد C درج شوند.

با تنظیمات پیش فرض AIC23 داده‌های نمونه برداری شده در فریمهای ۳۲ بیتی، ۱۶ بیت پرارزش، نمونه‌های کanal چپ و ۱۶ بیت کم ارزش نمونه‌های کanal راست می‌باشد.

شکل ۵-۳ بلوک دیاگرام داخلی کدک AIC23 را نشان می‌دهد.



شکل ۴-۳ بلوک دیاگرام کدک TLV320AIC23

۶-۴ وقفه

در پردازنده‌ها منظور از وقفه یک سیگنال آسنکرون به پردازنده است که نشان می‌دهد پردازنده اجرای عادی برنامه فعلی را متوقف نماید و سرویس مربوط به رخداد متناظر با وقفه اتفاق افتاده را انجام دهد. در پردازنده‌های سری C6000 تعداد ۱۶ منبع وقفه^۱ وجود دارد در حالی که تعداد ۱۲ وقفه CPU برای برنامه ریزی در دسترس است که نام و اولویت آنها در شکل ۶-۳ مشخص شده است. وقفه‌های RESET و NMI و^۲ و^۳ NMI در دسترس است که نام و اولویت آنها در شکل ۶-۳ مشخص شده است. وقفه‌های RESET و NMI بعد از وقفه NMI بالاترین اولویت را دارد و معمولاً^۴ تعدادی وقفه خارجی از پنهان‌پردازنده می‌آیند. برای آگاه کردن پردازنده از مشکلات سخت افزاری مورد استفاده قرار می‌گیرد.

Interrupt Name	Priority
RESET	Highest
NMI	
INT4	
INT5	
INT6	
INT7	
INT8	
INT9	
INT10	
INT11	
INT12	
INT13	
INT14	
INT15	Lowest

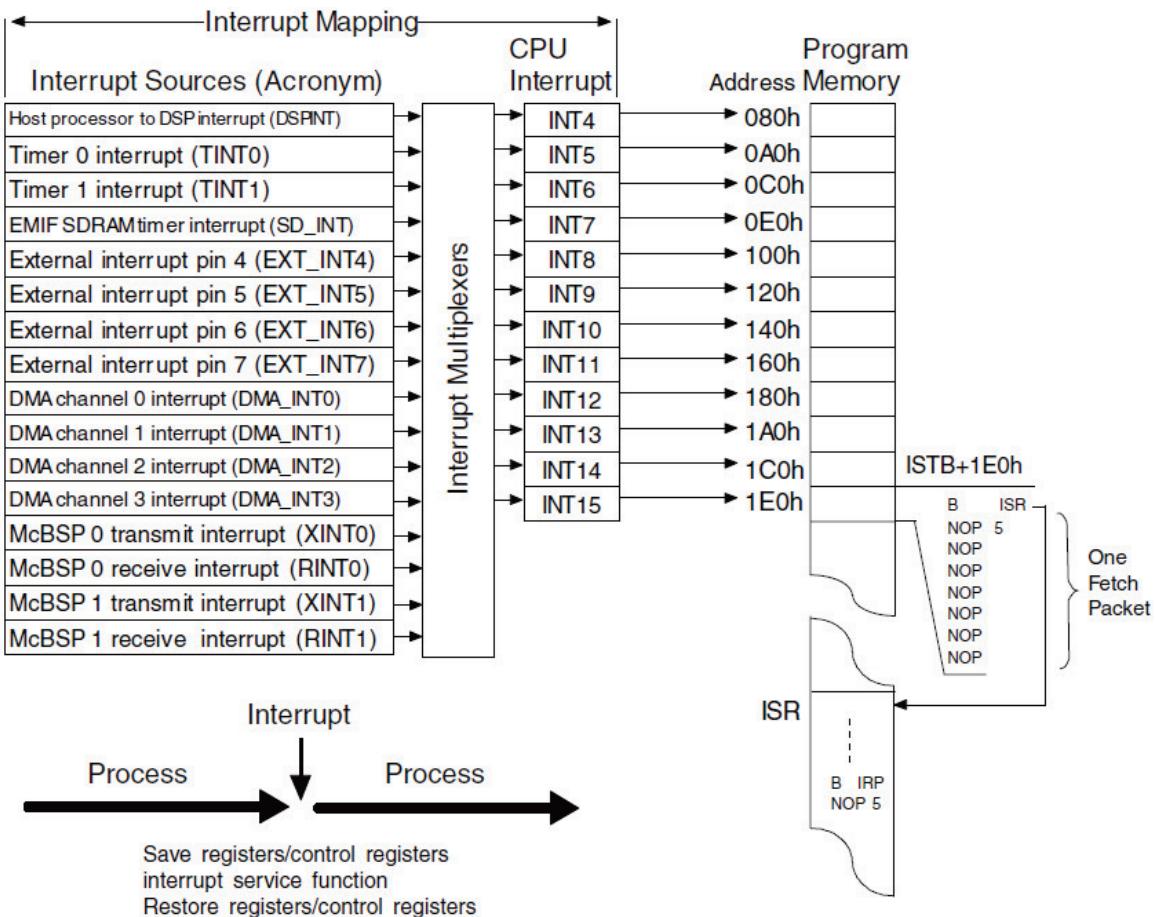
شکل ۶-۳ اولویت وقفه‌های CPU

لیست منابع وقفه در شکل ۷-۳ آمده است. به این ترتیب باید منابع وقفه مورد نیاز به وقفه‌های CPU نظری^۵ گردند که این کار از طریق تنظیم بیتهاي دو رجیستر Interrupt Multiplex Register انجام می‌گردد. لیست رجیسترهاي کنترلی وقفه در شکل ۸-۳ آمده‌اند.

¹ Interrupt source

² Non-Maskable interrupt

³ Map



شکل ۷-۳ لیست منابع وقفه و نحوه عملکرد وقفه

	Name	Description
CSR	Control Status Register	Globally set or disable interrupts
IER	Interrupt Enable Register	Enable interrupts. Bit <i>n</i> corresponds to INT _{<i>n</i>}
IFR	Interrupt Flags Register	Shows status of interrupts. Bit <i>n</i> corresponds to INT _{<i>n</i>}
ISR	Interrupt Set Register	Manually set flags in IFR
ICR	Interrupt Clear Register	Manually clear flags in IFR
ISTP	Interrupt Service Table Pointer	Pointer to the beginning of the interrupt service table
NRP	Nonmaskable Interrupt Return Pointer	Return address used on return from a nonmaskable interrupt
IRP	Interrupt Return Pointer	Return address used on return from a maskable interrupt

شکل ۸-۳ رجیسترها کنترلی وقفه

برای آنکه وقفه های maskable (INT04-INT15) سرویس داده شوند شرایط زیر باید برقرار گردند:

- بیت^۱ GIE در رجیستر^۲ CSR باید '۱' باشد.
- بیت^۳ NMIE از رجیستر^۴ IER باید '۱' باشد.
- بیت^۵ IE متناظر با وقفه CPU مورد نظر در رجیستر IER '۱' باشد.
- وقفه مورد نظر اتفاق بیفتد که در نتیجه بیت متناظر با آن در رجیستر^۶ IFR '۱' شده باشد و بیت متناظر با وقفه با اولویت بالاتر در IFR '۱' نباشد.

منظور از^۷ IST جدولی (قسمتی از حافظه) حاوی دستورات اجرایی^۸ برای وقفه ها می باشد. در این جدول برای هر وقفه یک ISFP^۹ که حاوی هشت دستور می باشد وجود دارد. یک^{۱۰} ISR ساده ممکن است در این جدول جا شود یا اینکه در ISFP مربوطه از دستور پرسن به ISR مربوطه استفاده شود. شکل ۹-۳ ساختار IST را نشان می دهد. با توجه به آنکه هر ISFP ۳۲ بایت است (هشت دستور و هر دستور چهار بایت) آدرس هر ISFP نسبت به آدرس ISFP قبلی ۳۲ بایت (20h) افزایش می یابد. شکل ۹-۳ یک ISFP را نشان می دهد که کاملاً در فضای مربوطه در IST جا شده است. در آخر سرویس وقفه از دستور IRP B یعنی بازگشت از وقفه استفاده می شود. در اینجا با توجه به ساختار pipeline در این پردازنده ها برای جلوگیری از اجرای پنج دستور بعدی در خانه های بعدی حافظه از دستور 5 NOP (معادل پنج دستور^{۱۱} NOP) استفاده می شود. IST را در یک فایل اسembly می نویسیم و آن را در یک بخش، مثلاً با نام vec. می گذاریم. سپس در فایل command این بخش را در آدرس مشخصی از حافظه قرار می دهیم.

¹ Global interrupt enable

² Control status register

³ Non-Maskable interrupt enable

⁴ Interrupt enable register

⁵ Interrupt enable

⁶ Interrupt flag register

⁷ Interrupt service table

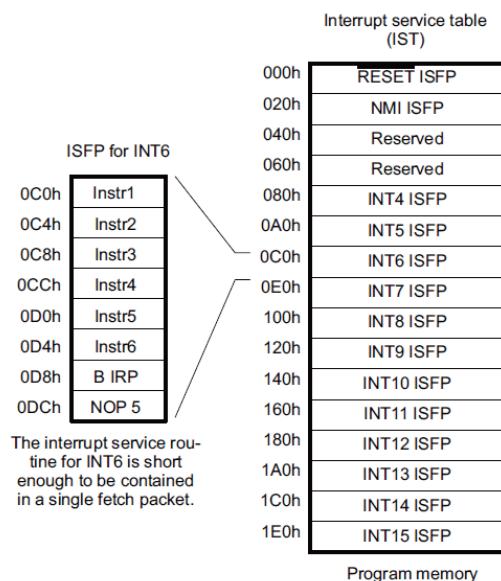
⁸ Fetch packet

⁹ Interrupt serive fetch packet

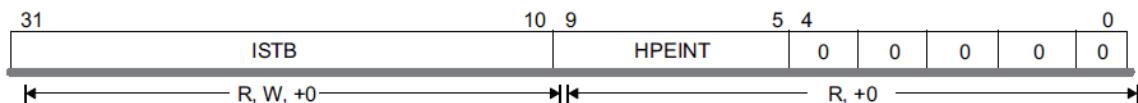
¹⁰ Interrupt service routine

¹¹ No operation

پردازنده از رجیستر^۱ ISTP برای تعیین آدرس شروع سرویس روتنی یک وقفه استفاده می‌کند. شکل ۱۰-۳ رجیستر ISTP و فیلد های آن را نشان می‌دهد. فیلد HPEINT با شماره وقفه رخ داده با اولویت بالاتر پر می‌شود و ISTB آدرس پایه برای IST می‌باشد. با توجه به تعداد بیتهای بخش های مختلف ISTP در می‌یابیم که آدرس شروع IST در حافظه بر حسب بایت باید مضربی از 1024 باشد. باید توجه داشت که در صورتی که آدرس شروع IST را مقداری غیر از 0 در نظر گرفتیم با توجه به آنکه بعد از وقفه RESET مقدار رجیستر ISTP برابر 0 می‌گردد، سرویس روتنی وقفه RESET همچنان باید در آدرس 0 حافظه قرار داشته باشد. شکل ۱۱-۳ مثالی از قرار دادن IST در حافظه به غیر از آدرس 0 می‌باشد.

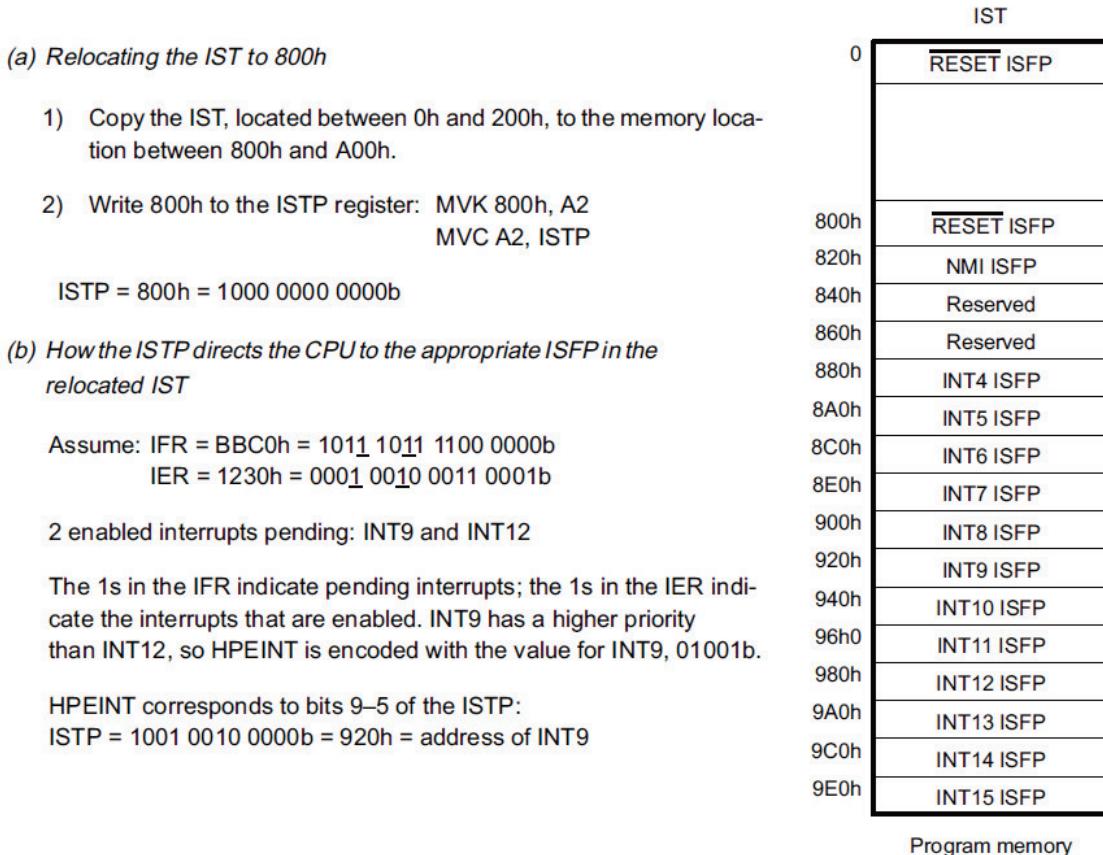


شکل ۹-۳ جدول IST و یک ISFP



شکل ۱۰-۳ رجیستر ISTP

^۱Interrupt service table pointer



شکل ۳-۱۱-۳، تغییر مکان جدول IST در حافظه

وقتی وقفه ای اتفاق می‌افتد پردازنده با ذخیره محل جاری اجرای برنامه، با استفاده از فیلد^۱ ISTB از رجیستر^۲ ISTP و شماره وقفه اتفاق افتاده، آدرس محل قرار گرفتن دستورات وقفه رخ داده را می‌یابد. به عنوان مثال در صورت رخ دادن وقفه INT15 پردازنده به آدرس ISTB+1E0h می‌رود.

برای تنظیم وقفه می‌توان با کدهای اسembly رجیسترها مرتبه را با مقادیر مشخص برنامه ریزی کرد. همچنین می‌توان با استفاده از توابع API در برنامه C تنظیمات مربوط به وقفه را انجام داد.

برای نوشتن تابع سرویس وقفه (ISR) در برنامه C از کلمه کلیدی interrupt استفاده می‌شود. در زیر ساختار کلی تابع وقفه آمده است.

¹ Interrupt service table base

² Interrupt service table pointer

```
interrupt void myInterSerRout()
{
}
```

در ادامه به عنوان مثال مراحل انجام کارهای لازم برای استفاده از وقفه (RINT1) McBSP 1 receive آمد است.

شکل ۱۲-۳ فایل اسمبلی IST را برای این منظور نشان می‌دهد. در فایل اسمبلی کلماتی که بعد از راهنمای ref. می‌آیند یعنی در فایل دیگری از برنامه تعریف شده‌اند و اینجا مورد استفاده قرار می‌گیرند (linker می‌فهمد برای یافتن تعریف آنها فایلهای دیگر را جستجو کند) و ".sect" و ".vec". نشان می‌دهد کدهای بعد از آن در بخشی به نام ".vec" در حافظه قرار گیرد. آدرس این بخش در فایل command مشخص می‌شود. شکل ۷-۲ فایل command را برای این پروژه نشان می‌دهد. به این ترتیب با توجه به آنکه مقدار پیش فرض ISTB برابر ۰ می‌باشد در اینجا لازم به تنظیم ISTP نمی‌باشد. همچنین لازم است منبع وقفه مورد نیاز را به یک وقفه CPU نظری کنیم که برای این منظور می‌توانیم از تابع IRQ_map() از کتابخانه cs16713.lib استفاده کرد. تابع hookint() برای کردن رجیسترها مربوط به وقفه با استفاده از توابع API از کتابخانه cs16713.lib نوشته شده است. این تابع در شکل ۱۳-۳ آمده است. در این تابع وقفه مربوط به پورت سریال به وقفه شماره ۱۵ پردازندۀ نظری می‌شود و این وقفه فعال می‌گردد. که البته پیش از آن وقفه‌های non-maskable فعال می‌گردند. برای جلوگیری از اختلال در عملکرد این تابع ابتدای این تابع کلیه وقفه‌ها غیر فعال می‌گردند و سپس آخر تابع وقفه‌ها فعال می‌شوند.

در ادامه باید تابع ISR برای وقفه مورد نظر نوشته شود. برای نوشتن تابع وقفه در C از کلمه کلیدی interrupt استفاده می‌شود. در اینجا با دریافت وقفه از پورت سریال محتوای رجیستر DDR از پورت سریال با استفاده از تابع MCBSP_read() خوانده می‌شود و سپس با ضرب مقدار خوانده شده در یک ضریب در رجیستر XDR از پورت سریال با استفاده از تابع MCBSP_write() نوشته می‌شود. این تابع در شکل ۱۴-۳ نشان داده شده است.

```

.global _c_int00
.global _serialPortRcvISR
.sect ".vec"
RESET: MVKL    .S2    _C_int00, B0
        MVKH    .S2    _c_int00, B0
        B       .S2    B0
        NOP
        NOP
        NOP
        NOP
        NOP
NMI:   .loop 8
        NOP
        .endloop
RESV1: .loop 8
        NOP
        .endloop
RESV2: .loop 8
        NOP
        .endloop
INT4:  .loop 8
        NOP
        .endloop
INT5:  .loop 8
        NOP
        .endloop
INT6:  .loop 8
        NOP
        .endloop
INT7:  .loop 8
        NOP
        .endloop
INT8:  .loop 8
        NOP
        .endloop
INT9:  .loop 8
        NOP
        .endloop
INT10: .loop 8
        NOP
        .endloop
INT11: .loop 8
        NOP
        .endloop
INT12: .loop 8
        NOP
        .endloop
INT13: .loop 8
        NOP
        .endloop

```

```

INT14: .loop 8
      NOP
      .endloop
INT15: MVKL     .S2    _serialPortRcvISR, B0
      MVKH     .S2    _serialPortRcvISR, B0
      B       .S2    B0
      NOP
      NOP
      NOP
      NOP
      NOP

```

شکل ۱۲-۳ کد اسambilی برای تعریف IST

```

void hookint()
{
    IRQ_globalDisable();           // Globally disables interrupts
    IRQ_nmiEnable();              // Enables the NMI interrupt
    IRQ_map(IRQ_EVT_RINT1,15);    // Maps an event to a physical
interrupt
    IRQ_enable(IRQ_EVT_RINT1);    // Enables the event
    IRQ_globalEnable();           // Globally enables interrupts
}

```

شکل ۱۳-۳ تابع برای انجام برخی موارد مرتبط با وقفه

```

interrupt void serialPortRcvISR()
{
    int sample;

sample=MCBSP_read(hCodec);
    sample*=gain;
    MCBSP_write(hCodec, sample);
}

```

شکل ۱۴-۳ تابع ISR

۱۴-۵-۳ مراحل خواندن و نوشتن در کدک بورد C6713 DSK

برای نوشتن برنامه ای که با استفاده از کدک از سیگنال آنالوگ نمونه برداری کند. چند مرحله باید انجام شود، از جمله پیکربندی کدک و پیکربندی پورت سریال، همچنین تنظیم وقفه برای دریافت وقفه از پورت سریال.

وقتی با بورد C6713 DSK کار می کنیم سیگنال آنالوگ از طریق پورتهای IN و MIC به کدک AIC23 وارد می شوند و ارتباط پردازنده با کدک نیز از طریق پورت سریال McBSP0 و McBSP1 می باشد. اولی برای

تنظیم و کنترل کدک به کار می‌رود و دومی برای انتقال داده بین پردازنده و کدک. برای خواندن و نوشتن از کدک توابعی از csl6713.lib و dsk6713bsl.lib در دسترس می‌باشند. برای استفاده از آنها باید کارهای زیر انجام شود.

۱- تعریف نوع چیپ در اول برنامه C

```
#define CHIP_6713
```

۲- درج هدر فایلهای زیر بالای کد C

```
#include <dsk6713.h>
#include <dsk6713_aic23.h>
```

۳- مشخص کردن محل قرار گرفتن فایلهای هدر فوق در پنجره Build option قسمت Preprocessor-> include search path

۴- اضافه کردن کتابخانه های rts6700.lib و csl6713.lib و dsk6713bsl.lib به پروژه

۵- تعریف متغیر از نوع DSK6713_AIC23_Config که حاوی مقادیر رجیستر های کدک AIC23 برای تنظیم آن می‌باشد و مقدار دهی آن با مقدار DSK6713_AIC23_DEFAULTCONFIG. این مقدار در هدر فایل dsk6713_aic23.h تعریف شده است.

```
DSK6713_AIC23_Config config=DSK6713_AIC23_DEFAULTCONFIG;
```

۶- تعریف متغیری از نوع DSK6713_CodecHandle

```
DSK6713_AIC23_CodecHandle hCodec;
```

۷- در تابع main() تابع DSK6713_init() فراخوانی می‌شود.

۸- تابع DSK6713_AIC23_openCodec()

```
hCodec=DSK6713_AIC_openCodec(0, &config);
```

۹- برای تنظیم فرکانس نمونه برداری کدک تابع DSK6713_AIC23_setFREQ() از کتابخانه dsk6713bsl فراخوانی می‌شود. فرکانس نمونه برداری کدک یکی از مقادیر جدول ۱-۳ می‌باشد. آرگومانهای این تابع به ترتیب handle کدک است که نتیجه مقدار تابع DSK6713_AIC23_openCodec می‌باشد، و دیگری فرکانس نمونه برداری با مقداری نظری dsk6713_aic.h می‌باشد. این مقدار در هدر فایل DSK6713_AIC23_FREQ_16KHZ تعریف شده است.

۱۰- دو تابع DSK6713_AIC23_write() و DSK6713_AIC23_read() از کتابخانه dsk6713bsl.lib

برای خواندن و نوشتن از کدک می‌توانند مورد استفاده قرار بگیرند. این دو تابع با چک کردن بیتهاي RRDY و XRDY به دریافت و ارسال داده اقدام می‌کنند به این صورت که در

هنگام فراخوانی در صورتی که بیت RRDY برابر '1' باشد تابع DSK6713_AIC23_read() مقدار رجیستر DRR را می‌خواند و مقدار '1' بر می‌گرداند و در صورتی که بیت RRDY برابر '0' باشد این تابع مقدار '0' بر می‌گرداند. در زیر نحوه استفاده از این توابع نشان داده شده است.

```
while (! DSK6713_AIC23_read(hCodec, &sample))
while (!DSK6713_AIC23_write(hCodec, sample))
```

در حقیقت این دو تابع با کمک دو تابع MCBSP_write() و MCBSP_read() نوشته شده‌اند و بر مبنای polling بر xrdy و rrdy می‌باشند. می‌توان برای نوشتن و خواندن از کدک به جای دو تابع DSK6713_AIC23_write() و DSK6713_AIC23_read() مستقیماً از این دو تابع استفاده کرد.

- ۱۱ با استفاده از تابع DSK6713_AIC23_rset() می‌توان ورودی کدک را Line In یا MIC تعیین

کرد. از این تابع برای تغییر مقدار رجیسترها کدک استفاده می‌شود.

```
void DSK6713_AIC23_rset(DSK6713_AIC23_CodecHandle hCodec, Uint16 regnum, Uint16 regval);
```

روش استفاده از آن در مثال زیر دیده می‌شود.

```
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;
DSK6713_AIC23_rset(hCodec, 0x0004, inputsource);
```

۴ آزمایش سوم: طراحی و پیاده سازی فیلتر FIR و استفاده از توابع پردازش سیگنال TI

۱-۱ هدف

هدف این آزمایش پیاده سازی فیلترهای FIR^۱ بر روی پردازنده DSP می‌باشد. در ضمن این آزمایش دانشجویان با نحوه استفاده از توابع کتابخانه پردازش سیگنال TI آشنا می‌شوند.

۲-۱ مقدمه

یکی از عملیات مهم در مبحث پردازش سیگنال، فیلترینگ است. دسته مهمی از فیلترهای دیجیتال فیلترهای FIR هستند. این دسته از فیلترها پایدار هستند و فیلترهای با فاز خطی همگی FIR هستند. فیلترهای با فاز خطی، اعوجاج فاز ایجاد نمی‌کنند و فقط سیگنال ورودی را در صورتی که در باند گذر باشد با تأخیر در خروجی می‌فرستند. همچنین این فیلترها در مبحث پردازش سیگنال چند نرخی، زیاد استفاده می‌شوند. از روش‌های طراحی این فیلترها می‌توان به روش پنجره و روش Parks-McClellan اشاره کرد [۱۶].

در این آزمایش در قسمت اول یک فیلتر پایین گذر با مشخصات مورد نظر به کمک ابزار fdatool نرم افزار MATLAB طراحی می‌کنید. سپس این فیلتر را برای پردازش بی‌درنگ بر روی پردازنده DSP بر روی بورد C6713 DSK پیاده می‌نمایید. در قسمت دوم یک فیلتر FIR با طول زیاد را با استفاده از تابع فیلتر بهینه برای پردازنده C67xx پیاده می‌نمایید. در قسمت سوم یک فایل صوتی که با یک سیگنال تون خراب شده است دریافت می‌کنید، با کمک نرم افزار MATLAB در حوزه فرکانس سیگنال تون را تعیین می‌کنید و سپس به صورت دستی یک فیلتر میان نگذر برای حذف سیگنال تون طراحی می‌کنید. سپس این فیلتر را به صورت پردازش بی‌درنگ بر روی بورد پیاده می‌کنید و با حذف تون از سیگنال صوتی به سیگنال فیلتر شده گوش می‌دهید.

۳-۱ تئوری

کانولوشن گستته و پاسخ فرکانسی

^۱ Finite impulse response

خروجی یک سیستم خطی و تغییر ناپذیر با زمان (LTI) گستته به رشتہ ورودی $\{x[n]\}$ از کانولوشن رشتہ ورودی با پاسخ ضربه سیستم $\{h[n]\}$ به دست می‌آید. به این ترتیب $y[n]$ خروجی سیستم در لحظه n از رابطه (1) به دست می‌آید.

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \quad (1)$$

همچنین تبدیل Z کانولوشن دو سیگنال برابر حاصل ضرب دو تبدیل Z سیگنال‌ها می‌باشد، یعنی:

$$\begin{aligned} Y(z) &= \sum_{n=-\infty}^{\infty} y[n]z^{-n} = X(z)H(z) \\ H(z) &= \sum_{n=-\infty}^{\infty} h[n]z^{-n} \quad X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \end{aligned} \quad (2)$$

و که

در ادامه پاسخ سیستم LTI به سیگنال سینوسی نمایی گستته زیر را در نظر بگیرید:

$$x[n] = Ce^{jwnT} \quad (3)$$

بنابراین طبق رابطه (1) خروجی سیستم به این ورودی سینوسی نمایی گستته برابر است با:

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} h[n]Ce^{jw(n-k)T} = Ce^{jwnT} \sum_{k=-\infty}^{\infty} h[n]e^{-jwkT} \\ &= x[n]H(z)|_{z=e^{jwT}} \end{aligned} \quad (4)$$

بنابراین سیگنال خروجی نیز یک سیگنال سینوسی با فرکانس سیگنال سینوسی ورودی می‌باشد که دامنه و فاز آن تغییر کرده است.

به $A(w) = |H(e^{jwT})|$ پاسخ فرکانسی سیستم گویند و $\theta(w) = \arg H(e^{jwT})$ پاسخ فاز نامیده می‌شوند. همان طور که دیده می‌شود پاسخ‌های دامنه و فاز متناوب با دوره تناوب $w_s = \frac{2\pi}{T}$ می‌باشند و به این ترتیب پاسخ فرکانسی در شکل قطبی به صورت $H(e^{jwT}) = A(w)e^{j\theta(w)}$ قابل می‌باشد و خروجی (4) به شکل

اگر سیگنال ورودی به یک سیستم LTI با پاسخ ضربه حقیقی، یک سیگنال سینوسی حقیقی باشد، یعنی:

$$x[n] = C \cos(wnT + \phi) = \operatorname{Re} \left\{ Ce^{j\phi} e^{jwnT} \right\} \quad (5)$$

سیگنال خروجی سیستم برابر است با:

$$y[n] = \operatorname{Re} \left\{ H(e^{jwT}) Ce^{j\phi} e^{jwnT} \right\} = CA(w) \cos(wnT + \theta(w) + \phi) \quad (6)$$

به عبارت دیگر سیستم دامنه سیگنال سینوسی را در پاسخ دامنه به ازای فرکانس ورودی ضرب می کند و فاز آن را بر طبق پاسخ فاز در آن فرکانس تغییر می دهد. این نکته پایه فیلترینگ دیجیتال می باشد.

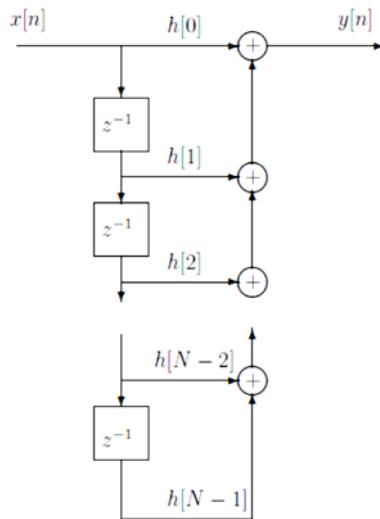
FIR فیلتر

اگر پاسخ ضربه سیستم LTI خارج فاصله $\{0, 1, \dots, N-1\}$ برابر صفر باشد، سیگمای رابطه (1) به یک حاصل جمع متناهی تبدیل می گردد که در رابطه زیر دیده می شود:

$$y[n] = \sum_{k=0}^{N-1} h[k] x[n-k] = \sum_{k=n-N+1}^n x[k] h[n-k] \quad (7)$$

چنین فیلتری یک فیلتر FIR نامیده می شود و منظور از طراحی چنین فیلتری تعیین ضرایب $\{h[n]\}_{n=0}^{N-1}$ می باشد.

بلوک دیاگرام یک نوع متداول پیاده سازی چنین فیلتری در شکل ۱-۴ دیده می شود.



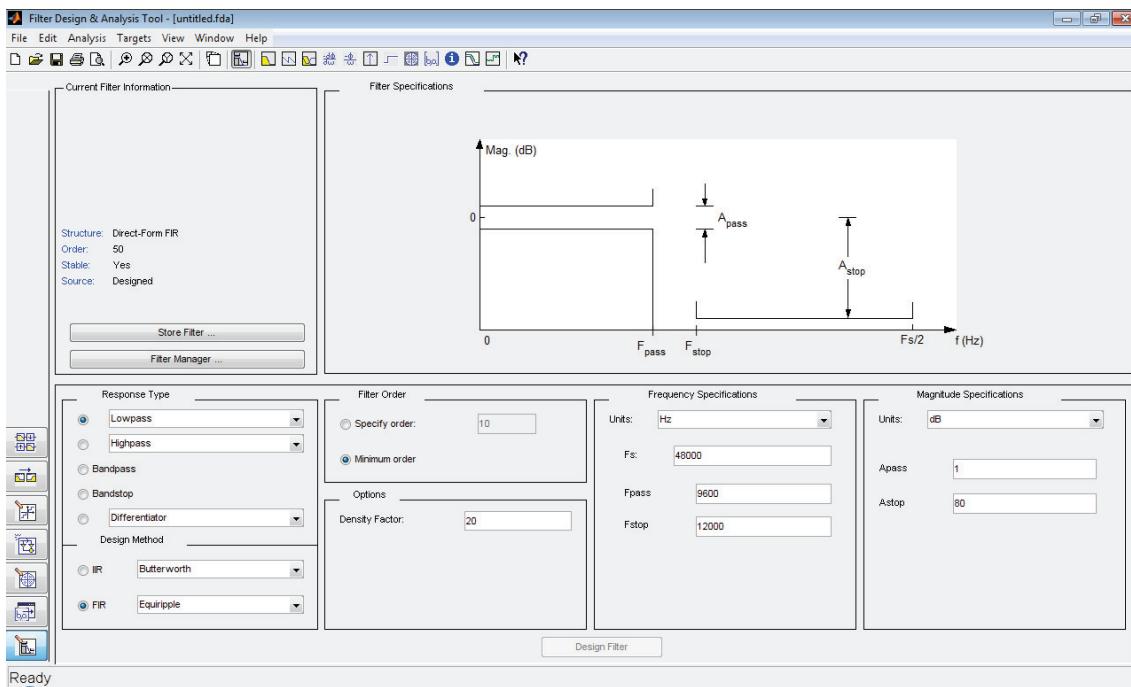
شکل ۱-۴-تحقیق Type I direct form FIR برای فیلتر

در پیاده سازی فیلتر FIR ، خط تأخیر با استفاده از یک آرایه در حافظه پیاده می شود. برای

مدیریت کردن چنین آرایه‌ای ساده‌ترین ایده آن است که جدیدترین مقدار در خانه اول، نمونه قبلی در خانه دوم و نمونه $N-1$ ام قبلی در خانه آخر آرایه قرار داشته باشد که به آن بافر خطی می‌گوییم. با آمدن نمونه جدید، نمونه‌های قدیمی را به خانه‌های مجاور انتقال می‌دهیم (و طبیعتاً قدیمی‌ترین نمونه از بین می‌رود) و نمونه جدید را در اولین خانه آرایه قرار می‌دهیم. راه حل دیگر برای مدیریت آرایه نمونه‌های سیگنال ورودی، ایده آرایه سیرکولار می‌باشد که در این آزمایش با نحوه پیاده‌سازی آن آشنا می‌شویم.

طراحی فیلتر FIR

برای طراحی فیلترهای دیجیتال FIR روش‌های گوناگونی نظری سری فوریه و استفاده از پنجره‌های مختلف، روش Parks & McClellan وجود دارد. همچنین در نرم افزار MATLAB دستورات متنوعی برای طراحی فیلتر با الگوریتم‌های مختلف وجود دارد. یکی از ابزارهای سودمند و با قابلیت^۱ GUI ابزار fdatool می‌باشد. برای استفاده از آن در پنجره فرمان MATLAB دستور fdatool را اجرا کنید. شکل ۲-۴ نمای این ابزار را نشان می‌دهد.



شکل ۲-۴ شمای ابزار fdatool

^۱Graphic user interface

پیاده سازی فیلتر FIR با کد C

فرض کنید ضرایب فیلتر FIR در آرایه $h[N]$ قرار داشته باشند و نمونه سیگنال را در بافر $x[n]$ جمع آوری کنیم. به این ترتیب کد شکل ۳-۴ قسمتی از کد برای پیاده سازی فیلتر FIR با زبان C است و شکل ۴-۴ قسمتی از کد برای پیاده سازی فیلتر FIR با روش سیرکولار بافر در C می‌باشد. همان طور که در شکل ۴-۴ دیده می‌شود، در روش سیرکولار بافر ضرایب فیلتر در یک آرایه عادی قرار دارند. اما در مورد نمونه‌های سیگنال آنها به ترتیب وارد آرایه می‌شوند و وقتی آرایه پر شد نمونه جدید از اول آرایه وارد می‌شود و این روند تکرار می‌گردد. یک اشاره‌گر محل قرار گرفتن جدیدترین نمونه را در آرایه نگه می‌دارد. با این توصیف اگر `newest` به محل قرار گرفتن جدیدترین نمونه در آرایه اشاره داشته باشد، `newest+1` (به پیمانه N) محل قرار گرفتن قدیمی‌ترین نمونه در آرایه می‌باشد. پردازنده‌های TI قابلیت پیاده‌سازی سیرکولار بافر را به صورت سخت افزاری دارند، که البته از طریق کد نویسی اسمبلی در دسترس می‌باشد. به این ترتیب رجیسترها A4-A7 و B4-B7 قابلیت آدرس دهی سیرکولار دارند که وضعیت آدرس دهی آنها در رجیستر^۱ AMR مشخص می‌شود و طول بلوکی که به صورت سیرکولار قابل دسترس قرار می‌گیرد توانی از ۲ است که در رجیستر AMR تعیین می‌شود.

```
#define N 16 //filter length
float h[N]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} // filter
float x[N]; // input buffer
float y;
int i;
float FIR_Filter1(float newsample)
{
    y=0;
    x[0] = newsample;
    for(i=0;i<N;i++) y += h[i]*x[i];
    for(i=N-1;i>0;i--) x[i]=x[i-1];
    return y;
}
```

شکل ۳-۴ قسمتی از کد برای فیلتر FIR بر مبنای تحقیق مستقیم

^۱Address mode register

```

#define N 16 // filter length
float h[N] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}; //filter
float x[N]; // input buffer
float y;
int i, x_index, newest=0;
float FIR_filter2(float newsample)
{
    ++newest;
    if (newest==N) newest=0;
    x[newest] = newsample;

    y = 0;
    x_index=newest;
    for(i=0;i<N;i++)
    {
        y += h[i]*x[x_index];
        -- x_index;
        if (x_index == -1) x_index=N-1;
    }
    return (y);
}

```

شکل ۴-۴ قسمتی از کد برای فیلتر FIR بر مبنای تحقق مستقیم و با پیاده سازی بر مبنای ایده سیرکولار بافر با کد C

۴-۴ فیلتر کردن با استفاده از توابع فیلتر TI

در پردازنده‌های DSP قابلیت‌هایی برای انجام الگوریتم‌های پردازش سیگنال نظری فیلتر کردن و محاسبه تبدیل فوریه سیگنال و ... در نظر گرفته شده است، که لزوماً کامپایلر نرم افزار CCS برای انجام الگوریتم‌های مشخص نوشته شده به زبان C قابلیت استفاده از توانایی‌های پردازنده DSP به صورت بهینه را ندارد. به این منظور شرکت TI کتابخانه‌هایی از توابع پر کاربرد متداول در پردازش سیگنال را برای خانواده‌های مختلف پردازنده‌های خود ارائه کرده است تا در برنامه نویسی کدهای C برای آنها بتوان از این توابع استفاده نمود. در اینجا روش استفاده از این توابع برای پردازنده‌های سری 6000 ذکر می‌گردد و به طور مشخص برای پردازنده‌های خانواده XX67 می‌خواهیم تابع فیلتر عمومی

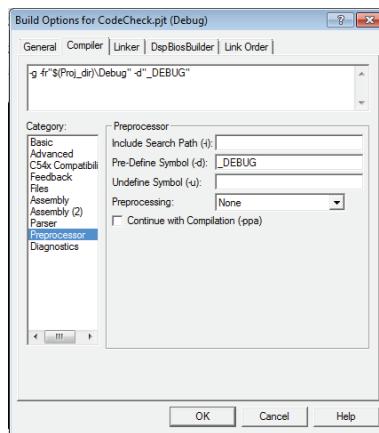
را در کد C به کار بگیریم. برای آگاهی بیشتر از توابع کتابخانه‌ای TI می‌توانید از مراجعی نظیر [۱] TMS320C67x DSP Library Programmer's Reference Guide کمک بگیرید.

مراحل استفاده از توابع ریاضی و پردازش سیگنال TI:

- ۱- در صفحه وب مربوط به پردازنده مورد استفاده (مثالاً 6713) از سایت شرکت TI^۱, فایل مربوط به کتابخانه تابع پردازش سیگنال را دانلود می‌کنیم و نصب می‌نماییم.
- ۲- فایل کتابخانه‌ای مربوطه (dsp67x.lib) را به پروژه اضافه (add) می‌کنیم.
- ۳- تابع پردازش سیگنال مورد استفاده را شناسایی می‌کنیم و آن را در اول برنامه include می‌کنیم.

```
#include "DSPF_sp_fir_gen.h"
```

- ۴- آدرس محل قرار گرفتن این heard file را در قسمت Include Search Path (i): از پنجره Build option از منوی Project وارد می‌کنیم. شکل ۴-۵ پنجره Build option و محل وارد کردن آدرس header file را نشان می‌دهد. در صورتی که لازم باشد آدرس تعداد بیش از یک هدر فایل وارد شود، آنها را با سمیکلون (؛) از هم دیگر جدا کنید.



شکل ۴-۵- محل وارد کردن آدرس فایل‌های header

- ۵- توضیحات مربوط به تابع مورد استفاده را از مرجع مربوطه که توسط TI ارائه شده است مطالعه کرده و تمهدات لازم را به کار می‌گیریم.

به عنوان مثال تابع DSPF_sp_fir_gen() یک فیلتر عمومی با ورودی‌های float می‌باشد. لازم به ذکر است که آدرس متغیرهای ورودی این تابع باید مضربی از 8 باشد (double word). برای این کار می‌توانید از کد زیر در برنامه ایده بگیرید و در برنامه خود استفاده کنید.

¹ www.ti.com

```
#pragma DATA_ALIGN (x,8)
```

۴-۵ آزمایش

۴-۵-۱ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS ، نرم افزار MATLAB ، بورد DSK C6713، اسیلوسکوپ و سیگنال

رنراتور، هدفون

۴-۵-۲ قسمت اول: حذف تون از سیگنال صوتی

در این قسمت یک فایل صوتی دریافت می کنید که با یک سیگنال تن قوى خراب شده است. برای حذف سیگنال تن و اصلاح فایل صوتی یک فیلتر مناسب میان نگذر طراحی خواهد کرد و آن را به کمک روش بافر خطی بر روی پردازنده پیاده خواهد کرد.

۱- فایل صوتی mefsin.wav را به کمک دستور wavread در نرم افزار MATLAB وارد کنید و

فرکانس نمونه برداری آن را تعیین کنید.

۲- به کمک دستور fft طیف سیگنال صوتی را در محدوده F_s رسم کنید (فرکانس نمونه برداری است)

۳- با توجه به طیف سیگنال صوتی فیلتر میان نگذر مناسبی برای حذف تون طراحی کنید. برای طراحی چنین فیلتریتابع انتقال یک فیلتر FIR مرتبه دو را در نظر بگیرید و صفر آن را روی فرکانسی که می خواهید حذف شود قرار دهید ($\frac{F_s}{2} \equiv \pi$). برای آنکه ضرایب فیلتر حقیقی باشند لازم است صفرهای فیلتر مزدوج مختلط باشند. به این ترتیب صفرهای فیلتر $z_1 = e^{j\theta}$ و $z_2 = e^{-j\theta}$ خواهند بود که $\theta \in [0, \pi]$. تابع انتقال چنین فیلتری در زیر داده شده است:

$$\begin{aligned} H(z) &= (1 - z_1 z^{-1})(1 - z_2 z^{-1}) \\ &= (1 - e^{j\theta} z^{-1})(1 - e^{-j\theta} z^{-1}) \\ &= 1 - 2 \cos(\theta) z^{-1} + z^{-2} \end{aligned} \quad (8)$$

ضرایب فیلتر میان نگذر را تعیین کنید.

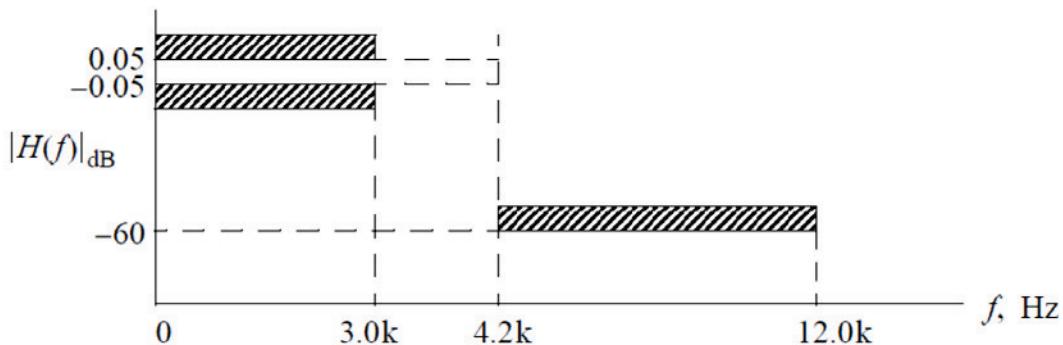
۴- با ضرایب به دست آمده با دستور filter در MATLAB سیگنال صوتی را فیلتر نمایید و طیف سیگنال فیلتر شده را رسم کنید و سیگنال فیلتر شده را با دستور sound پخش نمایید.

۵- پس از اطمینان از صحت فیلترینگ در نرم افزار MATLAB، پروژه‌ای در نرم افزار CCS ایجاد کنید و این فیلتر میان نگذر را پیاده نمایید. پورت Line In بورد را به پورت Line Out کارت صوتی وصل کنید و پورت Line Out بورد را به هدفون وصل کنید. بعد از کامپایل کردن برنامه و اجرای برنامه بر روی پردازنده فایل صوتی را از کامپیوتر پخش کنید و نتیجه سیگنال فیلتر شده را از هدفون بشنوید.

۴-۵-۳- قسمت دوم: طراحی و پیاده سازی فیلتر FIR به کمک بافر سیرکولار

در این قسمت یک فیلتر FIR پایین گذر طراحی و پیاده سازی می‌کنید. برای طراحی فیلتر از ابزار نرم افزار MATLAB استفاده کنید و سپس فیلتر بدست آمده را بر روی پردازنده DSP fdatool پیاده سازی می‌کنید. برای این منظور مراحل زیر را طی کنید.

۱- فرکانس نمونه برداری را 24KHz در نظر بگیرید. به کمک ابزار fdatool فیلتر FIR پایین گذر equiripple با مشخصات فرکانسی مطابق شکل ۶-۴ طراحی کنید.



شکل ۶-۴ مشخصات فیلتر پایین گذر با فرکانس نمونه برداری 24KHz

پاسخ دامنه و پاسخ فاز فیلتر را در نرم افزار MATLAB رسم کنید. محور افقی به فرکانس نمونه برداری مقیاس شده باشد.

۲- برای ذخیره ضرایب فیلتر به دست آمده در MATLAB در یک فایل header و استفاده از آن در کد C راههای گوناگونی دارد که در زیر به دو نمونه اشاره می‌شود:
a. در ابزار fdatool از منوی Targets گزینه ... Generate C header را انتخاب
نمایید و در پنجره باز شده در قسمت Data type to use in export نوع داده مورد

نظر (در اینجا single-precision float) را انتخاب نمایید. سپس فایل header تولید شده را باز کنید و تغییرات لازم را در آن اعمال نمایید و یا ضرایب را از فایل کپی نمایید.

b. با استفاده از توابع fopen و fprintf در MATLAB ضرایب فیلتر را در یک فایل بنویسید.

۳- برنامه‌ای به زبان C بنویسید که با روش سیرکولار بافر عمل فیلترینگ FIR را انجام دهد. داده‌های ورودی را از کانال چپ پورت Line In کدک گرفته و نتیجه را به کانال چپ پورت Line Out بفرستد. در برنامه خود محاسبات را اعشاری در نظر بگیرید، بدین صورت که داده‌های ورودی را به نوع float تبدیل کنید. از وقfe پورت سریال برای خواندن و نوشتند نمونه‌ها استفاده کنید.

۴- در نرم افزار CCS از منوی View گزینه Graph و بعد Time/Frequency را انتخاب کنید. با مشاهده تبدیل فوریه ضرایب فیلتر، پاسخ فرکانسی فیلتر را مشاهده نمایید و با پاسخ فرکانسی به دست آمده از نرم افزار MATLAB مقایسه کنید.

۵- ورودی سیگنال ژنراتور را به پورت Line Out بورد وصل کنید و پورت آن را به اسیلوسکوپ متصل نمایید. برنامه خود را بدون هیچ سطح بهینه سازی کامپایل نمایید و بر روی بورد اجرا کنید.

۶- سیگنال ورودی را یک سیگنال سینوسی انتخاب کنید. با تغییر فرکانس آن در باند گذر^۱ و باند توقف^۲ عملکرد فیلتر خود را بررسی نمایید.

برای جلوگیری از آسیب دیدن بورد DSK توجه داشته باشید دامنه سیگنال ورودی از ۱V RMS بیشتر نشود.

۷- برای مشاهده تخمینی از پاسخ دامنه فیلتر، سیگنال ورودی به فیلتر را نویز سفید انتخاب کنید. به کمک قابلیت نمایش طیف سیگنال در اسیلوسکوپ، طیف سیگنال خروجی فیلتر را مشاهده نمایید.

سؤال: چرا روش فوق تخمینی از پاسخ دامنه فیلتر است؟

۸- از منوی View گزینه Mixed Source/ASM را انتخاب کنید و به نحوه پیاده سازی الگوریتم فیلترینگ توجه کنید. با استفاده از نرم افزار CCS تعداد کلاک لازم برای تولید یک نمونه خروجی فیلتر را تعیین کنید.

¹ Passband

² Stopband

۹- برنامه خود را با سطوح مختلف بهینه سازی کامپایل نمایید و تعداد کلک لازم برای تولید یک نمونه خروجی را تعیین نمایید.

۱۰- طول فیلتر FIR را با اضافه کردن صفر به آن افزایش دهید و ماکریم طول فیلتر که انجام فیلترینگ به صورت زمان حقيقی ممکن است تعیین نمایید.

سؤال: در صورتی که فرکانس نمونه برداری را از 24KHz به 48KHz افزایش دهیم، مشخصات فیلتر پیاده سازی شده چه تغییری می کنند؟

۴-۵-۴ قسمت دوم: استفاده از توابع ریاضی بهینه شده TI

در این قسمت می خواهیم یک فیلتر FIR را به صورت off-line پیاده سازی نماییم. این روش در بحث پردازش بی درنگ سیگنالهای دیجیتال برای پردازش بلوکی سیگنال به کار می رود.

۱- از فیلتر FIR میان گذر equiripple طراحی شده در قسمت قبل برای این آزمایش استفاده می کنیم.

۲- پروژه ای در نرم افزار CCS ایجاد کنید. در این پروژه نیازی به خواندن و نوشتمن از کدک ندارید.

۳- در ادامه با نحوه کار کردن با تابع کتابخانه‌ای `DSPF_sp_fir_gen()` از آشنا `dsplib` می شویم. برای استفاده از این تابع ابتدا مشخصات آن را از مرجع TMS320C67x DSP [۱] مطالعه نمایید. صفحه مربوط به این تابع در بخش ضمایم این فصل آورده شده است. این تابع به صورت بلوکی سیگنال را فیلتر می کند. به طور خلاصه مشخصات این تابع در اینجا آورده شده است.

```
void DSPF_sp_fir_gen (const float *x, const float *h, float
* restrict r, int nh, int nr)
```

`x` و `h` و `r` به ترتیب نام آرایه‌های (اشاره‌گر) داده‌های ورودی، ضرایب فیلتر و داده‌های خروجی فیلتر می باشند. `nh` و `nr` نیز طول آرایه‌های ضرایب فیلتر و خروجی فیلتر هستند. این آرایه‌ها از نوع `float` می باشند. طول آرایه `x` برابر `nh+nr-1` است. لازم است ضرایب فیلتر با ترتیب بر عکس در بردار `h` قرار داشته باشند. طول بردارهای `h` و `r` باید بزرگتر یا مساوی ۴ باشند. این بردارها باید double-word aligned شوند. برای این منظور می توانید از راهنمای زیر مثلاً برای بردار `x` استفاده کنید.

```
#pragma DATA_ALIGN(x, 8)
```

۴- ابتدا برنامه‌ای بنویسید که به صورت off line عمل فیلترینگ را با استفاده از تابع DSPF_sp_fir_gen() انجام بدهد. برای آگاهی از نحوه استفاده از توابع کتابخانه پردازش سیگنال TI به بخش ۴-۴ مراجعه نمایید.

طول آرایه خروجی $nr=2000$ در نظر بگیرید.

۵- برای تست کد فیلتر در C مراحل زیر را انجام دهید:

a. یک بردار از داده‌های float دلخواه برای ورودی فیلتر در نرم افزار MATLAB با طول $nh+nr-1$ تولید کنید. بگذارید ۱ داده اول این بردار صفر باشد. این بردار را با استفاده از توابع fopen و fprintf نرم افزار MATLAB در یک فایل ذخیره نمایید. سپس به صورت دستی خط هدری بر مبنای توضیحات آزمایش اول مبنی بر ورود داده‌ها به CCS با استفاده از قابلیت breakpoint به آن اضافه کنید. البته می‌توانید با استفاده از توابع fopen و fread در C نیز این داده‌ها را در کد C خود بخوانید.

b. بردار ورودی فیلتر را با ضرایب فیلتر در نرم افزار MATLAB با دستور filter فیلتر نمایید.

c. در کد C خود در خط مربوط به اجرای تابع DSPF_sp_fir_gen() یا قبل از آن یک breakpoint قرار دهید و مطابق توضیحات آمده در دستور کار آزمایش اول آن را به فایل داده ورودی فیلتر مرتبط کنید.

d. بعد از کامپایل و اجرای برنامه چک کنید عمل خواندن داده‌ها از فایل و قرار گرفتن در بردار x (داده‌های ورودی فیلتر) به طور صحیح انجام شده باشد.

e. با مقایسه مقادیر بردار r با مقادیر بردار فیلتر شده در نرم افزار MATLAB از صحت برنامه خود اطمینان حاصل کنید.

۶- تعداد کلاکهای لازم برای اجرای این تابع را به کمک نرم افزار CCS تعیین کنید و با مقدار ارائه شده در مرجع [۱] که در رابطه (8) تکرار شده است مقایسه نمایید؟ همچنین با تعداد ضربهای لازم برای این فیلترینگ مقایسه کنید.

$$\left(4 \left\lfloor \frac{nh-1}{2} \right\rfloor + 14 \right) \left\lceil \frac{nr}{4} \right\rceil + 8 \quad (9)$$

۶-۴. ضمایم

۶-۱-۱. برگه اطلاعاتی تابع DSPF_sp_fir_gen

در صفحه بعد تصویر قسمتی از مرجع [۱] که مشخصات مربوط به تابع DSPF_sp_fir_gen می باشد آورده شده است.

۶-۲-۱. لیست توابع کتابخانه پردازش سیگنال TI

جدول ۱-۴ لیست توابع پردازش سیگنال TI برای پردازنده های سری 67x می باشد. برای کسب اطلاع بیشتر از نحوه استفاده از آنها به مرجع [۱] رجوع کنید. برای پردازنده های خانواده های دیگر نیز چنین کتابخانه هایی موجود می باشد.

جدول ۱-۴ لیست توابع کتابخانه TI برای پردازنده های TMS320C67X

Functions				
Adaptive Filtering	DSPF_sp_lms	DSPF_dp_lms		
Correlation	DSPF_sp_autocor	DSPF_dp_autocor		
FFT	DSPF_sp_bitrev_cplx DSPF_sp_fftSPxSP DSPF_dp_bitrev_cplx	DSPF_sp_cfft4_dif DSPF_sp_ifftSPxSP DSPF_dp_cfft4_dif	DSPF_sp_cfft2_dit DSPF_sp_icfft2_dif DSPF_dp_cfft2	DSPF_sp_cfft2_dit DSPF_dp_icfft2
Filtering and convolution	DSPF_sp_fir_cplx DSPF_sp_biquad DSPF_dp_cfft4_dif DSPF_dp_fir_r2 DSPF_dp_iirlat	DSPF_sp_fir_gen DSPF_sp_iir DSPF_dp_cfft2 DSPF_dp_fircirc DSPF_dp_convol	DSPF_sp_fir_r2 DSPF_sp_iirlat DSPF_dp_fir_cplx DSPF_dp_biquad	DSPF_sp_fircirc DSPF_sp_convol DSPF_dp_fir_gen DSPF_dp_iir
Math	DSPF_sp_dotp_sqr DSPF_sp_maxidx DSPF_sp_vecmul DSPF_dp_maxval DSPF_dp_vecsum_sq	DSPF_sp_dotprod DSPF_sp_minval DSPF_dp_dotp_sqr DSPF_dp_maxidx DSPF_dp_w_vec	DSPF_sp_dotp_cplx DSPF_sp_vecrecip DSPF_dp_dotprod DSPF_dp_minval DSPF_dp_vecmul	DSPF_sp_maxval DSPF_sp_vecsum_sq DSPF_dp_dotp_cplx DSPF_dp_vecrecip
Matrix	DSPF_sp_mat_mul DSPF_dp_mat_trans	DSPF_sp_mat_trans DSPF_dp_mat_mul_cplx	DSPF_sp_mat_mul_cplx	DSPF_dp_mat_mul
Misc	DSPF_sp_blk_move DSPF_sp_ftq15	DSPF_sp_blk_ewap16 DSPF_sp_q15tof1	DSPF_sp_blk_ewap32 DSPF_sp_minerror	DSPF_sp_blk_ewap64 DSPF_dp_blk_move

DSPF_sp_fir_gen

DSPF_sp_fir_gen *Single-precision generic FIR filter*

Function void DSPF_sp_fir_gen (const float *x, const float *h, float * restrict r, int nh, int nr)

Arguments

x	Pointer to array holding the input floating-point array.
h	Pointer to array holding the coefficient floating-point array.
r	Pointer to output array
nh	Number of coefficients.
nr	Number of output values.

Description This routine implements a block FIR filter. There are nh filter coefficients, nr output samples, and nh+nr-1 input samples. The coefficients need to be placed in the h array in reverse order {h(nh-1), ..., h(1), h(0)} and the array x starts at x(-nh+1) and ends at x(nr-1). The routine calculates y(0) through y(nr-1) using the following formula:

$$r(n) = h(0)*x(n) + h(1)*x(n-1) + \dots + h(nh-1)*x(n-nh+1)$$

where n = {0, 1, ..., nr-1}.

Algorithm This is the C equivalent for the assembly code. Note that the assembly code is hand optimized and restrictions may apply.

```
void DSPF_sp_fir_gen(const float *x, const float *h,
                      float * restrict r,
                      int nh, int nr)
{
    int i, j;
    float sum;
    for(i=0; i < nr; i++)
    {
        sum = 0;
        for(j=0; j < nh; j++)
        {
            sum += x[i+j] * h[j];
        }
        r[i] = sum;
    }
}
```

DSPF_sp_fir_gen**Special Requirements**

- The x and h arrays are double-word aligned.
- Little endianness is assumed for LDDW instructions.
- The number of coefficients must be greater than or equal to 4.
- The number of outputs must be greater than or equal to 4

Implementation Notes

- LDDW instructions are used to load two SP floating-point values simultaneously for the x and h arrays.
- The outer loop is unrolled four times.
- The inner loop is unrolled two times and software pipelined.
- The variables prod1, prod3, prod5, and prod7 share A9. The variables prod0, prod2, prod4, and prod6 share B6. The variables sum1, sum3, sum5, and sum7 share A7. The variables sum0, sum2, sum4, and sum6 share B7. This multiple assignment is possible since the variables are always read just once on the first cycle that they are available.
- The first eight cycles of the inner loop prolog are conditionally scheduled in parallel with the outer loop. This increases the code size by 14 words, but improves the cycle time.
- A load counter is used so that an epilog is not needed. No extraneous loads are performed.
- Endianness:** This code is little endian.
- Interruptibility:** This code is *intended* to be interrupt-tolerant but not interruptible. However, a bug in the assembly code for Rev 2.0 and earlier of the library causes this function to *not* be interrupt tolerant. Therefore, in order to safely use this function you must disable interrupts prior to the call and then restore interrupts after.

Benchmarks

Cycles	$(4*\text{floor}((\text{nh}-1)/2)+14)*(\text{ceil}(\text{nr}/4)) + 8$ e.g., nh=10, nr=100, cycles=758 cycles
Code size (in bytes)	640

۵ آزمایش چهارم: طراحی و پیاده سازی فیلترهای IIR

۱-۵ هدف

در این آزمایش ضمن یاد آوری چند ساختار پیاده سازی فیلتر IIR پیاده سازی یک فیلتر مرتبه ۶ بر روی بورد DSP انجام می شود. همچنین یک سیستم تولید اکو و یک مولد سیگنال سینوسی پیاده سازی می گردد.

۲-۵ مقدمه

دسته ای از فیلترهای دیجیتال، فیلترهای IIR هستند. ساختار این فیلترها دارای فیدبک است و بنابراین ممکن است ناپایدار شوند. معمولاً با پاسخ فرکانسی یکسان فیلتر IIR از مرتبه پایین تر نسبت به فیلتر FIR طراحی می شود. پیاده سازی fixed-point این فیلترها نسبت به فیلترهای FIR دشوار تر است. عموماً طراحی آنها بر مبنای فیلترهای آنالوگ است [۱۶]. برای پیاده سازی این نوع فیلترها ساختارهای پیاده سازی متنوعی ارائه شده است که وقتی با پیاده سازی fixed-point روبرو هستیم اهمیت این ساختارها نمایان می شود.

در این آزمایش در قسمت اول یک فیلتر IIR مرتبه شش به کمک ابزار fdatool در نرم افزار MATLAB طراحی می کنید و با ساختار second order section, direct form II پیاده می کنید. در قسمت دوم با پیاده سازی یک فیلتر IIR مرتبه اول، یک سیستم تولید اکو می سازید. همچنین در قسمت سوم یک مولد سیگنال سینوسی بر روی بورد پیاده می نمایید.

۳-۵ تئوری

فیلتر گستته ای که پاسخ ضربه آن تعداد نامتناهی مقدار دارد، تحت عنوان فیلتر^۱ IIR شناخته می شود. رابطه ورودی-خروجی سیستمی که به صورت زیر است در نظر بگیرید:

$$y[n] = \sum_{k=0}^M b_k x[n-k] - \sum_{k=1}^N a_k y[n-k] \quad (10)$$

یا به طور معادل:

¹ Infinite impulse response

$$y[n] = b_0x[n] + b_1x[n-1] + \dots + b_Mx[n-M] - a_1y[n-1] - a_2y[n-2] - \dots - a_Ny[n-N] \quad (11)$$

این رابطه بازگشتی یک فیلتر IIR را نشان می‌دهد. این رابطه نشان می‌دهد خروجی به ورودی در لحظه‌های حال و گذشته و ورودی‌های پیشین بستگی دارد. اگر برای این سیستم حالت استراحت اولیه^۱ در نظر بگیریم یعنی با فرض آنکه قبل از اعمال ورودی کلیه شرایط اولیه آن صفر باشد، تبدیل z وتابع انتقال چنین سیستمی به صورت روابط (12) و (13) می‌باشند. و ناحیه همگرایی آن ناحیه خارج بزرگترین قطب آن می‌گردد.

$$Y(z) = (b_0 + b_1z^{-1} + \dots + b_Mz^{-M})X(z) - (a_1z^{-1} + \dots + a_Nz^{-N})Y(z) \quad (12)$$

$$H(z) = \frac{b_0 + b_1z^{-1} + \dots + b_Nz^{-N}}{1 + a_1z^{-1} + \dots + a_Mz^{-M}} \quad (13)$$

بدون کم شده از کلیت فرض کنید $b_M = N$ (ضرایب M می‌توانند صفر باشند). رابطه (14) تابع انتقال را بر حسب N صفر و N قطب آن نشان می‌دهد.

$$H(z) = \frac{b_0z^N + b_1z^{N-1} + \dots + b_N}{a_0z^N + a_1z^{N-1} + \dots + a_N} = C \prod_{i=1}^N \frac{z - z_i}{z - p_i} \quad (14)$$

برای آنکه چنین سیستم علی پایدار باشد لازم و کافی است تمام قطبهای آن در صفحه مختلط داخل دایره واحد قرار داشته باشند.

فیلتر IIR فاز غیر خطی دارد اما معمولاً یک پاسخ فرکانسی مطلوب را با مرتبه پائین تری نسبت به فیلتر FIR محقق می‌کند.

۳-۵-۱. تحقیق فیلتر IIR

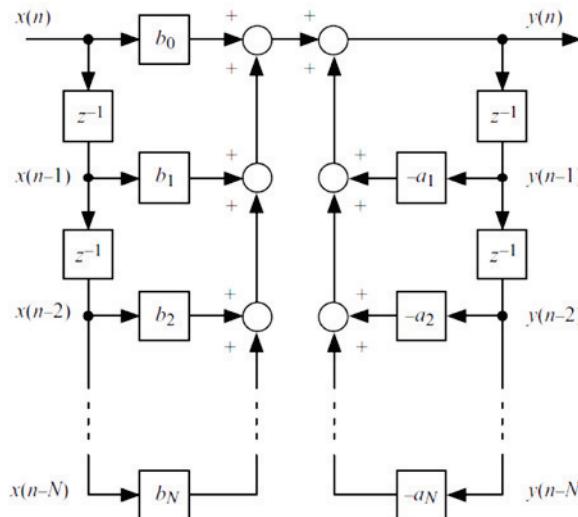
تابع انتقال کسری به صورتهای گوناگون قابل تحقیق است که چند نوع متداول در ادامه توضیح داده می‌شوند.

۱. تحقیق مستقیم نوع ۱

¹ Initially rest condition

² Direct form I

شکل ۱-۵ ساختار تحقق مستقیم نوع ۱ را برای رابطه (۱۰) نشان می‌دهد. همان طور که دیده می‌شود این ساختار $2N$ المان تأخیر نیاز دارد یعنی برای برنامه نویسی به $2N$ خانه حافظه نیاز است.



شکل ۱-۵ ساختار تحقق مستقیم نوع ۱ برای فیلتر IIR

۲. تحقق مستقیم نوع ۲

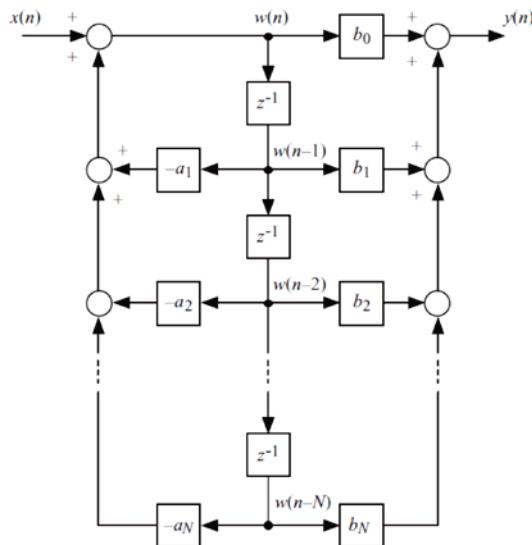
ساختار متداولی که معمولاً مورد استفاده قرار می‌گیرد تحقق مستقیم نوع ۲ می‌باشد که در شکل ۲-۵ دیده می‌شود. در این تحقق به N تأخیر نیاز است. برای اثبات درستی این تحقق، از خاصیت جابه‌جاپذیر بودن دو سیستم خطی استفاده کنید و دو ستون تحقق مستقیم نوع ۱ را در شکل ۱-۵ عوض کنید. برای پیاده سازی این تحقق، یک آرایه N تایی با نام w در نظر بگیرید که مقادیر آن با توجه به رابطه (۱۵) به هنگام می‌شوند.

$$w[n] = x[n] - a_1 w[n-1] - \dots - a_N w[n-N] \quad (15)$$

مقدار اولیه بردار w برابر صفر است. خروجی فیلتر بر حسب ورودی با توجه به شکل ۲-۵ برابر است با:

$$y[n] = b_0 w[n] + b_1 w[n-1] + \dots + b_N w[n-N] \quad (16)$$

^۱ Direct form II



شکل ۵-۲ ساختار تحقق مستقیم نوع ۲ برای فیلتر IIR

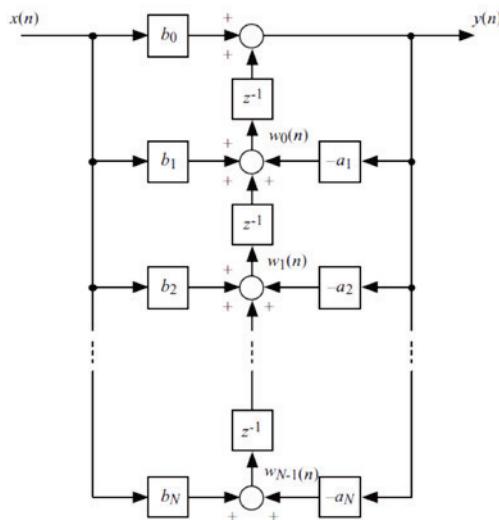
۳. تحقق مستقیم نوع ۲ ترانسپوز^۱

چنین تحقیقی در شکل ۳-۵ دیده می شود. در این تحقیق تعداد خطهای تأخیر به اندازه تحقق مستقیم نوع ۲ می باشد. در اینجا نیز یک آرایه $w[n]$ در نظر بگیرید که مقادیر این آرایه در لحظه n بر حسب مقادیر آرایه در لحظه $n-1$ طبق روابط بعد از محاسبه $y[n]$ طبق رابطه (17) به هنگام می شوند.

$$y[n] = b_0 x[n] + w_0[n-1] \quad (17)$$

$$\begin{aligned} w_0[n] &= b_1 x[n] + w_1[n-1] - a_1 y[n] \\ w_1[n] &= b_2 x[n] + w_2[n-1] - a_2 y[n] \\ &\vdots \\ w_{N-1}[n] &= b_N x[n] - a_N y[n] \end{aligned} \quad (18)$$

¹ Direct form II transpose



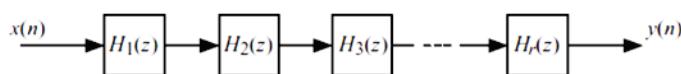
شکل ۳-۵ ساختار تحقق مستقیم نوع ۲ ترانسپوز برای فیلتر IIR

۴. تحقق سری^۱

رابطه (13) را می‌توان به صورت حاصل ضرب توابع انتقال مرتبه ۱ و/یا مرتبه ۲ تجزیه کرد، یعنی:

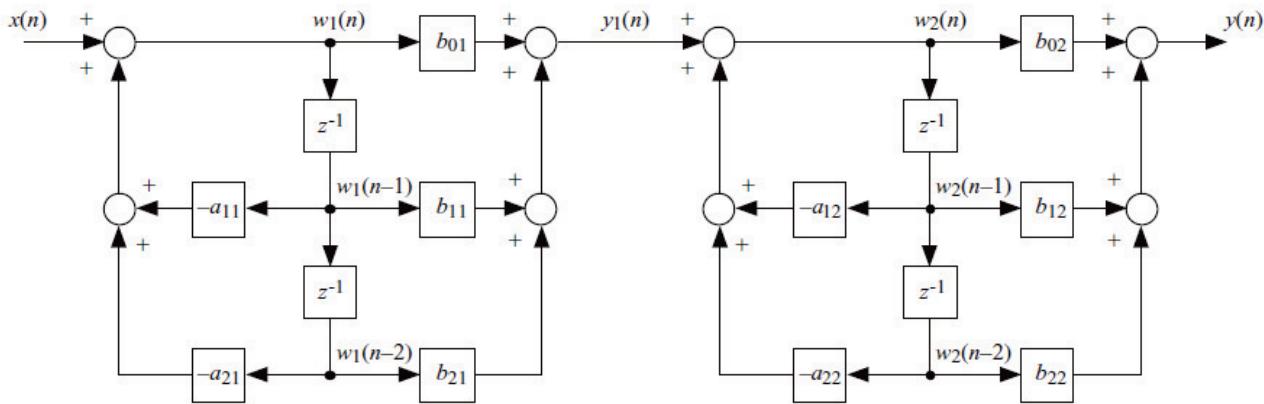
$$H(z) = CH_1(z)H_2(z)\cdots H_r(z) \quad (19)$$

که هر کدام از این بخشها را می‌توان به صورت تحقق مستقیم نوع ۲ پیاده نمود. چنین تحقیقی به صورت شکل ۴-۵ نشان داده شده است. در حقیقت در رابطه (13) قطبهای سیستم به تمامی ضرایب مخرج تابع انتقال بستگی دارند و با تغییر ضرایب مثلاً در اثر کوانتیزاسیون ضرایب، محل قطبها و در نتیجه پاسخ فرکانسی فیلتر عوض می‌شود. در صورتی که وقتی یک فیلتر مرتبه بالاتر را به تعدادی فیلتر حداقل از مرتبه ۲ تبدیل کنیم، این مسئله بهبود می‌یابد. شکل ۵-۵ تحقق یک فیلتر مرتبه چهار با متوالی کردن دو فیلتر مرتبه دو نشان می‌دهد.



شکل ۵-۵ ساختار تحقق سری فیلتر IIR

¹ Cascade



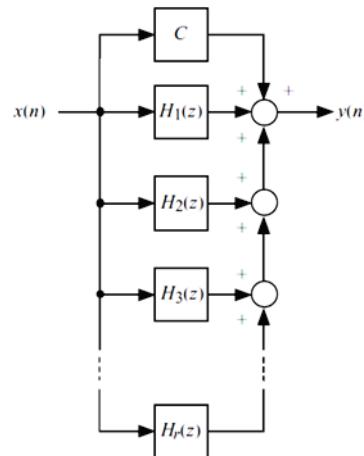
شکل ۵-۵ تحقق فیلتر IIR مرتبه چهار با دو بخش با تحقق مستقیم نوع ۲

۵. تحقق موازی^۱

تابع انتقال رابطه (۱۳) را می‌توان به صورت مجموع کسرهای جزئی بسط داد، یعنی به صورت حاصل جمع تعدادی تابع انتقال حداکثر از مرتبه ۲:

$$H(z) = C + H_1(z) + H_2(z) + \dots + H_r(z) \quad (20)$$

شکل ۶-۵ ساختار پیاده سازی موازی را نشان می‌دهد.



شکل ۶-۵ ساختار تحقق موازی فیلتر IIR

به این ترتیب برای پیاده سازی یک تابع انتقال کسری یک تحقق را انتخاب می‌کنیم و بر اساس آن تابع انتقال را پیاده می‌کنیم. کد شکل ۷-۵ قسمتی از یک برنامه C برای پیاده سازی یک فیلتر FIR با تحقق مستقیم نوع ۲ می‌باشد.

¹ Parallel

```

for (section=0 ; section< NUM_SECTIONS ; section++)
{
    Wn=input-a[section][0]*w[section][0]-a[section][1]*w[section][1];
    yn=b[section][0]*wn+b[section][1]*w[section][0] +
        b[section][2]*w[section][1];
    w[section][1] = w[section][0];
    w[section][0] = wn;
    input = yn; // output of current section will be input to next
}

```

شکل ۷-۵ قسمتی از کد C برای پیاده سازی فیلتر IIR بر مبنای تحقق سری که هر بخش بر مبنای تحقق مستقیم نوع ۲ پیاده سازی شده است

۵-۴ آزمایش

۵-۴-۱ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS ، نرم افزار MATLAB ، بورد DSK C6713 ، اسیلوسکوپ و سیگنال ژنراتور، میکروفون، هدفون

۵-۴-۲ قسمت اول: پیاده سازی فیلتر IIR

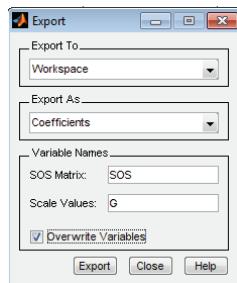
در این قسمت یک فیلتر IIR مرتبه ۶ طراحی می‌کنید و آن را بر روی پردازنده DSP پیاده می‌کنید. به این منظور برای انجام این قسمت مراحل زیر را انجام دهید.

۱- فرکانس نمونه برداری را 8000Hz در نظر بگیرید. به کمک ابزار fdatool فیلتر میان نگذر IIR مرتبه ۶ بیضوی^۱ با فرکانس مرکزی 1800Hz و فرکانس قطع پایین 1200Hz و فرکانس قطع بالای 2400Hz و Astop=40dB و Apass=1dB طراحی کنید.

۲- این فیلتر را به صورت سری کردن سه فیلتر مرتبه ۲ پیاده سازی می‌کنیم. ابزار fdatool ضرایب این فیلترهای مرتبه ۲ را تعیین می‌کند. به این منظور از منوی Edit گزینه Convert to Second-Order را انتخاب نمایید. همچنین برای انتخاب ساختار پیاده سازی هر بخش از منوی Edit گزینه sections را انتخاب نمایید. Direct-Form II, SOS convert Structure... را انتخاب کنید و سپس از بین ساختارهای موجود برگزینید. بعد از طراحی فیلتر از منوی File گزینه Export را انتخاب نمایید (شکل ۸-۵). ماتریس

^۱ Elliptic

SOS ضرایب فیلتر را نشان می دهد که به تعداد بخشهای فیلتر سطر دارد و در هر سطر سه ستون اول ضرایب صورت تابع انتقال و سه ستون دوم ضرایب مخرج تابع انتقال می باشند. همچنین بردار G ضرایب هر بخش است



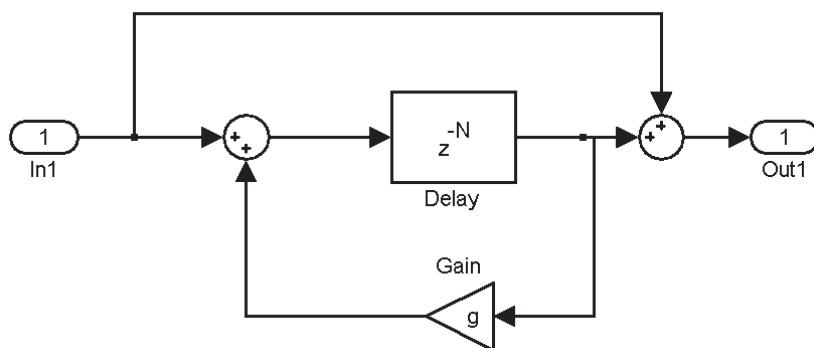
شکل ۸-۵ پنجره Export ابزار fdatool

- ۳- پاسخ دامنه فیلتر را به کمک نرم افزار MATLAB رسم کنید.
- ۴- پروژه ای در نرم افزار CCS ایجاد کنید. فرکانس نمونه برداری را 8000Hz و ورودی بورد را پورت Line In انتخاب نمایید و خروجی را نیز از پورت Line Out بگیرید. برنامه ای C بنویسید که فیلتر IIR را با تحقق سری و هر بخش تحقق مستقیم نوع ۲ پیاده کند. ضرایب فیلتر IIR مرتبه ۶ که در قسمت قبل طراحی نموده اید در کد C خود وارد کنید. عملیات فیلتر کردن در تابع سرویس روتین وقفه انجام شود و داده را از کانال چپ پورت Line In بورد دریافت کند.
- ۵- برای رسم پاسخ فرکانسی بورد به کمک سیگنال ژنراتور به ورودی بورد سیگنال نویز سفید بدھید و طیف سیگنال فیلتر شده را روی اسیلوسکوپ مشاهده نمایید که تخمینی از پاسخ فرکانسی فیلتر می باشد.

۴-۴-۳- قسمت دوم: سیستم تولید پژواک

شکل ۹-۵ بلوک دیاگرام یک سیستم پژواک را نشان می دهد. همان طور که در این بلوک دیاگرام دیده می شود در این سیستم پژواک کسری از سیگنال خروجی به ورودی داده می شود و به این ترتیب اثر محوشوندگی^۱ در سیگنال خروجی ایجاد می شود.

¹ Fading



شکل ۹-۵ بلوک دیاگرام سیستم تولید اکو

۱- تابع انتقال سیستم اکو شکل ۹-۵ را به دست آورید. حداقل مقدار g برای پایداری سیستم چقدر است؟

در صورتی که $g=0$ باشد، این سیستم حاصل جمع سیگنال ورودی را با تأخیر یافته خود در خروجی ایجاد می کند.

۲- پروژه ای در CCS برای پیاده سازی سیستم اکو شکل ۹-۵ ایجاد کنید. فرکانس نمونه برداری را ۱۶۰۰۰Hz در نظر بگیرید و سیگنال ورودی از پورت MIC خوانده شود. مقدار پارامتر gain را برابر ۰.۵ و تأخیر $N=1000$ در نظر بگیرید. به صدای اکو دار تولید شده گوش کنید.

۳- به کمک gel فایل میزان بهره (gain) و تأخیر را حین اجرای برنامه تغییر دهید.
با افزایش بهره ناپایداری سیستم اکو را مشاهده کنید.

۴-۴. قسمت سوم: مولد سیگنال سینوسی (اختیاری)

تولید سیگنال سینوسی با فیلتر IIR:

۱- نشان دهید تبدیل z سیگنال $y[n] = \sin(nwT)$ است.

$$Y(z) = \frac{z^{-1} \sin(wT)}{1 - 2 \cos(wT) + z^{-2}}$$

۲- تابع انتقال یک سیستم IIR مرتبه ۲ به صورت $H(z) = \frac{Y(z)}{X(z)} = \frac{b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}}$ را در نظر بگیرید.
با انتخاب پارامترهای آن به صورت $b_1 = \sin(wT)$ و $a_2 = -2 \cos(wT)$ و $a_1 = 1$ پیاده سازی این سیستم

و اعمال ورودی ضربه به آن، انتظار داریم یک سیگنال سینوسی در خروجی داشته باشیم. این سیستم را در نرم افزار MATLAB شبیه سازی کنید. (راهنمایی: از دستور filter و سیگنال ورودی $[1, \text{ones}(1, N)]$ استفاده کنید).

۳- با پیاده سازی این سیستم IIR روی بورد پردازنده یک سیگنال ژنراتور سینوسی تولید کنید.

سؤال: حداکثر فرکانس قابل تولید این سیستم چقدر است؟

۶ آزمایش پنجم: تبدیل فوریه (FFT) و تخمین طیف

۱-۶ هدف

در این آزمایش ضمن آشنایی بیشتر با FFT ، با استفاده از بورد DSK C6713 یک اسپکتروم آنالایزر ساخته می شود.

۲-۶ مقدمه

یکی از کاراترین تبدیلها در مبحث پردازش سیگنال تبدیل فوریه می باشد. این تبدیل سیگنال را از حوزه زمان به حوزه فرکانس منتقل می کند. اسپکتروم آنالایزر ابزاری برای نمایش سیگنال در حوزه فرکانس می باشد. در این آزمایش با استفاده از تبدیل فوریه گسسته (FFT) و متوسط گیری از تناوبنگار تخمینی از طیف توان به دست آوریم و به این ترتیب یک اسپکتروم آنالایزر دیجیتال می سازیم. در ضمن با چالشهای برنامه نویسی پردازش بی درنگ سیگنال و رفع آنها نیز آشنا می شویم.

۳-۶ تئوری

تبدیل فوریه گسسته زمان^۱ (DTFT)

فرض کنید سیگنال آنالوگ $(t)x$ با دوره تناوب T یا فرکانس زاویه‌ای نمونه برداری $w_s = \frac{2\pi}{T}$ نمونه برداری شود تا سیگنال گسسته زمان $(nT)x[n] = x(nT)$ به دست آید. تبدیل فوریه گسسته زمان برای سیگنال $x[n]$ در صورت وجود طبق رابطه (21) تعریف می شود.

$$X(w) = \sum_{n=-\infty}^{\infty} x[n] e^{-jwnT} \quad (21)$$

توجه کنید که $X(w)$ متناوب با دوره تناوب w_s می باشد. سیگنال $x[n]$ از روی $X(w)$ با رابطه تبدیل فوریه معکوس طبق رابطه (22) به دست می آید.

¹ Discrete time Fourier transform

$$x[n] = \frac{1}{w_s} \int_{\frac{-w_s}{2}}^{\frac{w_s}{2}} X(w) e^{jwnT} dw \quad (22)$$

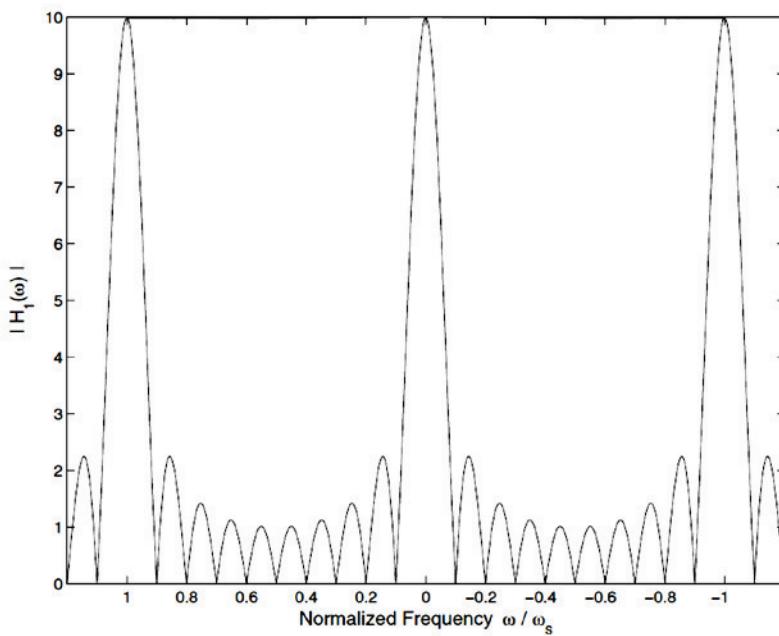
در عمل برای محاسبه تبدیل فوریه گستته زمان، سیگنال باید از نظر زمانی محدود باشد، مثلاً فقط برای مقادیر $0 \leq n \leq N-1$ مقادیر $x[n]$ غیر صفر باشد یا از مقادیر خارج این فاصله صرف نظر شود. که این معادل آن است که از روی سیگنال $x[n]$ ، سیگنال دیگری که از ضرب پنجره مستطیلی با ضرایب $h_1[n]$ در سیگنال $x[n]$ به دست می‌آید، بسازیم.

$$h_1[n] = \begin{cases} 1 & n = 0, 1, \dots, N-1 \\ 0 & elsewhere \end{cases} \quad (23)$$

اثر پنجره کردن سیگنال بر روی طیف سیگنال $x[n]$ با استفاده از خواص تبدیل فوریه روشن می‌گردد. اگر $H(w) = x[n]h[n]$ باشد، طیف $y[n]$ طبق رابطه (24) با طیف سیگنال $x[n]$ و طیف پنجره (w) ارتباط دارد.

$$Y(w) = \frac{1}{w_s} \int_{\frac{-w_s}{2}}^{\frac{w_s}{2}} X(\lambda) H(w - \lambda) dw \quad (24)$$

يعنى از کانولوشن طیف سیگنال اولیه و طیف پنجره، طیف سیگنال پنجره به دست می‌آید. شکل ۱-۶ طیف پنجره مستطیلی را نشان می‌دهد.

شکل ۱-۶ اندازه تبدیل فوریه پنجره مستطیلی به طول $N=10$

به طور ایده آل انتظار داریم طیف پنجره به تابع ضربه نزدیک باشد، یعنی عرض بیم اصلی کم و ارتفاع لوبهای کناری کوچک باشد. پنجره های متنوعی ارائه شده‌اند که سعی در مصالحه بین عرض لوب اصلی و ارتفاع لوبهای کناری به ازای پنجره با طول محدود، هستند. از جمله این پنجره‌ها می‌توان به پنجره های مستطیلی، لوبهای کناری به ازای پنجره کرد. رابطه (25) تعریف پنجره همینگ را نشان می‌دهد.

$$h[n] = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) & n = 0, 1, \dots, N-1 \\ 0 & elsewhere \end{cases} \quad (25)$$

در این پنجره لوبهای فرعی حداقل 40dB کمتر از لوب اصلی هستند.

تبدیل فوریه گسسته (DFT^1)

فرض کنید $[x[n]]$ سیگنالی باشد که به ازای اندیشهای خارج از بازه $0 \leq n \leq N-1$ صفر باشد. در این صورت تبدیل فوریه گسسته این سیگنال یک سیگنال N نقطه‌ای است که طبق رابطه (26) تعریف می‌گردد.

¹ Discrete Fourier transform

$$X[k] = X\left(k \frac{w_s}{N}\right) = \sum_{n=0}^{N-1} x[n] e^{-j \frac{2\pi}{N} kn} \quad (26)$$

در حقیقت DFT سیگنال $x[n]$ برابر N می‌باشد که در فرکانس‌های زاویه‌ای با اختلاف $\frac{w_s}{N}$ برداشته شده‌اند. همچنین تبدیل فوریه معکوس N نقطه‌ای طبق رابطه (27) به دست می‌آید.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j \frac{2\pi}{N} nk} \quad k=0,1,\dots,N-1 \quad (27)$$

FFT¹

در صورتی که طول سیگنال گستته N توانی از ۲ باشد الگوریتمهای سریعی برای محاسبه DFT وجود دارند که به FFT مشهور هستند. باز محاسباتی محاسبه DFT سیگنال N نقطه‌ای به روش مستقیم (رابطه (6)) از مرتبه N^2 می‌باشد، در حالی که برای الگوریتمهای FFT از مرتبه $N \log_2(N)$ می‌باشند.

تخمین طیف توان با استفاده از FFT

یک روش برای محاسبه طیف توان استفاده از تابعی به نام تناوبنگار² می‌باشد. تناوبنگار برای دنباله N نقطه‌ای $y[n]$ طبق رابطه (28) تعریف می‌شود.

$$I_N(w) = \frac{1}{N} |Y(w)|^2 \quad (28)$$

$$Y(w) = \sum_{n=0}^{N-1} y[n] e^{-jwnT} \quad (29)$$

برای تخمین طیف متوسط تناوبنگار را بر روی L فریم حساب می‌کنیم و هر فریم شامل N نمونه می‌باشد. بنابراین اگر $x[n]$ دنباله مشاهده شده به طول $M = NL$ باشد، L فریم داده پنجره شده داریم که داده‌های پنجره شده k امین فریم طبق رابطه (30) فرمول بندی می‌شوند.

¹ Fast Fourier transform

² Periodogram

$$y_k[n] = \begin{cases} h[n]x[n+kN] & n = 0, 1, \dots, N-1 \\ 0 & elsewhere \end{cases} \quad (30)$$

که $h[n]$ پنجره دلخواه است و $I_{N,k}(w)$ نیز تناوبنگار فریم k ام می‌باشد. به این ترتیب یک تخمین‌گر طیف توان عبارت است از

$$\hat{S}(w) = \frac{1}{L} \sum_{k=0}^{L-1} I_{N,k}(w) \quad (31)$$

البته در صورتی که فریم‌های داده همپوشانی داشته باشند، تخمین بهتری خواهیم داشت.

۶-۴ آزمایش

۶-۴-۱ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS، نرم افزار MATLAB، بورد DSK C6713، اسیلوسکوپ و سیگنال ژنراتور

۶-۴-۲ قسمت اول: FFT

در اجرای این بخش می‌توانید از تابع `fft.c` و یا توابع کتابخانه پردازش سیگنال TI استفاده کنید.

۱- با فرض نرخ نمونه برداری 16KHz و طول فریم $N = 1024$ به طور تئوری تبدیل فوریه گستته سیگنال $\{x[n]\}$ را حساب کنید.

$$x[n] = \sin\left(2\pi \times 2000 \times \frac{n}{16000}\right) \quad n = 0, \dots, 1023$$

۲- پروژه ای در CCS برای اجرا بر روی C6713 DSK آماده کنید که FFT سیگنال $\{x[n]\}$ را حساب کند. برای ایجاد این پروژه می‌توانید مراحل زیر را طی کنید:

a. طول استک را در قسمت Build Options به مقدار 0x1000 افزایش دهید.

b. آرایه مختلط N تایی را با مقادیر حقیقی و موهومی سیگنال $\{x[n]\}$ پر کنید. با توجه به آنکه تبدیل DFT برای سیگنالهای مختلط تعریف می‌شود و نتیجه آن نیز مختلط می‌باشد تابع `fft()` یک آرایه از نوع `complex` می‌گیرد. نوع داده `complex` در فایل سرآیند `complex.h` تعریف

- شده است. که لازم است در ابتدای کد C خود آن را include نمایید.تابع () fft تبدیل فوریه آرایه ورودی را بر روی خودش می‌نویسد (In place computation) .
- با فراخوانی تابع () fft تبدیل فوریه سیگنال $\{x[n]\}$ را حساب کنید.
 - آرایه حقیقی N تایی را با مقادیر اندازه تبدیل فوریه سیگنال $\{x[n]\}$ پر کنید.
 - با مشاهده مقادیر FFT محاسبه شده در پنجره watch window و مقایسه با مقادیر ثئوری برنامه خود را تست کنید.
 - اندازه تبدیل فوریه سیگنال $\{x[n]\}$ را در نرم افزار CCS به کمک قابلیت نمایش گرافیکی داده‌ها نمایش دهید.
 - تعداد کلاک لازم برای تابع () fft مورد استفاده را تعیین کنید.

۶-۴-۳-قسمت دوم: اسپکتروم آنالایزر

۶-۴-۳-۱-ساختار اسپکترم آنالایزر

در این قسمت یک طیف سنج مقدماتی می‌سازیم. بورد DSK سیگنال آنالوگ را با نرخ 16000Hz نمونه برداری می‌کند و در فریمهای $N=1024$ قرار می‌دهد و سپس DSP با محاسبه و میانگین گیری از تناوبنگار تخمینی از طیف سیگنال می‌دهد که در نرم افزار CCS و با استفاده از قابلیت نمایش گرافیکی و animate آن نمایش داده می‌شود.

در اینجا از روش (11) برای تخمین طیف استفاده می‌شود. مراحل زیر را دنبال کنید:

- DSK را initialize نمایید و کارهای لازم را برای استفاده از توابع خواندن از کدک و نوشتمن در کدک انجام دهید.
- آرایه ای float با طول $N=513$ به نام spectrum برای تخمین طیف در بازه فرکانسی 0-8000Hz (k=0,...,512) تعریف کنید. (لازم به یادآوری است که طیف سیگنال حقیقی تقارن دارد، بنابراین مقادیر طیف برای فرکانس های 8000-16000Hz قرینه مقادیر برای 0-8000Hz می‌باشند).

۳- دو بافر (آرایه) به نامهای ping و pong با طول 1024 و نوع داده مختلط (این داده در هدر فایل complex.h تعریف شده است). تعریف کنید و یکی از آنها نمونه های سیگنال ورودی را از ADC با استفاده از پورت سریال دریافت می کند و برای دیگری محاسبه fft و جمع برای متوسط گیری تناوبنگار انجام می شود. نمونه های سیگنال درتابع ISR خوانده می شوند و به نوع float تبدیل می شوند و در آرایه ping یا pong ذخیره می گردند. 1024 نمونه اول در قسمت حقیقی آرایه ذخیره شوند و 1024 نمونه بعدی در قسمت موهمی آن.

يعنى اگر $a[n]$ دنباله حقیقی ۱۰۲۴ تایی اول و $b[n]$ دنباله حقیقی ۱۰۲۴ تایی دوم باشد در آرایه ping یا pong یک آرایه مختلط $c[n] = a[n] + jb[n]$ خواهیم داشت.

۴- وقتی آرایه با نمونه های سیگنال پر شد:

a. تبدیل فوریه 1024 نقطه ای را بر روی آرایه مختلط اعمال کنید.

b. اگر $a[n]$ و $b[n]$ دو دنباله حقیقی با طول N باشند و به ترتیب A_k و B_k ضرایب DFT آنها باشند و $c[n] = a[n] + jb[n]$ دنباله مختلطی با ضرایب C_k باشد که از

$$|A_k|^2 + |B_k|^2 = \frac{|C_k|^2 + |C_{N-k}|^2}{2}$$

روی آن دو ساخته شده است آنگاه بین این ضرایب رابطه برقرار است. با توجه به این رابطه، مجموع مربعات ضرایب FFT را به آرایه spectrum اضافه کنید. همچنین با توجه به آنکه سیگنال ورودی حقیقی است، کافی است برای $k = 0, 1, \dots, 512$ آرایه spectrum را پر کنیم به این ترتیب مقدار تناوبنگار در فرکانسهاي $\{kw_s / N; k = 0, 1, 2, \dots, N/2\}$ تخمین زده می شود.

سؤال: رابطه اخیر را اثبات کنید.

c. بعد از آنکه محاسبه fft و جمع زدن با آرایه spectrum تمام شد تا پر شدن آرایه بعدی صبر کنید. باید به طریقی از پر شدن آرایه مطلع شوید.

d. به ازای $L=8$ فریم که به آرایه spectrum اضافه شد، آرایه spectrum را برای نرمالیزاسیون بر 1024×8 تقسیم کنید. (ممکن است در عمل عدد L را مقداری غیر از 8 انتخاب کنید).

e. بعد از تخمین طیف با متوسط گیری به ازای L فریم، و قبل از آنکه تخمین جدید محاسبه شود، یک خط dummy قرار می دهیم مثلاً تعریف می کنیم $dummy = ping[0]$ تا خطی برای قرار دادن break point در نرم افزار CCS باشد. (بعد از load برنامه در پردازندۀ از منوی

- break گرینه animate را انتخاب کنید، به این ترتیب نرم افزار CCS هنگام رسیدن به point پنجره های خود را به روز می کند و اجرای برنامه را ادامه می دهد.)
- f. آرایه spectrum را پاک کنید و دوباره تخمین طیف را از سر بگیرید.
- g. هنگام کامپایل از بهینه سازی کامپایلر استفاده کنید.

۶-۴-۳-۲. تست اسپکتروم با داده مشخص

درتابع ISR به جای پر کردن آرایه ping یا pong با داده ورودی از پورت سریال، آن را با مقدار $10000 \cos\left(n \times 100 \times \frac{2\pi}{1024}\right)$ صحیحی که از ADC وارد می شود مدل می کند.

درتابع $\cos()$ به طور بازگشتی مقدار قبلی آرگومان آن را با $\frac{2\pi}{1024} \times 100$ جمع بزنید و در صورتی که از مقدار 2π فراتر رفت مقدار 2π را از آن کم کنید.

ثبت کنید fft این سیگنال برای $n=100$ و $n=924$ باشد و برای سایر مقادیر صفر. با توجه به این موضوع با مشاهده مقدار متغیر spectrum نتیجه طیف سنج خود را بررسی کنید. این کار را برای فرکانس‌های دیگر نیز تکرار کنید.

۶-۴-۳-۳. ایجاد نمایش گرافیکی برای اسپکتروم در نرم افزار CCS

در این قسمت برای اسپکتروم آنالایزر خود با استفاده از قابلیتهای break point و گرافیکی و animate نرم افزار CCS پنجره نمایش بسازید. برای این منظور مراحل زیر را طی کنید:

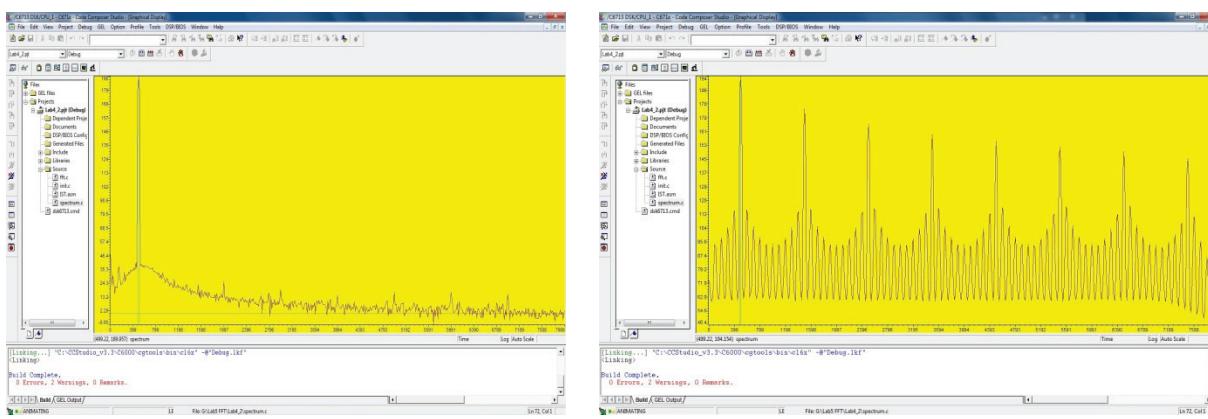
- ۱- در جلوی خط [0] dummy=ping یک break point قرار دهید (به این ترتیب یک دایره قرمز رنگ در حاشیه سمت چپ این خط دیده می شود).
- ۲- از منوی View گزینه Graph و سپس Time/Frequency... را انتخاب نمایید و تنظیمات زیر را وارد کنید:

Start address	Spectrum
Acquisition buffer size	513
Display data size	513
Dsp data type	32-bit floating point

-۳- روی دکمه animate کلیک کنید یا از منوی debug گزینه animate را انتخاب کنید.

۶-۴-۳-۴. تست اسپکتروم آنالایزر با داده خارجی

سیگنال ژنراتور را به ورودی DSK وصل کنید. در تابع ISR نمونه داده‌ها را از پورت سریال در یافت کنید. با تنظیم سیگنال ژنراتور سیگنال ورودی را سینوسی، مربعی، نویز، سینوسی، مربعی، نویز، سینوسی، مربعی با فرکانس FM تغییر دهید و طیف آنها را بررسی نمایید. شکل ۶-۶ تصویر طیف سیگنال سینوسی و مربعی با تکرار ۵۰۰ Hz را که با استفاده از اسپکتروم آنالایزر دیجیتال در این آزمایش به دست آمده است، نشان می‌دهد.



شکل ۶-۶- تصویر طیف سیگنال‌های سینوسی و مربعی با فرکانس تکرار ۵۰۰ Hz

۶-۴-۴. قسمت سوم: استفاده از توابع پردازش سیگنال TI (اختیاری)

به جای تابع ()fft از تابع آماده DSPF_sp_cfftr2_dit به نام TI استفاده کنید و تعداد کلاک لازم برای آن را با تعداد کلاک لازم برای تابع ()fft مقایسه کنید.

مرجع:

A. Tretter, Communication System Design Using DSP Algorithms with Laboratory Experiments for TMS320C6713 DSK, Springer, 2000

۷ آزمایش ششم: ممیز ثابت (fixed-point)

۱-۷ هدف

در این آزمایش محاسبات صحیح و برخی اثرات ناشی از نمایش اعداد با تعداد بیت محدود شامل کوانتیزاسیون و سرریز بررسی می‌گردند.

۲-۷ مقدمه

پردازنده‌های دیجیتالی برای نمایش اعداد به ناچار از تعدادی بیت محدود استفاده می‌کنند که این موضوع منجر به ایجاد اثراتی در پیاده سازی الگوریتمها می‌شود. این اثرات شامل مباحث کوانتیزاسیون، سرریز، محاسبات صحیح و ... می‌گردد که در عمل چالش‌هایی به دنبال دارد. برای نمایش اعداد در سیستمهای دیجیتالی روش‌های نمایش متنوعی وجود دارند. پردازنده‌های DSP از نظر قابلیت انجام محاسبات حقیقی به دو دسته floating-point و fixed-point تقسیم بندی می‌گردند. پردازنده‌های floating-point قابلیت سخت افزاری برای انجام محاسبات ضرب و جمع حقیقی دارا می‌باشند، از این لحاظ برنامه نویسی آنها آسان‌تر و سریع‌تر است. در مقابل پردازه‌های fixed-point هستند که واحدهای پردازنده محاسباتی آنها صحیح هستند. این پردازنده‌ها عموماً ارزان‌تر، سریع‌تر و با توان مصرفی کمتر هستند. البته fixed-point بودن آنها به معنای آن نیست که در برنامه نویسی آنها با زبان C از متغیرهای حقیقی نظیر float نمی‌توان استفاده کرد. در حقیقت برای پیاده سازی محاسبات حقیقی بر روی آنها به صورت نرم افزاری محاسبات حقیقی بر روی واحدهای صحیح پیاده می‌شوند که این موجب تحمیل محاسبات بیشتر می‌گردد.

در این آزمایش در قسمت تئوری با مباحث نمایش اعداد در سیستمهای دیجیتال، کوانتیزاسیون، محاسبات صحیح، سرریز آشنا می‌شویم. سپس در قسمت عملی ضمن پیاده سازی توابع کسینوس و رادیکال به صورت fixed-point با محاسبات صحیح و کارآیی آنها در پیاده سازی توابع غیر خطی آشنا می‌شویم. همان طور که می‌دانید اغلب پردازنده‌های DSP به صورت fixed-point هستند. در قسمت بعدی آزمایش یک فیلتر IIR را به صورت fixed-point پیاده سازی می‌کنیم و تفاوت ساختار پیاده سازی را مشاهده می‌کنیم.

۳-۷ تئوری

۳-۱ نمایش اعداد

نمایش اعداد حقیقی:

معمولًا در پردازنده‌ها برای نمایش اعداد حقیقی از استاندارد IEEE 754 استفاده می‌کنند. در این استاندارد برای نمایش اعداد float (single precision) از ۳۲ بیت استفاده می‌شود که این بیتها به سه قسمت تقسیم می‌شوند. یک بیت علامت، هشت بیت برای نمایش توان و ۲۳ بیت برای نمایش قسمت اعشاری. شکل ۱-۷ تقسیم بندی این ۳۲ بیت را نشان می‌دهد. بیت s بیت علامت است. هشت بیت ($e_7 \dots e_0$) بیت توان

هستند که معادل مبنای ۱۰ آنها $e = \sum_{k=0}^7 e_k 2^k$ است و طبق این رابطه $0 \leq e \leq 255$ است. ۲۳ بیت

$f = \sum_{k=1}^{23} f_k 2^{-k}$ قسمت کسری یا مانتیس هستند و معادل مبنای ۱۰ آنها $f = (f_1 \dots f_{23})$

در مبنای ۲ است و به این ترتیب مانتیس در بازه $[0, 1 - 2^{-23}]$ قرار دارد. در این روش نمایش اگر $0 < e < 255$ باشد، مقدار حقیقی متناظر با این نمایش $(1+f) \cdot 2^{e-127}$ است.

31	30	23	22	0
s	e_7, e_6, \dots, e_0	$f_1, f_2, \dots, f_{22}, f_{23}$		

شکل ۱-۷ تقسیم بندی بیتها نمایش عدد حقیقی ۳۲ بیتی

برای نمایش اعداد اعشاری با دقت مضاعف^۱ ۶۴ بیت استفاده می‌شود. یک بیت علامت، ۵۲ بیت برای قسمت کسری و ۱۱ بیت برای توان.

چون در این روش نمایش قسمت توان وجود دارد عملکرد قسمت توان مانند آن است که نقطه اعشاری در عدهای مختلف جابجا می‌شود و از این رو به این طرز نمایش floating-point می‌گویند.

نمایش اعداد صحیح به روش مکمل ۲:

امروزه معمولاً در سیستمهای دیجیتال برای نمایش اعداد صحیح از روش نمایش مکمل ۲ استفاده می‌شود. در این روش اگر B بیت استفاده شود معادل $b_B \dots b_1 b_0$ در مبنای ۱۰ برابر

^۱ Double-precision

$[-2^{B-1}, 2^{B-1}-1]$ است. با این طرز نمایش کلیه اعداد صحیح در بازه $-2^{B-1} + \dots + b_1 \times 2^1 + b_0 \times 2^0$ قابل نمایش هستند. به عنوان مثال با چهار بیت نمایش 0111b معادل 7 و 1000b معادل 8 می‌باشد. حسن روش نمایش مکمل ۲ آن است که جمع و تفریق در یک واحد محاسباتی انجام می‌شوند.

Nمایش Format

برای نمایش اعداد صحیح و محاسبات fixed-point می‌توان فرمت نمایش q-فرمت را در نظر گرفت.

در فرمت Qm,n عدد $m+n$ بیتی به صورت $b_{m-1}\dots b_1 b_0 b_{-1} \dots b_{-n}$ تفسیر می‌شود. یعنی $m-1$ بیت صحیح و n بیت کسری و یک بیت علامت. به عنوان مثال نمایش Q1,15 را در نظر بگیرید (با فرض اینکه نمایش اعداد با ۱۶ بیت است می‌توان Q15 نوشت) نمایش 0x4000 معادل 0.5 در مبنای ۱۰ است یا 20480- معادل 0.625- در مبنای ۱۰ می‌باشد. برای تبدیل یک عدد حقیقی به فرمت Qm,n آن را در 2^n ضرب می‌کنیم و سپس گرد می‌کنیم. همچنین برای تبدیل یک عدد Qm,n به مبنای ۱۰ آن را در 2^{-n} ضرب می‌کنیم. به این ترتیب برای تبدیل یک عدد از Qr,s به Qm,n آن عدد را در 2^{s-n} ضرب می‌کنیم. در برنامه نویسی ضرب در 2^n معادل n بیت انتقال^۱ به چپ و ضرب در 2^{-n} معادل n بیت انتقال به راست است. به عنوان مثال برای تبدیل یک عدد 29 به Q2,29 آن عدد اول را ۱۴ بیت به راست انتقال می‌دهیم.

۲-۳-۷ محاسبات صحیح

برای انجام محاسبات صحیح q-فرمت، برنامه نویس باید نحوه تفسیر اعداد مختلف را در نظر داشته باشد و در برنامه به نحو مناسب، محاسبات را انجام دهد. ایده کلی برای انجام محاسبات fixed-point همان چیزی است که در سالهای ابتدایی تحصیل آموخته‌ایم. برای جمع دو عدد باید مکان ممیز (نقطه اعشاری) در آنها یکسان باشد و حاصل ضرب دو عدد Qm,n و Qr,s به صورت عدد $Qm+r-1,n+s$ است و رقم قبل ممیز وجود دارد. به عنوان مثال حاصل ضرب دو عدد ۱۶ بیتی با نمایش Q1,15 یک عدد با نمایش Q1,30 است و در صورتی که حاصل ۳۲ بیتی باشد بیت MSB بیت علامت تکرار شده است. در برنامه

^۱ Shift

نویسی عملوندهای ضرب ۱۶ بیتی هستند و متغیر حاصل ضرب ۳۲ بیتی در نظر گرفته می‌شود. سپس متغیر حاصل ضرب دوباره به صورت ۱۶ بیتی تبدیل می‌گردد. در نمایش ۹-فرمت بیتهای پرازش حاصل ضرب نگه داشته می‌شوند و از بیتها کم ارزش صرف نظر می‌گردد.

مثال: برای جمع دو عدد A و B که اولی با فرمت Q4,4 و دومی با فرمت Q0,8 نمایش داده شده‌اند باید مکان ممیز را در آنها یکسان نمود که برای این کار دو امکان وجود دارد: یکی $(A <> 4) + B$ و دیگری C $(B >> 4) + A$ می‌باشد. (علامت <> و >> به معنای انتقال به راست و انتقال به چپ در برنامه نویسی C می‌باشند). برای گرد کردن B می‌توان از کد $(B + 8) >> 4 + A$ استفاده کرد. در حالت کلی $x = (x + 2^{m-1}) >> m$ عدد x را بر 2^m تقسیم می‌کند و حاصل را گرد می‌کند.

۳-۳-۷ خطای کوانتیزاسیون

برای نمایش مقادیر حقیقی با تعداد بیت محدود باید مقدار واقعی را به مقدارهای مشخصی کوانتایز کنیم. خطای کوانتیزاسیون به دو دسته کوانتیزاسیون سیگنال و کوانتیزاسیون ضرایب تقسیم می‌شود.

خطای کوانتیزاسیون سیگنال:

در مبدل‌های آنالوگ به دیجیتال، سیگنال آنالوگ ورودی نمونه برداری می‌شود تا به سیگنال گسسته زمان تبدیل شود. سپس مقادیر سیگنال با تعدادی بیت محدود نمایش داده می‌شوند. فرض کنید مقادیر سیگنال گسسته با B بیت به طور یکنواخت مقدار یک نمونه را نشان دهند. به این ترتیب تعداد سطوح کوانتیزاسیون برابر 2^B می‌شود و خطای کوانتیزاسیون برابر $e[n] = x[n] - x(nT)$ است. طبیعتاً هرچه تعداد بیشتر بیت برای نمایش مقدار سیگنال استفاده کنیم خطای کوانتیزاسیون کاهش می‌یابد. خطای کوانتیزاسیون به نحوه تبدیل مقدار سیگنال نمونه برداری شده $(nT)x$ به سیگنال گسسته اندازه $[n]$ نیز بستگی دارد.

خطای کوانتیزاسیون ضرایب:

معمولًاً روش‌های طراحی فیلتر، ضرایب حقیقی برای فیلتر می‌دهند. با کوانتایز کردن این ضرایب احتمالاً پاسخ فرکانسی فیلتر تغییر می‌کند که این تغییر ممکن است ناچیز و قابل چشم پوشی باشد و یا تا حدی باشد که مثلاً یک فیلتر IIR پایدار را به یک فیلتر ناپایدار تبدیل کند. برای کنترل کردن این خطا از جمله راه حل‌هایی

که وجود دارد نمایش ضرایب با تعداد بیت بیشتر، و همچنین استفاده از ساختارهای مختلف پیاده سازی فیلتر می‌باشد.

۳-۴-خطای محاسبات

سرریز:

بعد از انجام محاسبات در صورتی که حاصل به تعداد بیت بیشتر برای نمایش نیاز داشته باشد گوییم سرریز اتفاق افتاده است. به عنوان مثال حاصل جمع دو عدد $x_1 = 0.875$ و $x_2 = 0.125$ (0001b) می‌شود که معادل 1 است و در مقایسه با که به روش Q1,3 و با چهار بیت نمایش داده شده‌اند برابر b 1000 می‌شود که معادل 1 است و در مقایسه با مقدار واقعی 1 خطای قابل توجه می‌باشد. یا اگر $x_3 = -0.5$ (1100b) باشند حاصل $x_4 = 0.625$ (0101b) است که معادل نمایش 0.875 است، در حالی که مقدار صحیح 1.125 است. برای بهبود مسئله سرریز فعلاً راه حل کارا تغییر مقیاس^۱ می‌باشد. به این ترتیب که ورودی‌ها یا ضرایب را با یک تغییر مقیاس در نظر می‌گیرند. همچنین اشباع محاسبات^۲ نیز به کاهش اثر سرریز کمک می‌کند. منظور از اشباع محاسبات قابلیتی است که در صورتی که سرریز رخ داد نتیجه محاسبه به صورت ماکزیمم یا مینیمم قابل نمایش ذخیره گردد. مثلاً در مثال اول حاصل $x_1 + x_2$ برابر 0111b معادل 0.875 ذخیره می‌شود و در مثال دوم حاصل $x_3 - x_4$ برابر b 1000 می‌شود که همان طور که دیده می‌شود خطای کمتری است. قابلیت اشباع محاسبات در برخی پردازنده‌ها وجود دارد که با یک بیت کترلی فعال و غیر فعال می‌شود.

روند کردن^۳:

می‌دانیم به طور کلی بعد از جمع کردن دو عدد B بیتی، حاصل B+1 بیت می‌شود یا حاصل ضرب دو عدد B بیتی که به روش مکمل 2 نمایش داده شده‌اند برابر $-B-1$ 2B-1 بیت می‌شود. بنابراین بعد از انجام محاسبات در صورتی که فرض کنیم می‌خواهیم باز با همان تعداد بیت حاصل را نمایش دهیم، احتمال دارد که از دقت محاسبات کاسته شود. برای کاهش این خطا در پردازنده‌های DSP معمولاً برای واحدهای محاسباتی تعدادی

¹ Scaling

² Saturation arithmetic

³ Roundoff error

گارد بیت در نظر می‌گیرند مثلاً رجیستر accumulator را به جای ۳۲ بیت، ۴ بیت قرار داده‌اند تا لازم نباشد در حین محاسبات بی در پی نظیر فیلتر کردن، مرتب نتیجه محاسبات را روند کنند و فقط نتیجه نهایی را گرد می‌کنند که منجر به خطای کمتر می‌شود.

۷-۴ آزمایش

۷-۴-۱ تجهیزات مورد استفاده

کامپیوتر، نرم افزار CCS، نرم افزار MATLAB، اسیلوسکوپ و سیگنال ژنراتور

۷-۴-۲ قسمت اول: پیاده سازی تابع کسینوس به صورت fixed-point

اغلب پردازنده‌های DSP به صورت fixed-point هستند. برای استفاده کارا از این پردازنده‌ها باید حتی‌الامکان محاسبات به صورت صحیح در آنها انجام شود. برای این منظور اعداد حقیقی را به صورت q-فرمت مناسب تبدیل می‌کنیم. یکی از چالشهای محاسبات fixed-point پیاده سازی تابع غیر خطی نظیر تابع مثلثاتی، تابع لگاریتم، تابع نمایی و تقسیم می‌باشد. در این آزمایش ضمن پیاده سازی تابع کسینوس به صورت fixed-point با محاسبات صحیح نیز آشنا می‌شوید.

برای پیاده سازی تابع کسینوس به صورت fixed-point از بسط مک‌لورن آن استفاده می‌کنیم که در زیر آمده است:

$$\cos(\theta) = 1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - \frac{1}{6!}\theta^6 + \dots$$

در این قسمت از نرم افزار CCS به صورت شبیه ساز پردازنده TMS320C6416 استفاده می‌کنیم. پردازنده‌های خانواده 64XX پردازنده‌های fixed-point هستند.

الف) ابتدا برای محاسبه کسینوس تابعی به نام `fcos1` می‌نویسید که آرگومان آن `float` است و محاسبات انجام می‌دهد. به این منظور پروژه‌ای در نرم افزار CCS ایجاد کنید و از کد شکل ۲-۷ برای محاسبه تابع `float` کسینوس استفاده نمایید. همچنین ابتدا در قسمت `setup` نرم افزار CCS پردازنده C6416 Device Cycle

little Endian و Accurate Simulator را انتخاب کنید.

یادآوری: وقتی پردازنده 6416 را انتخاب می‌کنید باید کتابخانه rts6400.lib را به پروژه اضافه کنید.

```
float fcoef[4]={1.0,-1/2.0,1.0/(2.0*3.0*4.0),
                -1.0/(2.0*3.0*4.0*5.0*6.0)};
float fcose1(float x)
{
    float out;
    out=fcoef[0];
    out+=fcoef[1]*x*x;
    out+=fcoef[2]*x*x*x*x;
    out+=fcoef[3]*x*x*x*x*x*x;
    return out;
}
```

شکل ۷-۲ پیاده سازی تابع کسینوس به روش اول و به صورت محاسبات ممیز شناور

برای تست تابع fcose1 مقدار تابع را برای مقادیر جدول ۱-۷ محاسبه کنید و با مقدار مورد نظر مقایسه کنید.

سؤال: با توجه به نحوه پیاده سازی تابع کسینوس به این صورت توضیح دهید چرا باید آرگومان کسینوس عددی بزرگ نباشد.

این تابع را علاوه بر پردازنده TMS320C6416 بر روی پردازنده TMS320C6713 (به عنوان سیمولاتور) که یک پردازنده floating-point است اجرا کنید و تعداد کلاک لازم برای اجرا آن را تعیین کنید و در جدول ۲-۷ وارد کنید. لازم به ذکر است که در صورتی که در CCS پروژه ای برای اجرا بر روی پردازنده 6416 ساخته اید و بخواهید آن را بر روی پردازنده 6713 اجرا کنید ضمن آنکه در setup نرم افزار CCS 6713 را انتخاب می‌کنید بعد از آنکه در نرم افزار CCS پروژه قبلی را باز کردید باید در پنجره Option خانواده پردازنده را نیز عوض کنید.

ب) برای پیاده سازی یک تابع ایده های متفاوت و خلاق گونه ای وجود دارد. در این قسمت بسط تابع کسینوس به صورت دیگری دیده و پیاده می شود که از نظر محاسباتی کاراتر است. تابع fcose2 را که در شکل ۳-۷ آمده است در پروژه CCS خود وارد کنید و ستونهای مربوطه در جدول ۱-۷ و جدول ۲-۷ را برای آن پر

کنید.

```
float fcoef[4]={1.0,-1/2.0,1.0/(2.0*3.0*4.0),
                -1.0/(2.0*3.0*4.0*5.0*6.0)};
float fcos2(float x)
{
    float out,x2;
    x2=x*x;
    out=x2*fcoef[3];
    out=(out+fcoef[2])*x2;
    out=(out+fcoef[1])*x2;
    out+=fcoef[0];
    return out;
}
```

شکل ۳-۷ پیاده سازی تابع کسینوس به روش دوم و به صورت محاسبات ممیز شناور

پ) در این قسمت تابع کسینوس را به صورت fixed-point پیاده می نماییم. کد تابع `icos` را که در شکل ۷-۴ آمده است بررسی نمایید. در اینجا ضرایب بسط با فرمت Q1,15 نمایش داده می شوند. متغیر `x` ورودی تابع `icos` نیز با فرمت Q2,14 در نظر گرفته شده است، یعنی $-2 \leq x < 2$. در اینجا متغیر ۳۲ بیتی cosine که مقدار کسینوس محاسبه شده در هر مرحله است و مقدار آن به روز می شود حاصل ضرب دو متغیر ۱۶ بیتی را نگه می دارد. در اینجا مقدار حاصل ضرب در هر مرحله در بازه $(-1,1]$ قرار می گیرد (این نکته باید بررسی شود و ممکن است در حالت کلی درست نباشد ولی در اینجا صحیح است).

سؤال: چرا مقدار متغیر cosine ، ۱۳ بیت به راست انتقال پیدا کرده است؟

```
#define UNITQ15 0x7fff
short iCoef[4]={ (short) (UNITQ15), (short) (-(UNITQ15/2.0)),
                  (short) (UNITQ15/(2.0*3.0*4.0)),
                  (short) (-(UNITQ15/(2.0*3.0*4.0*5.0*6.0)))};
short icos(short x)
{ long cosine,z;
  short x2;
  z = (long)x * x;
  x2 = (short)(z>>15); // x2 has x(Q14)*x(Q14)
  cosine = (long)iCoef[3] * x2;
  cosine = cosine >> 13;
  cosine = (cosine + (long)iCoef[2]) * x2;
  cosine = cosine >> 13;
  cosine = (cosine + (long)iCoef[1]) * x2;
  cosine = cosine >> 13;
  cosine = cosine + iCoef[0];
  return((short)cosine);
}
```

شکل ۷-۸ پیاده سازی تابع کسینوس به صورت محاسبات ممیز ثابت

با پر کردن جدول ۱-۷ و جدول ۲-۷ صحت و کارآیی پیاده سازی تابع کسینوس را به صورت fixed-point بررسی کنید. در مواردی که نتیجه صحیح نمی باشد علت را توضیح دهید.

جدول ۱-۷ نتیجه تست پیاده سازی های مختلف برای تابع کسینوس

θ	$\cos(\theta)$	fco1	fco2	Icos
0	1.0000			
0.1	0.9950			
-0.8	0.6967			
19.373155	0.8660			
-0.523592	0.8660			
1.5707963	0.0000			
3.141592	-1.0000			

جدول ۲-۷ تعداد کلاک برای اجرای پیاده سازی های مختلف برای تابع کسینوس بر روی دو پردازنده

تابع کسینوس	TMS320C6416 پردازنده	TMS320C6713 پردازنده
fco1		
fco2		
Icos		

۷-۴-۳- قسمت دوم: پیاده سازی تابع رادیکال به صورت fixed-point

تابع رادیکال را با استفاده از رابطه زیر به صورت floating-point و همچنین fixed-point برای $0.5 \leq x \leq 1$ در CCS بر روی پردازنده TMS320C6416 (به صورت شبیه ساز) پیاده سازی نمایید و برای تست صحت آن جدول ۳-۷ و برای تست کارآیی جدول ۴-۷ را پر کنید.

$$\sqrt{x} = 0.2075806 + 1.454895x - 1.34491x^2 + 1.106812x^3 - 0.536499x^4 + 0.1121216x^5$$

جدول ۳-۷ نتیجه تست پیاده سازی تابع رادیکال به صورت fixed-point

x	0.5	0.6	0.7	0.8	0.9
\sqrt{x}	0.7071	0.7746	0.8367	0.8944	0.9487
\sqrt{x} (floating-point)					
\sqrt{x} (fixed-point)					

جدول ۴-۷ تعداد کلاک لازم برای اجرای تابع رادیکال بر روی پردازنده TMS320C6416

تابع رادیکال	fixed-point	floating-point
\sqrt{x}		

۷-۴-۴. قسمت سوم: پیاده سازی فیلتر IIR به صورت fixed-point (اختیاری)

در این قسمت یک فیلتر IIR طراحی می‌کنید و آن را به صورت fixed-point پیاده سازی می‌کنید.

فرکانس نمونه برداری را 8000 Hz در نظر بگیرید.

الف) در نرم افزار MATLAB به کمک ابزار fdatool یک فیلتر پایین گذر IIR از مرتبه ۴ با فرکانس قطع 2 از نوع باتوروث طراحی کنید. ساختار فیلتر را Direct Form II, Second order Section انتخاب نمایید. پاسخ فرکانسی فیلتر را با کمک MATLAB رسم کنید.

ب) پروژه ای در CCS برای پیاده سازی یک فیلتر IIR به صورت fixed-point ایجاد کنید.

پ) برنامه ای بنویسید که یک سیگنال تصادفی از حافظه که مقادیر آن در بازه $(-1,1)$ فرض می‌شوند را با فیلتر طراحی شده در قسمت الف فیلتر نماید. در برنامه C برای فیلترینگ فقط از متغیرهای صحیح استفاده نمایید (پیاده سازی به صورت fixed-point). همچنین در این قسمت ضرایب فیلتر را با q-فرمت مناسب با ۱۶ بیت نمایش دهید.

در نظر گرفتن نکات زیر در پیاده سازی fixed-point می‌تواند کمک کننده باشد:

برای جلوگیری از سرریز محاسبات، سیگنال ورودی را به میزان مناسب تغییر مقیاس دهید. مثلاً اگر سیگنال ورودی در متغیر x از نوع short است با دستور $x = \text{short}(2^8)$ آن را بر 2^8 تقسیم کنید و با توجه به خطی بودن

سیستم می‌دانید خروجی فیلتر بر 2^8 تقسیم شده است.

ضرایب فیلتر را با q -فرمت مناسب نشان دهید (Q1,15)

ت) فیلتر پیاده سازی شده در قسمت پ را تست نمایید. برای این منظور از ترم افزار MATLAB استفاده کنید. یعنی سیگنال تصادفی مشابه را در MATLAB با دقت double با فیلتر طراحی شده فیلتر نمایید و با نتایج فیلتر پیاده سازی شده به صورت fixed-point مقایسه کنید. دستور sosfilt() در MATLAB برای فیلتر کردن سیگنال با گرفتن ماتریس ضرایب بخش‌های درجه ۲^۱ (sos)، می‌تواند مفید باشد.

ث) برنامه ای بنویسید که سیگنال آنالوگ را از پورت Line-In دریافت کند و با فیلتر IIR قسمت الف فیلتر نماید و در پورت Line-Out بنویسد. برای کنترل خواندن و نوشتن از وقfe استفاده کنید.
ج) به ورودی Line-In از بورد DSK سیگنال نویز وارد کنید. و با مشاهده طیف سیگنال خروجی روی اسیلوسکوپ پاسخ اندازه سیستم را مشاهده نمایید.

¹ second order section

۵-۷ ضمیمه

۵-۷-۱ ابزار Fixed-point نرم افزار MATLAB

برای شبیه سازی الگوریتمها می توان از ابزار fixed-point نرم افزار MATLAB استفاده کرد. به عنوان مثال دستور quantizer یک شی کوانتایزر می سازد که از آن می توان در دستور quantize استفاده کرد و مقادیر حقیقی را کوانتایز نمود. مثال زیر کاربردی از استفاده از این دستورات را نشان می دهد.

مثال: برای کوانتایز کردن اعداد با فرمت Q1,15 می توان دستورات زیر را در MATLAB به کار برد:

```
q15=quantizer('fixed','round','saturate',[16,15]);
```

```
quantize(q15,0.7)
```

جدول ۵-۷ گزینه های مختلف و معانی آنها را به عنوان آرگومان دستور quantizer نشان می دهد.

جدول ۵-۷ لیست آرگومانهای ورودی برای تابع quantizer در MATLAB

Property name	Property value	Description
mode	'double'	Double-precision mode
	'float'	Custom-precision floating-point mode
	'fixed'	Signed fixed-point mode
	'single'	Single-precision mode
	'ufixed'	Unsigned fixed-point mode
	'ceil'	Round toward negative infinity
roundmode	'convergent'	Convergent rounding
	'fix'	Round toward zero
	'floor'	Round toward positive infinity
	'round'	Round toward nearest
overflowmode	'saturate'	Saturate on overflow
	'wrap'	Wrap on overflow
format	[B m]	Format for fixed or ufixed mode. B is wordlength, m is number of fractional bits

برای مشاهده q-فرمت یک عدد در مبنای ۲، ۱۶، ۱۰ می توان از دستورات num2hex و num2bin استفاده کرد.

۸ آزمایش هفتم: ارتباط MATLAB با CCS و پردازنده

۱-۸ هدف

آشنایی با برخی قابلیت‌های نرم افزار MATLAB در ارتباط با نرم افزار CCS و پردازنده که در عملیات تست الگوریتم و نمونه سازی سریع (Fast Prototyping) و پیاده سازی پردازنده در حلقه (Processor-in-the-Loop) به کار می‌آیند.

۲-۸ مقدمه

نرم افزار MATLAB نرم افزار محاسباتی و شبیه سازی قدرتمند و پرکاربردی در رشته مهندسی برق می‌باشد. در این نرم افزار ابزارهایی برای ارتباط با پردازنده‌های پردازش سیگنال Embedded IDE شرکت‌های مختلف از جمله TI وجود دارد. در این آزمایش با دستوراتی از نرم افزار Link از زیر مجموعه‌های MATLAB آشنا می‌شویم. به کمک این دستورات از درون نرم افزار MATLAB پروژه‌ای که برای افروzen یک واحد به مقادیر یک آرایه می‌باشد، در CCS باز می‌کنیم و مقادیر اولیه آرایه را مشخص می‌کنیم سپس برنامه را اجرا می‌کنیم و مقادیر آن آرایه را در MATLAB می‌خوانیم. در قسمت بعدی مسیر سر راستی برای آشنایی با نحوه ایجاد کد به صورت MATLAB در MATLAB و ایجاد پروژه از MATLAB برای CCS با استفاده از بلوک‌های Simulink سریع در MATLAB و ایجاد پروژه از CCS برای MATLAB با استفاده از چند بلوک از Simulink می‌پردازیم. در این قسمت یک تخمینگر طیف با استفاده از چند بلوک از Simulink می‌سازیم و بر روی پردازنده پیاده می‌نماییم. همچنین نرم افزار Target Support Package نیز معرفی می‌گردد. لازم به ذکر است که در این آزمایش نسخه MATLAB 2010Rb مد نظر قرار گرفته است. نسخه‌های قدیمی‌تر در نرم افزارهای Embedded IDE Link و Target Support Package با نسخه آخر از لحاظ دسته بندی بلوکها در Simulink و برخی قابلیت‌های موجود تفاوت‌هایی دارند.

۳-۸ تئوری

۱-۳-۸ نرم افزار Embedded IDE link

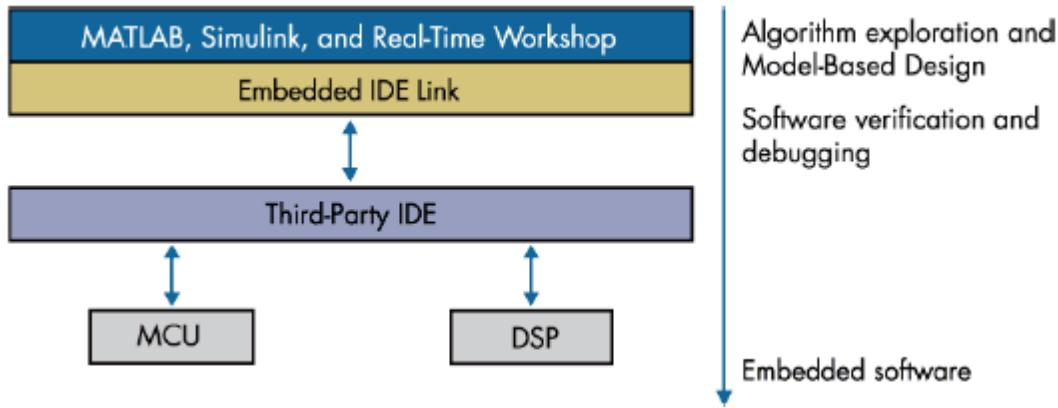
یکی از برنامه های زیر مجموعه MATLAB نرم افزار Embedded IDE Link می باشد. این نرم افزار ارتباط MATLAB و Simulink را با برخی نرم افزارهای ساخت شرکتهای دیگر شامل Altium ، Analog Devices Visual DSP++ ، Texas Instruments Code Composer Studio Eclipse IDE و Green Hills MULTI، TASKING از خانواده های TI's C2000، TI's C5000 و TI's C6000 در این نرم افزار پشتیبانی می شوند. برای دریافت اطلاعات از پردازنده هایی که توسط این نرم افزار پشتیبانی می شوند و قابلیت های موجود و نسخه های CCS مورد قبول برای این نرم افزار، می توانید از Help نرم افزار MATLAB کمک بگیرید. به کمک این ابزار می توان برای پردازنده از درون MATLAB کد تولید و Build نمود (code generation). به این ترتیب روشی سریع برای پیاده سازی الگوریتم های پردازش سیگنال برای برخی پردازنده های متداول در اختیار قرار می دهد. همچنین از طریق لینک ارتباطی که از طریق نرم افزارهای یاد شده با پردازنده ایجاد می کند می توان از قابلیت های MATLAB استفاده کرد و به تست الگوریتم پیاده سازی شده بر روی پردازنده پرداخت. همچنین قابلیت پردازنده در حلقه^۱ (یا به اسم دیگر سخت افزار در حلقه^۲) توسط این نرم افزار پوشش داده می شود. در شبیه سازی PIL قسمتی از برنامه در سخت افزار اجرا می شود و قسمتی بر روی کامپیوتر. شکل ۲-۸ شمایی از نحوه ارتباطی نرم افزار Embedded IDE Link با قسمت هایی از MATLAB و با پردازنده را نشان می دهد.

به طور مشخص در این آزمایشگاه که با نرم افزار CCS کار می کنیم، با کمک نرم افزار Embedded IDE Link می توان از درون MATLAB ارتباط با CCS برقرار کنیم و به باز کردن پروژه build کردن آن و load کردن برنامه بر روی پردازنده، قرار دادن breakpoint در کد برنامه، خواندن از حافظه پردازنده و نوشتمن در آن از طریق MATLAB اقدام نمود. یعنی می توان از داده های تولید شده در MATLAB برای تست برنامه استفاده کرد و از قابلیت های MATLAB نظری

¹ Processor-in-the-Loop

² Hardware-in-the-Loop

نمایش گرافیکی و همچنین پردازش‌های بیشتر بر روی داده‌های تولید شده در پردازنده در محیط استفاده نمود.



شکل ۱-۸ ترکیب نرم افزارهای MATLAB، Simulink و Real-Time Workshop برای ایجاد محیطی جهت تولید، خطایابی و تست کد برای پردازنده‌های سیگنال (DSP) و میکروکنترلرهای (MCU)

برای آشنایی اولیه می‌توانید با اجرای دستور ccstutorial در خط فرمان MATLAB اطلاعاتی کسب نمایید و برای به دست آوردن اطلاعات بیشتر از مراجع Embedded IDE Link User's Guide و Embedded IDE link 4 User's Guide, For Use with Texas Instruments' Guide [۱۸] و [۱۹] و همچنین Help نرم افزار MATLAB استفاده نمایید.

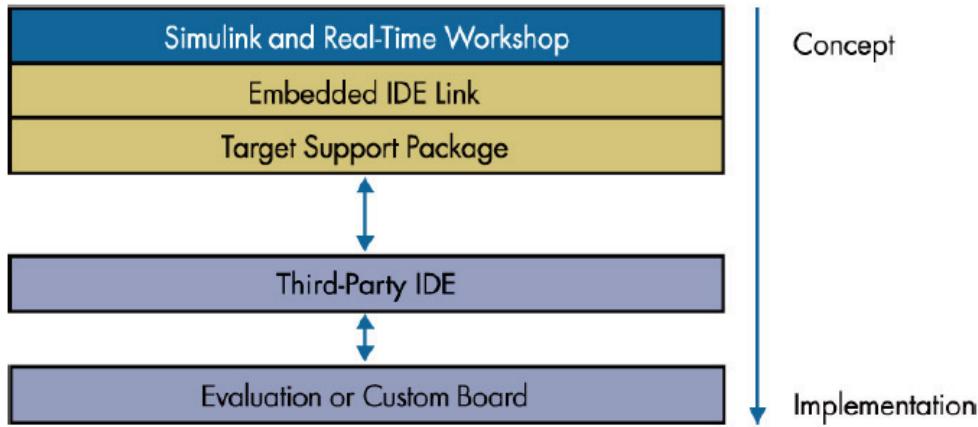
۳-۸ نرم افزار Target Support package

با استفاده از نرم افزار Target Support Package تجهیزات جانبی روی بوردهای پردازنده در کنار الگوریتمهای تولید شده در MATLAB و Simulink در دسترس قرار می‌گیرند. این ابزار از نرم افزار Embedded IDE Link نیز استفاده می‌نماید.

این نرم افزار از پردازنده‌های شرکت Texas Instruments (C5000, C2000, C6000)، STMicroelectronics و Infineon C166، Freescale MPC5xx، Analog Devices Blackfin و ST10 پشتیبانی می‌کند.

شکل ۳-۸ جایگاه این نرم افزار در ارتباط با نرم افزارهای MATLAB و با پردازنده را نشان

می‌دهد. از این نرم افزار مانند ابزار قسمت قبل برای تولید کد به صورت سریع^۱ استفاده می‌شود.



شکل ۲-۸ ترکیب برای Target Support Package و Embedded IDE Link .Real-Time Workshop ، Simulink طراحی، شبیه سازی و پیاده سازی الگوریتمهای پردازش سیگنال بر روی پردازنده‌های مختلف

برای کسب اطلاع بیشتر راجع به این نرم افزار می‌توانید از مرجع [۲۰] Help نرم افزار MATLAB User's Guide استفاده نمود.

۴-۴ آزمایش

۴-۱ تجهیزات مورد استفاده

بورد DSK 6713، نرم افزار MATLAB و Simulink و Embedded IDE Link (به همراه CCS)، نرم افزار (Target Support Package

^۱ Rapid prototyping

۲-۴-۸ قسمت اول: ارتباط MATLAB با CCS

در این قسمت طی یک آزمایش ساده یک لینک ارتباطی بین MATLAB و نرم افزار CCS از درون نرم افزار MATLAB با استفاده از ابزار Embedded IDE Link نرم افزار MATLAB برقرار می کنیم.

- ۱- یک پروژه در نرم افزار CCS با عنوان addOne ایجاد کنید. سپس یک برنامه C به این پروژه اضافه کنید که مقدار آرایه سراسری به نام data_Array از نوع float و به طول 10 را یک واحد اضافه کند. پروژه خود را تکمیل کنید و ذخیره نمایید و از نرم افزار CCS خارج شوید.
- ۲- در نرم افزار MATLAB یک m فایل بنویسید که کارهای زیر را انجام دهد. بهتر است مراحل زیر را خط به خط اجرا نمایید و نتایج آن را مشاهده نمایید.

(الف) ابتدا با اجرای دستور ccsboardinfo از وضعیت CCS نرم افزار آگاه شوید. سپس با استفاده از دستور tics، که شماره بورد و شماره پردازنده را می گیرد، یک شیئ برای ارتباط با نرم افزار CCS بسازید. شماره بورد و شماره پردازنده را از نتیجه دستور ccsboardinfo انتخاب می کنیم. کد زیر نمونه ای از استفاده از این دستور را نشان می دهد.

```
cc=ticcs('boardnum',0,'procnum',0)
```

در این دستور عبارات داخل ' عیناً نوشته شوند. بعد از اجرای این دستور نرم افزار CCS در پس زمینه اجرا می شود که با اجرای ... Windows Task Manager می توان به این موضوع پی برد و همچنین با اجرای دستور visible(cc,1) نرم افزار CCS نمایش داده می شود.

(ب) در ادامه پروژه مورد نظر که در اینجا پروژه addOne می باشد به کمک دستور open از درون نرم افزار CCS باز می کنیم. این دستور به آدرس محل قرار گرفتن فایل پروژه مورد نظر نیاز دارد.

```
open(cc,'c:\.....\addOne.pjt')
```

بعد به کمک دستور build آن را در نرم افزار CCS کامپایل و لینک می کنیم. و بعد با اجرای دستور load آن را در حافظه پردازنده بار می کنیم.

```
build(cc)
```

```
load(cc,'c:\.....\addOne\debug\addOne.out')
```

(پ) در این قسمت می خواهیم آرایه data_Array را از درون MATLAB با مقادیر ۱ تا ۱۰

مقدار دهی کنیم. برای این منظور از دستور write در MATLAB استفاده می‌کنیم. اما این دستور به آدرس شروع آرایه data_Array نیاز دارد. دستور آدرس address یک متغیر در حافظه پردازنده را برگرداند. کدهای زیر نمونه استفاده از این دو دستور را نشان می‌دهند. در دستور write باید مشخص کنیم نوع داده برای نوشتن در حافظه پردازنده چیست. در اینجا تابع single آرگومان ورودی را به نوع معادل float در زبان C تبدیل می‌کند.

```
data_Array_addr=address(cc,'data_Array')
write(cc, data_Array_addr, single([1:10]))
```

ت) با استفاده از دستور run در MATLAB برنامه را بر روی پردازنده اجرا کنید. همچنین دستور isrunning وضعیت اجرای برنامه در پردازنده را بر می‌گرداند.

```
run(cc)
isrunning(cc)
```

ث) بعد از اجرای برنامه با استفاده از دستور read در MATLAB محتوای آرایه data_Array را بخوانید و از صحت اجرای برنامه اطمینان حاصل کنید. نمونه استفاده از دستور read در زیر آمده است.

```
data_A=read(cc,data_Array_addr,'single', 10)
```

ج) در نهایت با دستور close به صورت زیر پروژه را در نرمافزار CCS ببندید.

```
close(cc, 'addOne','project')
```

و با پاک کردن شیئ cc از محیط MATLAB با CCS ارتباط MATLAB با CCS قطع می‌گردد.

```
clear cc
```

نکته: لازم به ذکر است که در این روش برای دسترسی به حافظه باید پردازنده متوقف باشد. که در صورت لزوم این امر با استفاده از قرار دادن breakpoint در خطوط مورد نظر در برنامه یا با استفاده از قابلیت اجرای animate محقق می‌گردد. دستورهای insert و remove برای قرار دادن breakpoint در برنامه در مکانهای مورد نظر مورد استفاده قرار می‌گیرند.

جدول ۱-۸ در قسمت ضمایم لیست متودهای موجود برای شیء ticcs در MTLAB برای ارتباط با نرم افزار CCS را نشان می‌دهد.

در نرم افزار CCS قابلیتی به نام رابط RTDX وجود دارد که امکان ارتباط پردازنده با نرم

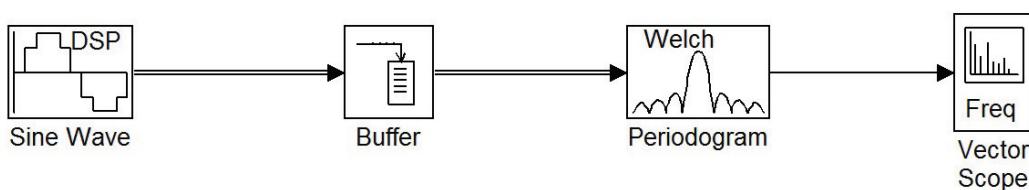
افزارهای جانبی نظیر visual C++، Excel را در حین اجرای برنامه بر روی پردازنده، فراهم می‌کند. در ورژنهای جدید نرم افزار MATLAB برای پردازنده‌های سری C6000 قابلیت RTDX را پشتیبانی نمی‌شود.

۴-۳-۸ قسمت دوم: پردازنده در حلقه (PIL)

در این قسمت از ابزارهای Embedded IDE Link و Target Support Package استفاده می‌کنیم تا با استفاده از بلوکهای سیمولینک پروژه‌ای برای اجرا بر روی پردازنده ایجاد کنیم. همان طور که خواهید دید این روش برای ایجاد کد، یک روش سریع برای پیاده‌سازی الگوریتمهای پردازش سیگنال می‌باشد، که البته لزوماً کدهای تولید شده به این روش بهینه نمی‌باشند. در این قسمت می‌خواهیم یک تخمینگر طیف داشته باشیم. در آزمایش پنجم طی پروژه‌ای در نرم افزار CCS الگوریتمی برای تخمین طیف بر روی پردازنده پیاده نمودیم. در اینجا چنین الگوریتمی بر روی پردازنده با استفاده از نرم افزار MATLAB در زمان کم ایجاد می‌کنیم و با برخی قابلیتهای Processor-in-the-Loop (PIL) قسمتی از پروژه MATLAB نظیر (PIL) پردازشی بر روی سخت افزار، که در این آزمایش بورد C6713 DSK می‌باشد، اجرا می‌گردد و قسمتی بر روی کامپیوتر PC اجرا می‌گردد. همچنین خواهیم دید که با استفاده از بلوکهای سیمولینک و ابزار Target Support Package چگونه می‌توانیم به سادگی یک پروژه مستقل برای اجرا بر روی پردازنده ایجاد کنیم.

۴-۳-۹ شبیه سازی در سیمولینک

ابتدا بلوک دیاگرامی برای شبیه سازی سیستم در سیمولینک مطابق شکل ۴-۸ بیندید.



شکل ۴-۸ بلوک دیاگرام شبیه سازی برای تخمینگر طیف در simulink

بعد از قرار دادن بلوکها در یک مدل سیمولینک، اکنون لازم است پارامترهای آنها را تنظیم

نمایید.

فرکانس نمونه برداری را ۸۰۰۰ Hz در نظر بگیرید. در بلوک Sine Wave پارامترها را به صورت زیر در نظر بگیرید.

- **Sample time** = ۱/۸۰۰۰
- **Samples per frame** = ۸۰
- **Form output after final data value by** = Setting to zero

قابلیت پردازش فریمی دارد که در این روش در هر زمان شبیه سازی^۱ به جای پردازش یک نمونه یک فریم را پردازش می نماید و به این ترتیب سرعت شبیه سازی بالاتر می رود. برای بلوک بافر پارامترها را به صورت زیر تنظیم نمایید.

Output buffer size (per channel) = ۱۲۸

Buffer overlap = ۴۸

Initial conditions = ۰

Treat Mx1 and unoriented sample-based signals as = One channel

در اینجا در تخمین طیف هر فریم با فریم قبلی به اندازه ۴۸ نمونه همپوشانی دارد.

برای تخمین غیرپارامتری طیف در بلوک periodogram پارامترهای زیر را تنظیم می کنیم.

Measurement = Power spectral density

Window = Hamming

Window sampling = Periodic

Select the Inherit FFT length from input dimensions check box.

Number of spectral averages = ۷

این بلوک بروی فریم داده ورودی پنجره Hamming اعمال می کند و طول فریم داده آن نیز

۱۲۸ می باشد که از بلوک قبلی بافر به آن وارد می شود.

برای انجام تنظیمات Vector Scope در پنجره تنظیمات آن موارد زیر را لحاظ می کنیم.

Input domain: Frequency

Frequency units: Hertz

Frequency range: [۰...Fs/2]

Inherit sample time from input: not checked

^۱ Simulation time

Sample time of Original time series: 1/8000

برای تنظیم پارامترهای شبیه سازی از منوی Simulation گزینه Configuration Parameters.

... را انتخاب نمایید و در قسمت Solver پارامترهای زیر را وارد نمایید.

- **Stop time** = inf
- **Type** = Fixed-step
- **Solver** = Discrete (no continuous states)

Tasking mode for periodic sample times: Single Tasking

حال با اجرای شبیه سازی نتیجه را بر روی پنجره Vector Scope مشاهده نمایید. برای تنظیم مقیاس محور عمودی، بر روی پنجره آن کلیک راست نمایید و گزینه Autoscale را انتخاب نمایید.

MATLAB Signal Processing Blockset document → Getting Reference: مرجع:

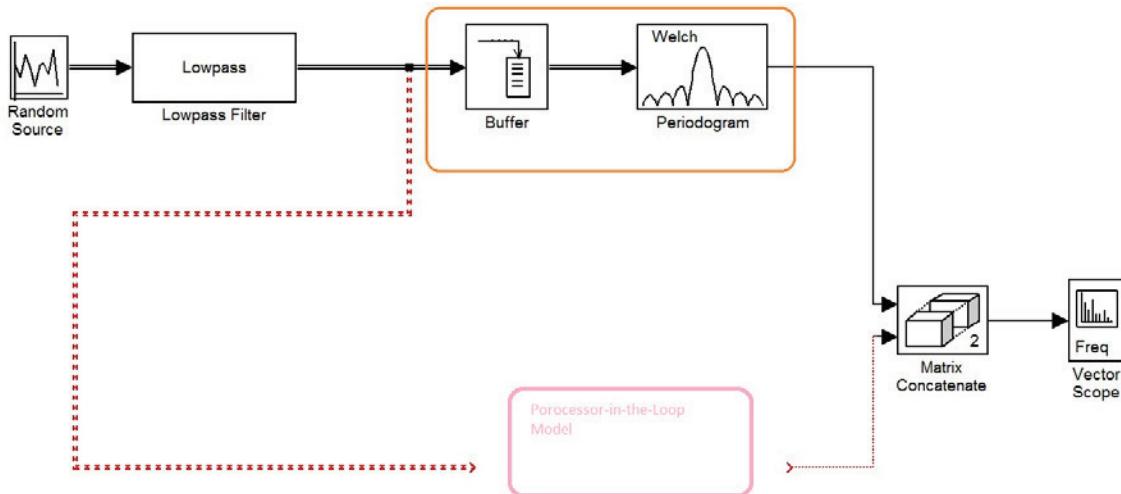
Started → Frequency Domain Signals → Power Spectrum Estimate

۴-۳-۲-۴ پیاده سازی PIL

در این قسمت مراحل انجام یک شبیه سازی به صورت PIL انجام می شود. با کمک قابلیتهای تولید کد MATLAB برای پردازنده و بورد C6713 TI ساخت تخمینگر طیف را بر روی پردازنده پیاده می کنیم. سیگنال ورودی به آن را نویز سفید فیلتر شده در MATLAB تولید می کنیم و نتیجه شبیه سازی را نیز در MATLAB بر روی پنجره Vector Scope مشاهده می نماییم.

به صورت PIL کمک می کند تا نتیجه پیاده سازی الگوریتم را بر روی پردازنده مشاهده نماییم.

شکل ۵-۸ مدل تغییر یافته قسمت قبل را برای این قسمت نشان می دهد. در این شکل به جای مستطیل Processor in the loop Model ابتدا باید بلوک آن را تولید که در مراحل بعدی این کار را انجام می دهید.



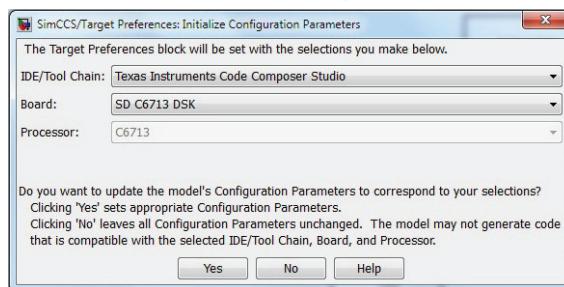
شکل ۸-۴ مدل سیمولینک برای شروع برای پیاده سازی تخمینگر طیف بر روی پردازنده به صورت PIL

الف) بورد را به کامپیوتر وصل کنید.

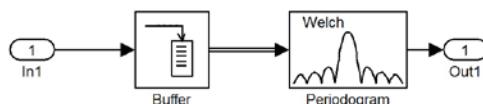
ب) از قسمتی از مدل که می خواهید در پردازنده نیز اجرا شود یک زیر سیستم بسازید. برای این کار بلوکهای بافر و تخمینگر طیف را انتخاب نمایید و کلید سمت راست را روی آن بزنید و از منوی باز شده Create subsystem را انتخاب نمایید.

پ) زیر سیستم مورد نظر که می خواهید بر روی پردازنده اجرا شود باز کنید و بلوک Target را از کتابخانه Embedded IDE Link یا Target Support Package به آن اضافه نمایید. همراه بلوک Target Preference نشان می دهد. این بلوک اتصالی برای ارتباط با بلوک دیگر ندارد. حین اضافه کردن این بلوک به مدل پنجره ای باز می شود که مشخصات پردازنده را جویا می شود.

تنظیمات آن را بر مبنای بورد C6713 DSK تنظیم کنید (شکل ۶-۸).



شکل ۸-۵ تنظیم پارامترهای بلوک Target Preference بر مبنای بورد مورد استفاده



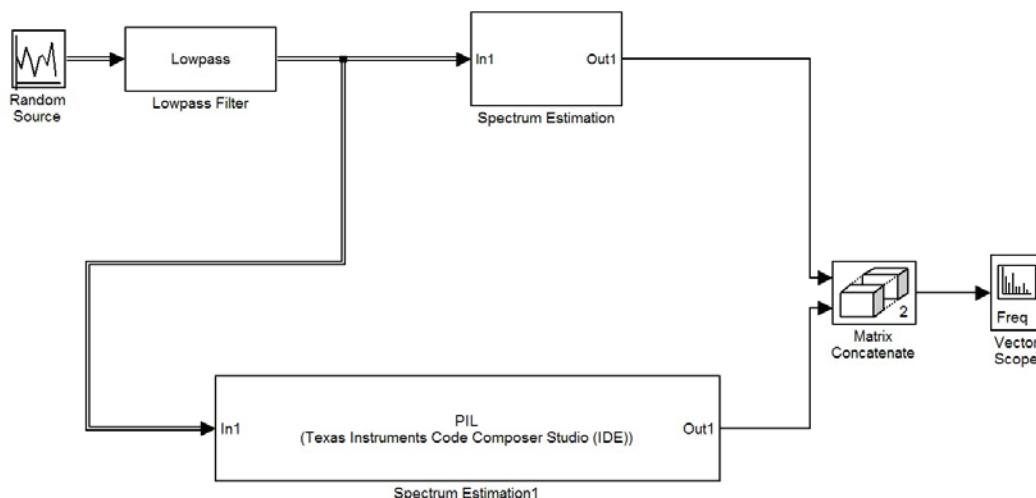
شکل ۶-۸ زیر مدل بلوکی جهت اجرا بر روی پردازنده

ت) برای انجام تنظیمات لازم، در قسمت Real-Time Workshop برای Target selection Embedded Browse فایل idelink_ert.tlc را انتخاب نمایید و همچنین قسمت Configuration Parameters Build تنظیم مربوط به ... Real-Time Workshop IDE Link Create_Processor_in_the_Loop_project action را انتخاب کنید.

MATLAB برای ساخت بلوک PIL از توابع MEX استفاده می‌نماید بنابراین برای تنظیم کامپایلر آن از دستور mex-setup در خط فرمان MATLAB استفاده نمایید.

ث) بر روی زیرسیستم مورد نظر راست کلیک نمایید و از منوی Real-Time Workshop گزینه Build را انتخاب نمایید. به این ترتیب یک پروژه نیز در نرم افزار CCS برای تخمین طیف بر روی پردازنده ایجاد می‌شود.

ج) در صورتی که عملیات ساخت بلوک PIL با موفقیت انجام شود، بلوک PIL تولید شده را کپی نمایید و در مدل اولیه قرار دهید. به این ترتیب مدلی شبیه شکل ۸-۸ خواهد داشت.



شکل ۷-۸ پیاده سازی پردازنده در حلقه برای تخمین گر طیف و مقایسه نتیجه اجرا بر روی پردازنده و بر روی PC

چ) با اجرای شبیه سازی به طور همزمان تخمینگر طیف در simulink (co-simulation) و در

پردازنده اجرا می‌شوند و نتیجه اجرای آنها به صورت همزمان بر روی یک Vector Oscop نمایش داده می‌شود.

ح) با استفاده از فایل Memory Map در نرم افزار CCS حجم کد تولید شده به صورت اتوماتیک توسط MATLAB برای این تخمینگر طیف را تعیین کنید و با حجم کد تخمینگر طیف که در آزمایش پنجم نوشته‌ید مقایسه نمایید.

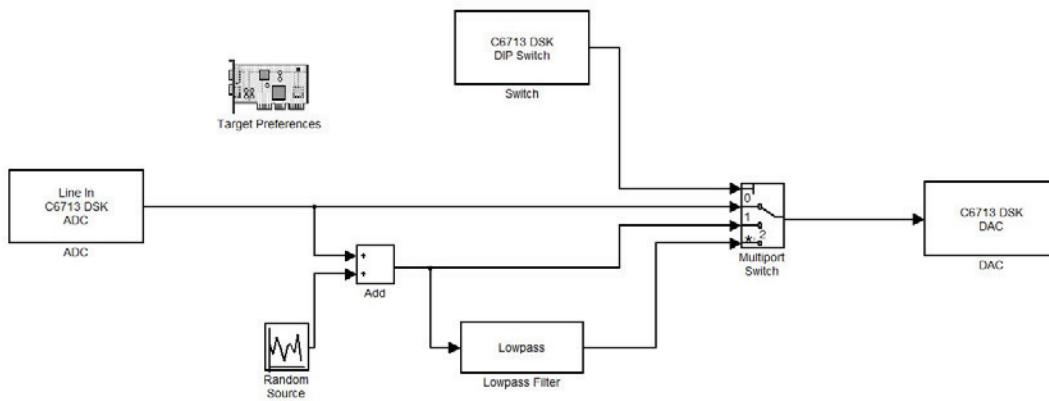
Embedded IDE Link (MATLAB Doc) → Demos → For Use with Texas مرجع:

Instruments Code Composer Studios → Workflows → Getting Started with Application Development

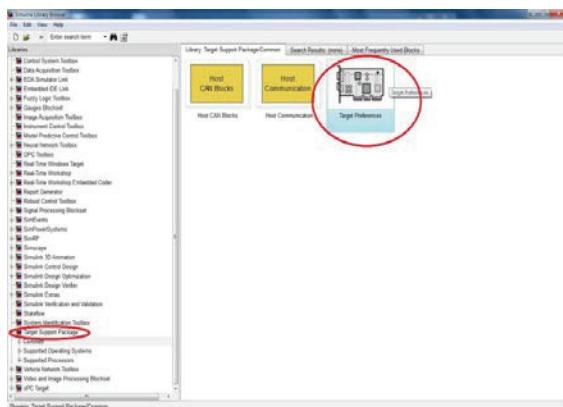
۴-۴- قسمت چهارم: ایجاد پروژه کامل از MATLAB برای CCS (اختیاری)

اکنون در این قسمت می‌خواهیم یک پروژه کامل CCS برای اجرا بر روی پردازنده از درون MATLAB ایجاد کنیم. برای این منظور علاوه بر بلوکهای متداول Simulink، برای استفاده امکانات بورد C6713 DSK آن از بلوکهای موجود در کتابخانه Target DIP Switch و DAC, ADC نظیر Support Package استفاده می‌نماییم.

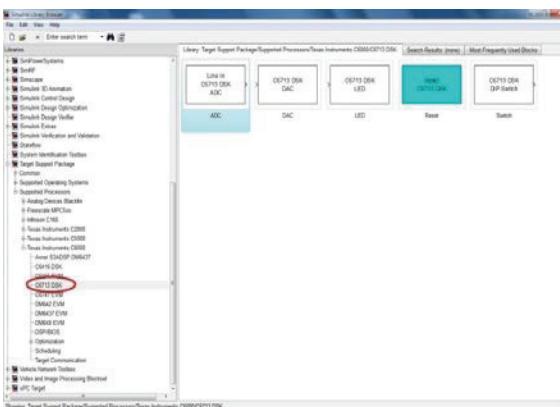
بلوک دیاگرام شکل ۹-۸ را در سیمولینک ایجاد نمایید. در این شکل از بلوکهای نرم افزار Target در کنار بلوکهای دیگر Simulink استفاده شده است. نرم افزار Target Support Package با بوردهای مختلف پردازش Support Package سیگنال از شرکتهای مختلف از جمله Texas Instrument دارد. برای ایجاد پروژه ای در CCS و اجرای آن بر روی بورد پردازنده ابتدا در محیط سیمولینک الگوریتم مورد نظر را پیاده نموده و تست می‌نماییم. همان طور که در شکل ۱۰-۸ دیده می‌شود بلوکهای ADC و DAC و Switch از کتابخانه Texas Instrument Target Support Package و زیر قسمت مربوط به بورد C6713 DSK از سری C6000 آورده شده‌اند. همچنین یک بلوک Target Preference نیز در صفحه مدل قرار می‌دهیم.



شکل ۸-۸ بلوک دیاگرام یک سیستم حذف نویز برای پیاده سازی بر روی پردازنده



شکل ۸-۹ بلوک Target Preference از کتابخانه Target Support Package



شکل ۹-۸ محل برخی بلوکهای موجود در کتابخانه Target C6713 DSK برای کار با بورد Support Package

در اینجا نویز به صورت دیجیتال به سیگنال اضافه می‌شود. برای نویز اضافه شونده واریانس ۱ در نظر بگیرید. همان طور که می‌بینید برای حذف نویز یک فیلتر قرار داده شده است. در هنگام شبیه سازی با قرار دادن بلوک **Oscope** در هر کجای بورد شکل سیگنال آن نقطه را تماشا کنیم. اما هنگامی که با بورد سر و کار داریم چنین امکاناتی نداریم! در اینجا از سوئیچ بورد برای مشاهده سیگنالهای مختلف استفاده شده است. در اینجا بلوک سوئیچ را از کتابخانه **Targ. Supp. Pack.** به مدل اضافه می‌کنیم. این بلوک برای مدل شبیه یک ورودی می‌باشد که وضعیت DIP Switch روی بورد را بر می‌گرداند.

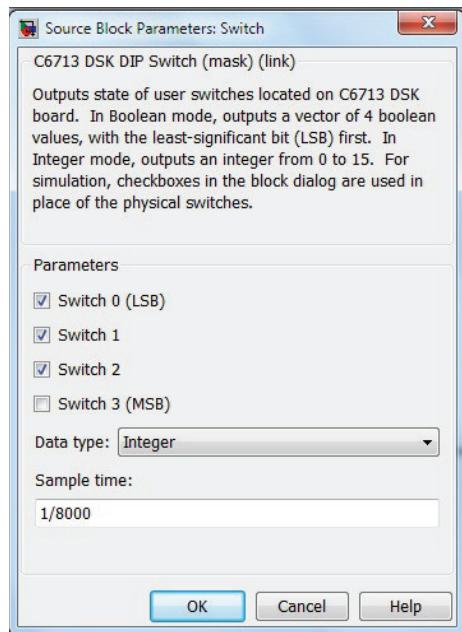
در بلوک **Multiport Switch** سیگنال بالایی سیگنال کنترلی می‌باشد. اگر مقدار سیگنال کنترلی ۰ باشد اولین سیگنال به خروجی راه می‌یابد. به همین ترتیب ۱ بودن سیگنال کنترلی دومین سیگنال

را به خروجی می‌فرستد و الى آخر.

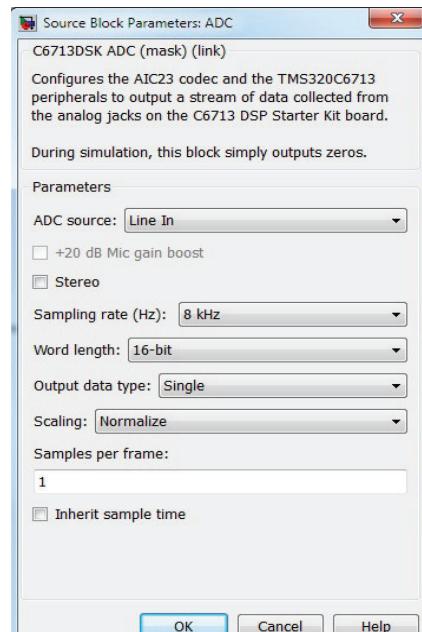
بعد از اطمینان از صحت عملکرد آن با انجام تنظیمات مربوطه در سیمولینک، پروژه را Build می‌کنیم. در اینجا از سیگنال ژنراتور به پورت Line In بورد C6713DSK سیگنال سینوسی با فرکانس 100HZ می‌دهیم. در برنامه نویز سفید گوسی با آن جمع می‌شود. همچنین با استفاده از یک فیلتر پایین گذر با فرکانس قطع 500Hz به حذف نویز می‌پردازند.

به منظور انجام تنظیمات بلوکها، برای بلوک ADC فرکانس نمونه برداری را 8000 Hz و پورت ورودی را Line In در نظر می‌گیریم. همچنین تعداد sample per frame را برابر ۱ قرار می‌دهیم به این ترتیب هر دفعه یک نمونه سیگنال وارد می‌شود. به این ترتیب صفحه تنظیمات بلوک ADC شبیه شکل ۱۲-۸ باید باشد.

از بلوک Switch برای کنترل سیگنالی که به پورت خروجی می‌رود استفاده می‌کنیم. در حقیقت این بلوک DIP Switch بورد می‌باشد. Sample Time Integer و Date type را نیز ۱/8000 انتخاب کنید. تنظیمات مربوط به آن در شکل ۱۳-۸ دیده می‌شود.



شکل ۱۲-۸ تنظیم پارامترهای بلوک سویچ برای بورد C6713 DSK



شکل ۱۳-۸ تنظیم پارامترهای بلوک ADC برای بورد C6713 DSK

با استفاده از ابزار Real-Time Workshop از مدل سیمولینک کد C ساخته می‌شود. همچنین می‌تواند به CCS وصل شود و کد تولید شده را برای build شدن توسط MATLAB به آن

بفرستد و بر روی پردازنده پیاده نماید.

انجام تنظیمات مربوط به شبیه سازی:

از منوی Configuration Parameters گزینه Simulation را انتخاب نمایید.

در قسمت solver تنظیمات زیر را اعمال نمایید :

```
stop time: inf
Discrete solver :Fixed-step
Fixed-step size (fundamental sample time): 1/8000
Tasking mode for periodic sample times: Single Tasking
```

در قسمت Target selection با استفاده از دکمه Browse برای Real-Time Workshop فایل idelink_grt.tlc را انتخاب نمایید. در قسمت Embedded IDE Link در قسمت Run Time برای build-and-execute گزینه Build action را انتخاب نمایید. همچنین stack size را برابر 8192 و برای compiler option setting عبارت -O2 را بنویسید.

سپس از منوی Tools گزینه Build Model را انتخاب نمایید. محیط workspace نرم افزار MATLAB وضعیت اجرای دستور را نشان می‌دهد، همچنین در نرم افزار CCS هم پیغامهای مربوطه نمایش داده می‌شوند. به این ترتیب یک پروژه در نرم افزار CCS ساخته می‌شود و build می‌شود و در پردازنده اجرا می‌شود.

به پورت Line In بورد از سیگنال ژنراتور سیگنال سینوسی با فرکانس 100 Hz وصل کنید و با استفاده از DIP Switch روی بورد سیگنال سینوسی بدون نویز، سیگنال سینوسی نویزی و سیگنال سینوسی فیلتر شده را مشاهده نمایید.

فرکانس سیگنال ورودی را 500Hz و 1000Hz تغییر دهید و با استفاده از DIP Switch سیگنالهای مختلف را مشاهده نمایید.

برای مشاهده پاسخ فرکانسی فیلتر وضعیت DIP Switch را برای عدد 1 تنظیم نمایید و سیگنال ورودی را نویز سفید بدهید و با استفاده از قابلیت نمایش طیف اسیلوسکوپ طیف سیگنال خروجی را مشاهده نمایید.

با استفاده از فایل Memory Map در نرم افزار CCS حجم کد تولید شده به صورت اتوماتیک توسط MATLAB را تعیین کنید و همچنین پروژه ایجاد شده را نیز بررسی نمایید.

۵-۸ ضمایم

لیست متودهای موجود برای شیئ ticcs برای ارتباط با نرم افزار CCS در جدول ۱-۸ آورده شده است. برای کسب اطلاع بیشتر از این متودها از راهنمای نرم افزار MATLAB استفاده نمایید.

جدول ۱-۸ لیست متودهای موجود برای شیئ ticcs برای ارتباط با نرم افزار CCS در MATLAB

Methods available for ticcs	Description
ACTIVATE	Set the active project, text file or build configuration
ADD	Add source file to a project
ANIMATE	Initiate a processor execution with breakpoint animation
ADDRESS	Search the processor's symbol table for an address
BUILD	Compile/Link to build a program file
CD	Change or query working directory of CCS
CLOSE	Close a CCS project
DIR	List files in CCS working directory
DISP	Display information about the TICCS object
HALT	Immediately terminate execution of the processor
INFO	Produce a list of information about the processor
INSERT	Insert a debug point into processor memory
ISREADABLE	Query if a block of processor memory is available for reading
ISRUNNING	Query status of processor execution
ISRTDXCAPABLE	Query if processor supports RTDX communications
ISVISIBLE	Query visibility of CCS application
ISWRITABLE	Query if a block of processor memory is available for writing
LIST	Produce various lists of information from CCS
LOAD	Load a program file into the processor
NEW	Create a default project or build configuration
OPEN	Open a project file
PROFILE	Return execution and stack profile report
READ	Return a block of data from the memory of the processor
REGREAD	Return data stored in a processor register
REGWRITE	Modify the contents of a processor register
RELOAD	Reload most recently loaded program file
REMOVE	Remove a file from a project or a debug point from memory
RESET	Reset the processor
RESTART	Return PC to the beginning of a program
RUN	Initiate execution of the processor
SAVE	Save a CCS project
SYMBOL	Return the processor's entire symbol table
VISIBLE	Hide or reveal CCS application window
WRITE	Place a block of MATLAB data into the memory of the processor

۹ ضمایم

۱-۹ لیست دستورات پردازنده‌های سری C6000

List of C6000 Instructions

.L

Instruction	Description
ABS	Integer absolute value with saturation
ADD(U)	Signed(unsigned) addition without saturation
AND	Bitwise AND
CMPEQ	Integer compare for equality
CMPGT	Signed integer compare for greater than
CMPGTU	Unsigned integer compare for greater than
CMPLT	Signed integer compare for less than
CMPLTU	Unsigned integer compare for less than
LMBD	Leftmost bit detection
MV	Move from register to register
NEG	Negate
NORM	Normalize integer
NOT	Bitwise NOT
OR	Bitwise OR
SADD	Integer addition with saturation to result size
SAT	Saturate a 40-bit Integer to a 32-bit Integer
SSUB	Integer subtraction with saturation to result size
SUB(U)	Signed (unsigned) integer subtraction without saturation
SUBC	Conditional integer subtract and shift – used for division
XOR	Exclusive OR
ZERO	Zero a register

.M

Instruction	Description
MPY	Signed integer multiply $16\text{lsb} \times 16\text{ lsb}$
MPYU	Unsigned integer multiply $16\text{lsb} \times 16\text{lsb}$
MPYUS	Integer multiply (unsigned) $16\text{lsb} \times (\text{signed}) 16\text{lsb}$
MPYSU	Integer multiply (signed) $16\text{lsb} \times (\text{unsigned}) 16\text{lsb}$
MPYH	Signed integer multiply $16\text{msb} \times 16\text{msb}$
MPYHU	Unsigned integer multiply $16\text{msb} \times 16\text{msb}$
MPYHUS	Integer multiply (unsigned) $16\text{msb} \times (\text{signed}) 16\text{msb}$
MPYHSU	Integer multiply (signed) $16\text{msb} \times (\text{unsigned}) 16\text{msb}$
MPYHL	Signed multiply high low $16\text{msb} \times 16\text{lsb}$

MPYHLU	Unsigned multiply high low $16\text{msb} \times 16\text{lsb}$
MPYHULS	Multiply high unsigned low signed (unsigned) $16\text{msb} \times (\text{signed}) 16\text{lsb}$
MPYHSLU	Multiply high signed low unsigned (signed) $16\text{msb} \times (\text{unsigned}) 16\text{lsb}$
MPY LH	Signed multiply low high $16\text{lsb} \times 16\text{msb}$
MPY LHU	Unsigned multiply low high $16\text{lsb} \times 16\text{msb}$
MPY LUHS	Multiply low unsigned high signed (unsigned) $16\text{lsb} \times (\text{signed}) 16\text{msb}$
MPY LSHU	Multiply low signed high unsigned (signed) $16\text{lsb} \times (\text{unsigned}) 16\text{msb}$
SMPY	Integer multiply with left shift and saturation
SMPYHL	Integer multiply high low with left shift and saturation
SMPYLH	Integer multiply low high with left shift and saturation
SMPYH	Integer multiply high with left shift and saturation

.S

Instruction	Description
ADD	Signed integer addition without saturation
ADDK	Integer addition using signed 16-bit constant
ADD2	Two 16-bit integer adds on upper and lower register halves
AND	Bitwise AND
B disp	Branch using a displacement
B IRP	Branch using an Interrupt return pointer
B NRP	Branch using a NMI return pointer
B reg	Branch using a register
CLR	Clear a bit field
EXT(U)	Extract and sign-extend(zero-extend) a bit field
MV	Move from register to register
MVC	Move between the control file and register file
MVK	Move a 16-bit signed constant into a register and sign extend
MVKH	Move 16-bit constant into the upper bits of a register
MVKLH	Move 16-bit constant into the lower bits of a register
NEG	Negate
NOT	Bitwise NOT
OR	Bitwise OR
SET	Set a bit field
SHL	Arithmetic shift left
SHR	Arithmetic shift right
SHRU	Logical shift right
SHRL	Logical shift left
SUB(U)	Signed (Unsigned) integer subtraction without saturation
SUB2	Two 16-bit Integer subtracts on upper and lower register halves

XOR	Exclusive OR
ZERO	Zero a register

.D

Instruction	Description	
ADD		Signed integer addition without saturation
ADDB/ADDAH/ADDAW		Integer addition using addressing mode
LDB(U)/LDH(U)/ LDW		Load from memory with a 5-bit unsigned constant offset or register offset
LDB(U)/LDH(U)/ (15-bit offset)	LDW	Load from memory with a 15-bit constant offset
MV		Move from register to register
STB/STH/STW		Store to memory with a register offset or 5-bit unsigned constant offset
STB/STH/STW offset)	(15-bit	Store to memory with a 15-bit offset
SUB		Signed integer subtraction without saturation
SUBAB/SUBAH/ SUBAW		Integer subtraction using addressing mode
ZERO		Zero a register

۲-۹ لیست دستورات اعشاری پردازنده‌های سری C67X

List of C67X Floating-Point Instructions

.L

Instruction	Description
ADDDP	Double-precision floating-point addition
ADDSP	Single-precision floating-point absolute value
DPINT	Convert double-precision floating-point value to integer
DPSP	Convert double-precision floating-point value to single-precision floating-point value
DPTRUNC	Convert double-precision floating-point value to integer with truncation
INTDP	Convert integer to double-precision floating-point value
INTDPU	Convert integer to double-precision floating-point value (unsigned)
INTSP	Convert integer to single-precision floating-point value
INTSPU	Convert integer to single-precision floating-point value (unsigned)

SPINT	Convert single-precision floating-point value to integer
SPTRUNC	Convert single-precision floating-point value to integer with truncation
SUBSP	Single-precision floating-point subtract
SUBDP	Double-precision floating-point subtract

.M

Instruction	Description
MPYSP	Single-precision floating-point multiply
MPYDP	Double-precision floating-point multiply
MPYI	32-bit integer multiply - result in lower 32 bits
MPYID	32-bit integer multiply - 64-bit result

.S

Instruction	Description
ABSSP	Single-precision floating-point absolute value
ABSDP	Double-precision floating-point absolute value
CMPGTSP	Single-precision floating-point compare for greater than
CMPEQSP	Single-precision floating-point compare for equality
CMPLTSP	Single-precision floating-point compare for less than
CMPGTD P	Double-precision floating-point compare for greater than
CMPEQDP	Double-precision floating-point compare for equality
CMPLTDP	Double-precision floating-point compare for less than
RCPSP	Single-precision floating-point reciprocal approximation
RCPDP	Double-precision floating-point reciprocal approximation
RSQRSP	Single-precision floating-point square-root reciprocal approximation
RSQRDP	Double-precision floating-point square-root reciprocal approximation
SPDP	Convert Single precision floating-point value to double-precision floating-point Value

.D

Instruction	Description
ADDAD	Integer addition using doubleword addressing mode
LD DW	Load doubleword from memory with an offset

۱۰ مراجع

- TMS320C67x DSP Library Programmer's Reference Guide (spru675c.pdf) [۱]
TMS320C6000 Code Composer Studio Tutorial (spru310c.pdf) [۲]
Code Composer Studio, User's Guide (spru328B.pdf) [۳]
TMS320C6000 CPU and Instruction Set Reference Guide (spru189f.pdf) [۴]
TMS320C67x C67x+ DSP CPU and Instruction Set reference Guide (spru733.pdf) [۵]
Texas Instruments, Stereo Audio D/A Convertor, 8 to 96 KHz, with Integrated [۶]
Headphone Amplifier, (slws106G.pdf)
TMS320C6000 DSP Peripherals Reference Guide (spru190.pdf) [۷]
TMS320C6000 DSP Multichannel Buffered Serial (McBSP) Refernce Guide [۸]
(spru580.pdf)
TMS320C6000 Chip Support Library API Library Reference Guide (spru401.pdf) [۹]
TMS320C6000 Assembly Language Tools User's Guide (spru186.pdf) [۱۰]
R. Chassaing and D. Reay, Digital Signal Processing and Applications with the [۱۱]
TMS320C6713 and TMS320C6416 DSK, 2nd Ed., Wiley, 2008
S. M. Kuo, B. H. Lee and W. Tian, Real-Time Digital Signal Processing [۱۲]
Implementation and Applications, 2nd Ed., Wiley, 2006
S. A. Tretter, Communication System Design Using DSP Algorithms with [۱۳]
Laboratory Experiments for TMS320C6713 DSK, Springer, 2008
N. Kehtarnavaz, Real-Time Digital Signal Processing Based on the [۱۴]
TMS320C6000, Newnes, 2005
[۱۵] شفاعی، کیانوش؛ مرجع کامل پردازنده های DSP سری های 2000 ، 5000 و 6000 ؛ موسسه
چاپ و انتشارات آستان قدس؛ ۱۳۸۹
A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, Third Ed., [۱۶]
Prentice Hall, 2010
J. G. Proakis and D. G. Manolakis, Digital Signal Processing Principles, [۱۷]
Algorithms, and Applications, 4th Ed. , Prentice Hall, 2007
Embedded IDE link 4 User's Guide, Mathworks, 2010 [۱۸]
Embedded IDE link 4 User's Guide, For Use with Texas Instruments' Code [۱۹]
Composer Studio, Mathworks, 2010
Target Support Package 4 User's Guide, For Use with Texas Instruments C6000 [۲۰]