

بخش دوم – دستورکار

پیشگفتار:

هدف از آزمایشگاه DSP که از آغاز پاییز سال تحصیلی ۱۳۸۶ برای نخستین بار در دانشکده برق دانشگاه صنعتی شریف تحت سرپرستی جناب آقای دکتر بابایی زاده، آغاز به کار نموده است، آشنایی با جنبه های عملی و کاربردی مفاهیم درس DSP می باشد. دانشجویان می توانند در این آزمایشگاه، با پیاده سازی real-time برخی مفاهیم درس DSP در قالب آزمایش های مختلف، این مفاهیم و اصول را هرچه عمیق تر و بهتر درک کنند.

آزمایش های این درس بر اساس پروسسور DSP ، TMS320C6414 طراحی شده اند. این پروسسور توسط شرکت Texas Instruments برای برآورده کردن نیازهای سرعت و کارایی بالا، در کاربردهای پردازش سیگنال ساخته شده است. لازم به ذکر است که آنچه در این آزمایشگاه مورد استفاده خواهد گرفت، برد آموزشی ساخته شده توسط شرکت TI بر اساس همین تراشه ی TMS320C6416 می باشد.

این برد، DSK6416 یا C6416 DSP Starter Kit نامیده می شود و زمینه ی مناسبی را برای دانشجویان به منظور انجام آزمایش ها و یادگیری هرچه بیشتر کار با این گونه پروسسورهای DSP فراهم می آورد. نرم افزار لازم برای پروگرام کردن و راه اندازی این برد، در محیطی به نام Code Composer Studio یا CCS فراهم می گردد که در این درس با تأکید بر زبان برنامه نویسی C به دانشجویان آموزش داده خواهد شد.

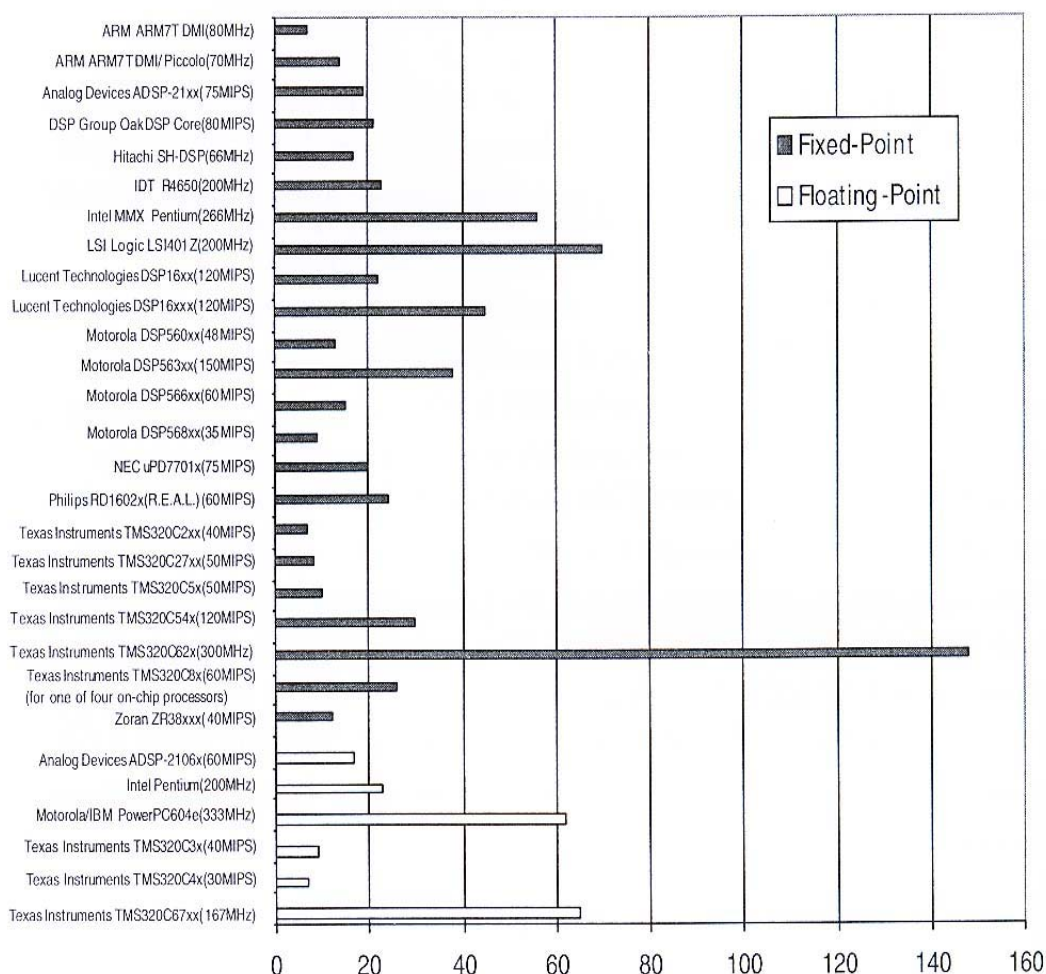
برنامه ی آزمایشگاه مشتمل بر ۸ جلسه ی آموزشی و عملی می باشد و امید است که مورد توجه و استفاده دانشجویان قرار بگیرد. در نهایت خواهشمندیم که با مشاهده ی هرگونه کم و کاست در دستور کار، مورد را به ما گوشزد نمایید تا در جهت رفع آن اقدام شود.

با تشکر

مقدمه:

ابتدا، جهت آشنایی بیشتر با برد و پروسسوری که قرار است تمام آزمایش ها بر اساس آن پیاده سازی شود، به بررسی اجمالی ساختار کلی و اجزای سازنده برد و برخی خصوصیات آن می پردازیم. پروسسورهای پردازش سیگنال های دیجیتال خانواده ی TMS320C6X مانند TMS320C6416 ، از پروسسورهای پر سرعت ساخت شرکت TI می باشند که در کاربردهای مختلف DSP که نیاز به سرعت و کارایی بالا دارند، مورد استفاده قرار می گیرند. این پروسسورها اصولاً در کاربردهای real-time که قرار است پردازش همگام با پیشامد خارجی انجام بگیرد به کار می روند و گستره ی وسیعی از کاربردها مانند ارتباطات و پردازش صحبت و پردازش تصویر را پوشش می دهند. عضوی از این خانواده که در این برنامه مورد توجه و استفاده قرار می گیرد، پروسسوری از نوع Fixed point (مفهوم پروسسورهای fixed point و تفاوت های آنها با پروسسور های Floating point ، در جلسه ی چهارم دستورکار به تفصیل بررسی خواهد شد.) می باشد که یکی از پر قدرت ترین تراشه های موجود امروزی محسوب می گردد و قادر است بالغ بر ۵۰۰۰ مگا دستورالعمل را در ثانیه اجرا کند. برای درک قدرت پردازش این پروسسور، به نمودار شکل ۲ که پروسسورهای این خانواده را با دیگر پروسسورهای DSP متداول بر اساس benchmark استاندارد تشخیص سرعت پردازشگرهای سیگنال مقایسه می کند، توجه کنید.

در این قسمت، در دو بخش کلی، ابتدا به بررسی ساختار داخلی و اجزای تشکیل دهنده ی درونی قلب برد یا همان تراشه ی TMS320C6416 می پردازیم و سپس با بررسی ساختار برد آموزشی به چگونگی ارتباط این تراشه با وسایل جانبی دیگر تعبیه شده روی برد، پی می بریم.

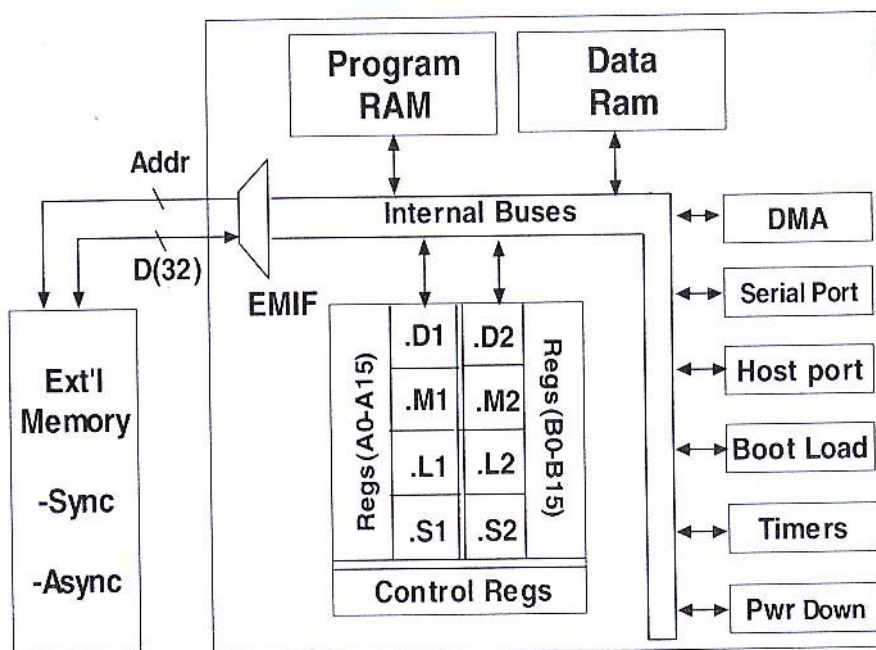


شکل ۲ - مقایسه ی پردازشگرهای متدوال DSP با معیار استاندارد سرعت و کارایی

• بخش اول - پروسسور TMS320C6416

شکل ۱، بلوک دیاگرام کلی پروسسورهای خانواده C6X را که تراشه ی ما نیز عضوی از آن است، نشان می دهد. همان طور که ملاحظه می شود، علاوه بر حافظه های RAM و ROM و تایمرها و دیگر اجزای داخلی، این پروسسور دارای هشت واحد عملیاتی می باشد که به دو قسمت مجزای A و B تقسیم شده اند. هر قسمت، دارای یک واحد M. (که برای انجام عملیات ضرب در این پروسسور به کار می رود)، یک واحد L. (که برای انجام عملیات منطقی و حسابی در این پروسسور به کار می رود)، یک واحد S. (که برای انجام عملیات حسابی روی بیت ها به کار می رود) و در نهایت یک واحد D. (که به منظور انجام عملیات load و store و عملیات حسابی در پروسسور به کار می رود) می باشد. ضمناً هر

قسمت دارای ۱۶ رجیستر ۳۲ بیتی می باشد. با همکاری و تقسیم وظایف CPU بین این دو قسمت و واحدهای عملیاتی مربوط به آنها، می توان به کارایی بالاتر و سرعت بیشتر در انجام دستورالعمل ها رسید.



شکل ۱- بلوک دیاگرام کلی پروسسورهای خانواده C6X

گذرگاه داخلی (Internal bus) این خانواده از پروسسورها، از یک آدرس باس ۳۲ بیتی و یک دیتا باس ۲۵۶ بیتی که خود شامل هشت باس ۳۲ بیتی برای انتقال دستورالعمل ها به هشت واحد عملیاتی ذکر شده می باشد تشکیل شده است. ارتباط حافظه های خارجی با این پروسسور از طریق یک باس خارجی شامل یک آدرس باس ۲۰ بیتی و یک دیتا باس ۳۲ بیتی انجام می گردد.

همانطور که از روی بلوک دیاگرام نیز قابل تشخیص است، ادوات جانبی (Peripherals) مربوط به یک پروسسور خانواده C6X، می تواند شامل حافظه های خارجی، تراشه های مربوط به عملیات DMA (Direct Memory Access) - که به منظور دستیابی به داده های درون حافظه های داخلی یا خارجی بدون دخالت مستقیم CPU برای افزایش سرعت به کار می روند-، تایمرها، پورتهای مربوط به برقراری ارتباط سریال با تراشه های دیگر-که (Multichannel Buffered Serial Port (McBSP نامیده می شوند و امکان برقراری یک ارتباط سریال پر سرعت را فراهم می کنند- و همچنین پورت های

Boot Loader - که برای لود کردن کدها از حافظه های خارجی به کار می روند- و نیز واحد های Power down - که به طور اتوماتیک هرگاه CPU غیر فعال است Power را به منظور صرفه جویی قطع می کنند- باشد.

نکته ی دیگری که راجع به این نوع CPU ها قابل ذکر است، قابلیت منحصر به فرد Pipelining آنها در اجرای دستورالعمل ها می باشد که در ادامه به توضیح آن می پردازیم. به طور کلی برای اجرای کامل یک دستورالعمل در چند مرحله صورت می گیرد. این مراحل شامل آوردن دستورالعمل از حافظه به روی باس (Fetch) و دیکود کردن آن (Decode) و در نهایت اجرای آن (Execute) می باشد. اگر این مراحل پشت سرهم و به طور متوالی انجام بگیرند، امکان استفاده از تمامی قابلیت های این پروسسور های توانمند، از جمله همه ی هشت واحد عملیاتی تعبیه شده روی آن، فراهم نمی گردد. به منظور افزایش کارایی و سرعت، CPU های این خانواده از تکنیکی به نام Pipelining استفاده می کنند. برای درک بهتر چگونگی انجام دستورالعملها در این تکنیک و تفاوت آن با شیوه ی عادی اجرای دستورالعمل ها به شکل زیر توجه نمایید. این شکل مراحل مختلف اجرای سه دستورالعمل را در دو روش عادی و Pipelining نشان می دهد. همانطور که دیده می شود، اجرای این دستورالعملها در روش دوم نیاز به clock cycle های کمتری دارد.

CPU Type	Clock Cycles								
	1	2	3	4	5	6	7	8	9
Non-Pipelined	F ₁	D ₁	E ₁	F ₂	D ₂	E ₂	F ₃	D ₃	E ₃
Pipelined	F ₁	D ₁ F ₂	E ₁ D ₂ F ₃	E ₂ D ₃	E ₃				

F_x = fetching of instruction x
 D_x = decoding of instruction x
 E_x = execution of instruction x

شکل ۳- روش عادی اجرای دستورها در مقایسه با روش Pipelining

در این شکل، هر مرحله از سه مرحله ی اجرای دستورها، که با حروف F و D و E (که نماینده ی کلمات Fetch و Decode و Execute می باشند) و اندیس های مشخص کننده اینکه هریک از این مراحل مربوط به کدام دستورالعمل می باشد، مشخص شده اند. نیاز به یک clock cycle دارند. همانطور که دیده می شود در روش pipelining هنگامی که CPU مشغول دیکود کردن یک دستورالعمل است و در نتیجه آدرس باس آن مورد نیاز نیست، به طور همزمان، آدرس باس برای Fetch

کردن دستورالعمل بعدی از حافظه به کار می رود. و به همین ترتیب با انجام همزمان مراحل مختلف در clock cycle ها، انجام سه دستورالعمل به جای نه clock cycle به طور بهینه در پنج clock cycle انجام پذیرفته است.

• بخش دوم - برد آموزشی DSK6416

در این قسمت با ساختار برد آموزشی مورد استفاده در این آزمایشگاه که از تراشه ی TMS320C6416 به عنوان پروسسور اصلی خود استفاده می کند، آشنا می شویم.

هر برد DSK یک سیستم DSP کامل محسوب می شود. این برد محیط مناسبی را جهت انجام آزمایشهای مربوط به پردازش سیگنال و آشنایی با کاربردهای عملی مفاهیم DSP فراهم می کند. هر برد به طور کلی شامل یک پروسسور TMS320C6416 و نیز یک codec ۳۲ بیتی به نام TLV320AIC23 یا به طور خلاصه یک AIC23 codec برای انجام عملیات A/D و D/A روی ورودی و خروجی می باشد. این codec به یک کلاک ۱۲ مگا هرتزی متصل است و امکان نمونه برداری از سیگنال ها را با نرخ های متفاوتی بین ۸ تا ۹۶ کیلو هرتز فراهم می آورد. (در حقیقت امکان نمونه برداری با تمام نرخ های بین ۸ تا ۹۶ کیلو هرتز توسط این codec وجود ندارد. بلکه نرخ های نمونه برداری مجاز توسط این برد به شرح زیرست:

Sampling rate (kHz)	
8.000	44.000
16.000	48.000
24.000	96.000
32.000	

شکل ۵ - لیست نرخ های مجاز نمونه برداری توسط AIC23

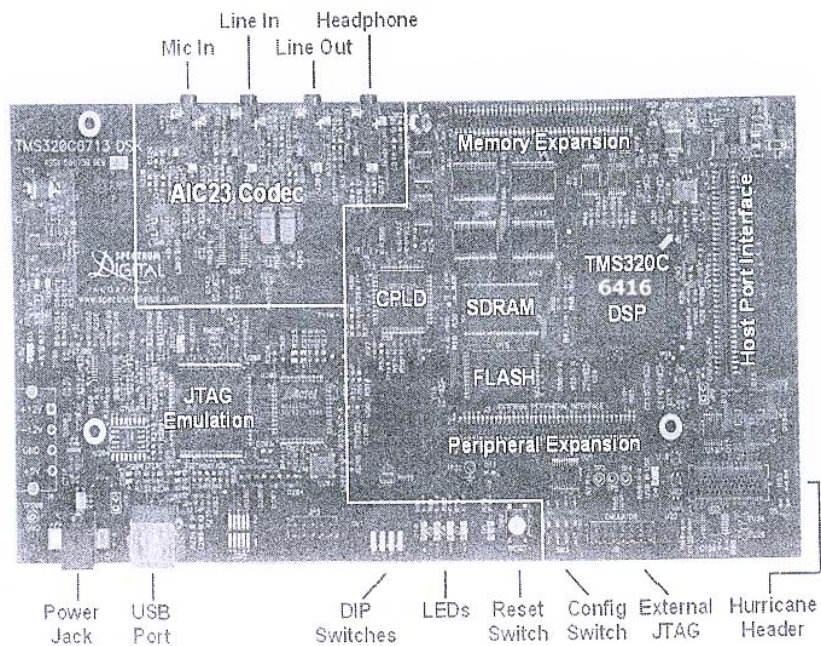
اگر بخواهیم با نرخ های دیگری به جز این نرخ ها از سیگنال نمونه برداری کنیم، باید توسط کانکتورهای موجود روی برد، یک Daughter Card به این منظور به برد اضافه کنیم که بحث در مورد این موضوع در حوزه ی مباحث این آزمایشگاه نمی گنجد.

برد شامل 16MB حافظه RAM از نوع Synchronous Dynamic Random Access یا SDRAM و همچنین 256KB حافظه ی Flash Memory می باشد. چهار کانکتور روی برد که از نوع کانکتور های استاندارد 3.5mm می باشند، ورودی ها و خروجی های برد را تشکیل می دهند. کانکتور MIC IN برای ورودی میکروفون، کانکتور LINE IN برای ورودی LINE، کانکتور LINE OUT برای خروجی LINE و در نهایت کانکتور HEADPHONE برای خروجی هدفون به کار می رود. این برد همچنین دارای چهار عدد LED و چهار عدد DIP SWITCH می باشد که علاوه بر اینکه با نشان دادن Status برد، برای تست کارکرد برد می توانند مورد استفاده قرار بگیرند، کاربردهای دیگری نیز در آزمایش های مختلف دارند که در طول جلسات آینده، توضیح داده خواهند شد.

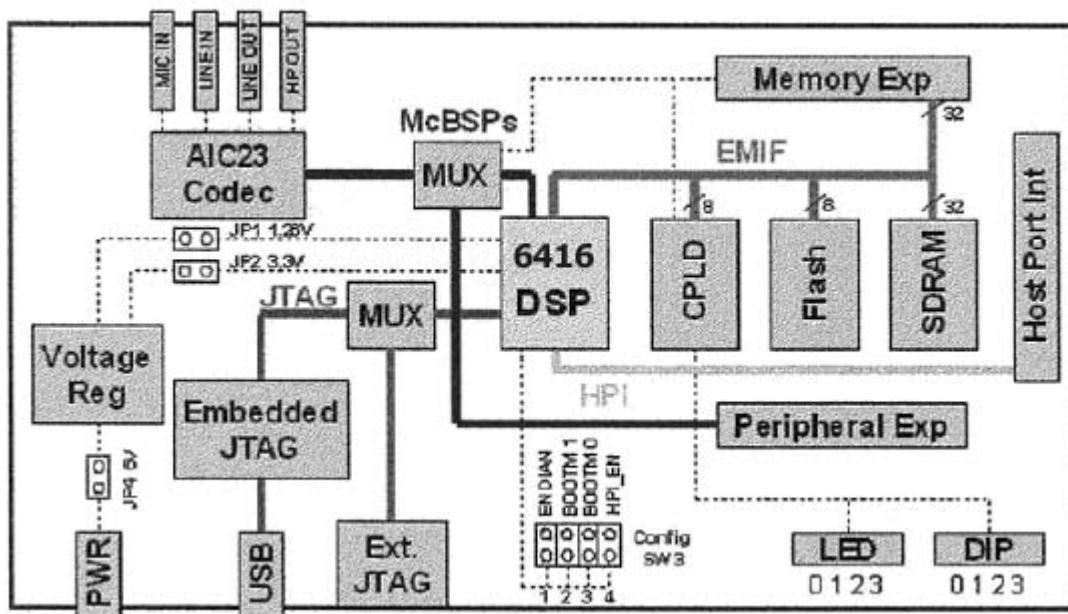
این برد دارای کابل JTAG می باشد که از قبل در ساختار درونی آن Embed شده است و در نتیجه ارتباط برد با کامپیوتری که قرار است جهت پروگرام کردن آن به کار رود، به آسانی توسط یک کابل USB انجام می گیرد. گرچه یک کابل JTAG جدا نیز برای کاربردهایی که در آنها استفاده از این نوع کابل مورد نیاز است، وجود دارد. سیستم دارای یک رگولاتور داخلی ولتاژ نیز می باشد که ولتاژ ثابت 1.6V یا 2.5V را برای کار CPU و ولتاژ 3.3V را برای کارکرد حافظه ها و وسایل جانبی آن، فراهم می آورد. نکته ی بسیار مهمی که باید در این جا به آن دقت نمود این است که:

برای راه اندازی برد باید حتماً ابتدا کابل USB آن را به کامپیوتر متصل کنید و سپس برد را به برق متصل نمایید. اگر این عملیات به ترتیب ذکر شده انجام نگیرند، می توانند سبب صدمه زدن یا حتی سوزاندن برد و تجهیزات آزمایشگاه گردد.

شکل های زیر ساختار داخلی برد را هم از نمای واقعی و هم توسط یک بلوک دیاگرام نشان می دهد. خلاصه ی آن چه در این بخش توضیح داده شد، در این شکل ها قابل مشاهده است.



شکل ۶- الف

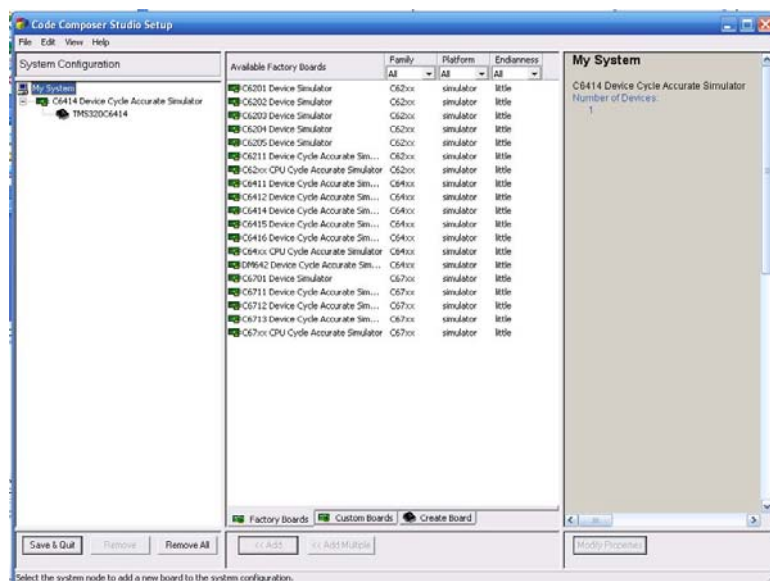


شکل ۶- ب

شکل ۶- برد DSK الف-نمای واقعی و ب-بلوک دیاگرام

جلسه اول:

- عنوان: آشنایی اولیه با جنبه های نرم افزاری برد
- هدف: آشنایی با چگونگی تست کردن و اطمینان از صحت عملکرد برد پیش از راه اندازی آن، آشنایی با محیط برنامه Code Composer Studio (CCS) و یادآوری مبانی برنامه نویسی به زبان C با بررسی برنامه ی آماده ی ولوم دادن به یک سیگنال ورودی و شبیه سازی این برنامه توسط Simulator برد.
- شرح آزمایش: ابتدا از روی CD که در اختیار شما قرار گرفته است، برنامه ی Code Composer Studio 3.1 را بر روی کامپیوتر خود نصب کنید. سپس از روی همان CD درایورهای برد را نیز install نمایید. در این هنگام icon های CCstudio 3.1 و نیز Setup CCstudio v3.1 باید روی DESKTOP شما ظاهر شده باشند. سپس از اجرای CCstudio برنامه ی Setup آنرا اجرا کنید لازم است توسط این Setup برد یا تراشه ی خود را به CCstudio بشناسانید. در این جلسه ما به سیمولاتور برد احتیاج داریم تا بتوانیم پس از آشنایی با محیط CCStudio برنامه ی ساده ای را در این محیط شبیه سازی کنیم پس از بین گزینه های موجود در سمت راست صفحه ی باز شده، گزینه ی C6416 Device Cycle Accurate Simulator را انتخاب نمایید. اکنون باید صفحه ی شما مانند شکل ۱ باشد. با انتخاب گزینه ی Save & Quit برنامه Setup را ترک کنید. از این پس برنامه CCstudio به عنوان سیمولاتوری مطابق با برد شما عمل خواهد کرد.



شکل ۱- پنجره ی Setup پس از انتخاب گزینه ی سیمولاتور مربوطه

پیش از پرداختن به جنبه های نرم افزاری کار با برد DSK6416، ابتدا لازم است در این قسمت، با چگونگی تست کردن برد آشنا شویم تا قبل از نوشتن هرگونه برنامه و تلاش برای اجرای آن، از سالم بودن اتصالات و تراشه های روی برد به طریقی اطمینان حاصل کنیم. اولین نکته ای که بدین منظور می تواند به ما کمک کند، این است که به محض وصل کردن برد به برق، به طور خودکار، برنامه ای به نام Post.c یا Power on, self test که در Flash Memory برد ذخیره گشته است، شروع به اجرا شدن می کند. این برنامه، حافظه های داخلی و خارجی و دو پورت سریال McBSP و تراشه های DMA و codec روی برد و همچنین LED های آنرا چک می کند. در صورتیکه تمامی این تست ها با موفقیت انجام گردد، تمام LED ها به طور همزمان سه بار چشمک خواهند زد و سپس در حالت روشن باقی خواهند ماند. در طول تست کردن codec برد توسط این برنامه نیز، یک سیگنال سینوسی با فرکانس 1KHZ به مدت یک ثانیه روی خروجی های LINE OUT و HEADPHONE تولید می شود که در صورت متصل بودن به بلندگو قابل شنیدن است.

اما، به جز این تست که با هربار وصل کردن برد به برق به طور خودکار انجام می پذیرد، دو روش دیگر نیز برای تست کردن برد در سطح پیشرفته تر وجود دارد که در ادامه به شرح آنها می پردازیم:

الف- استفاده از برنامه ی 6416DSK Diagnostics Utility :

این برنامه همزمان با install کردن کامل نرم افزار CCS و درایورهای برد، خود به خود روی DESKTOP نصب می گردد پس از متصل کردن برد به کامپیوتر و به برق زدن آن، با اجرا کردن این برنامه، می توان برد را در دو قسمت General و Advanced تست کرد. برای انجام هریک از این تست ها، کافست بر روی Tab مورد نظر رفته و دکمه ی Start را فشار دهید. این تست ها به طور خود به خود، شروع به انجام شدن می کند و در نهایت اگر هر قسمتی دچار مشکلی بود یا تست آن با موفقیت انجام نشد، پیغام خطا به کاربر نشان داده می شود. با این نرم افزار می توان تقریباً تمام اجزای مهم برد را تست نمود.

ب- استفاده از امکانات برنامه ی Code Composer :

توجه کنید که برای اجرا کردن این تست باید از پنجره ی Setup، برنامه ی CCStudio را باید از حالت سیمولاتور خارج نموده و این بار گزینه ی DSK V1.1 6416 را انتخاب نمایید. با اجرا کردن برنامه ی Code Composer، از منوی بالای آن، گزینه های

Quick Test → Check DSK → GEL را انتخاب نمایید. در این هنگام در صورت سالم بودن برد، پیغام های زیر به شما نمایش داده خواهد شد،

Switches: 15

Board Revision: 1

CPLD Revision: 2

این پیغام با این فرض نمایش داده می شود که هر چهار DIP SWITCH برد در حالت بالا باشند. با تغییر وضعیت این سوئیچ ها و اجرای دوباره ی همین تست، باید عدد معادل وضعیت سوئیچ ها به درستی نمایش داده شوند. مثلاً اگر وضعیت سوئیچ ها در حالت 0111 باشد، باید پیغام Switches:7 ظاهر شود.

حال که با چگونگی تست کردن برد به شیوه های مختلف آشنا شدیم، از درون Setup دوباره برنامه ی CCStudio را به حالت سیمولاتور C6416 ببرید و برنامه را اجرا کنید.

به طور کلی پروگرام کردن پروسسورهای DSP می تواند به زبان C یا به زبان اسمبلی صورت گیرد. در این آزمایشگاه با توجه به دشوار بودن پروگرام کردن برد با زبان اسمبلی به دلیل تعداد دستورالعمل های زیادی که باید بدین منظور با آنها آشنا شویم، تمرکز بر روی برنامه نویسی و پیاده کردن الگوریتم ها به زبان C می باشد. برنامه ی Code Composer محیط مناسبی را برای نوشتن کدها و پروگرام کردن برد به ما ارائه می دهد و همچنین امکان تحلیل و اجرای برنامه ها به صورت real-time را برای ما فراهم می کند.

هر الگوریتمی که قرار است با زبان برنامه نویسی C پیاده شود و سپس توسط برنامه ی CCStudio به یک فایل Executable قابل اجرا روی برد تبدیل گردد، باید درون یک Project و در کنار فایل های ضروری دیگر قرار بگیرد. به طور کلی، هر Project یا فایل .pjt شامل چند نوع فایل به شرح زیر می باشد:

۱- فایل های .lib: این فایل های آماده که باید به پروژه ضمیمه شوند، توابع لازمی که در هنگام برنامه نویسی مورد استفاده قرار خواهد گرفت، برای برد مورد نظر فراهم می کنند.

۲- فایل های .c: این فایل ها که به زبان C نوشته می شوند Source File های اصلی پروژه هستند که کد اصلی پروژه در آن نوشته می شود.

۳- فایل های h: : این نوع فایل ها نیز برای تعریف متغیرهای کلی و مورد نیاز در پروژه به کار می روند و باید در پروژه ضمیمه شوند.

۴- فایل های asm: در صورت نیاز، برخی فایل ها به زبان اسمبلی نوشته می شوند و به پروژه ضمیمه می گردند.

۵- فایل های cmd: : این نوع فایل ها که Linker Command Files نامیده می شوند، برای مپ کردن قسمت های مختلف حافظه به کار میرود.

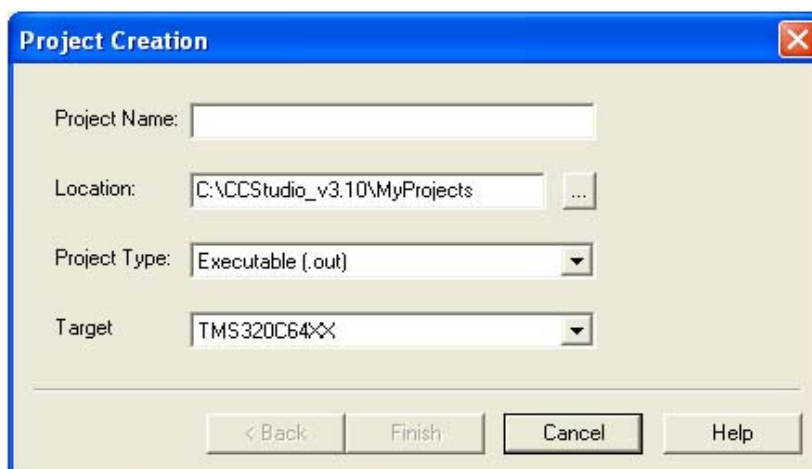
در این قسمت از آزمایش، پروژه ی آماده ای که یک عملیات ساده ی پردازش سیگنال یعنی ولوم دادن به یک سیگنال ورودی را انجام می دهد، قدم به قدم بررسی خواهیم کرد و از این راه با مراحل مختلف ساخت و پیاده سازی یک پروژه آشنا خواهیم شد. توجه کنید که هدف از این آزمایش برنامه نویسی یا پیاده کردن الگوریتم با کد نمی باشد. بلکه تمام این فایل ها به صورت آماده در اختیار شما قرار خواهد گرفت. به چگونگی کار با یک پروژه و عملیاتی که باید مرحله به مرحله انجام شوند دقت کنید.

قبل از انجام این کار، در آدرس C:\CCStudio_v3.10\myprojects یک Folder جدید به نام Volume1 بسازید و محتویات Folder ای به نام Volume که توسط مسئول آزمایشگاه روی Desktop شما قرار گرفته است را درون آن کپی کنید. حال مراحل آشنایی با نرم افزار Code Composer را در قسمت های زیر دنبال می کنیم.

● بخش اول – ساختن پروژه و افزودن فایل ها به آن

ابتدا لازم است به این منظور، پروژه ی جدیدی در محیط CCStudio ایجاد کنید. پس از اجرای برنامه ی CCStudio ، از منوی Project بالای صفحه، گزینه ی New را انتخاب نمایید. در پنجره ای که پیش روی شما باز می شود، مطابق شکل ۲ ، نام پروژه را Volume1 تایپ کنید.

در قسمت Location، همان Folder ای که خودتان پیش از این ساختید را انتخاب کنید. نوع پروژه را Executable انتخاب کنید تا فایل خروجی پروژه شما یک فایل out. و قابل اجرا شدن باشد. نوع برد خود را نیز در قسمت target انتخاب کرده و دکمه ی Finish را بزنید.



شکل ۲ - پنجره ی ساخت پروژه ی جدید

پروژه ی شما ساخته شد. حال لازم است فایل های مورد نیاز را به این پروژه ضمیمه کنید. از منوی Project گزینه ی Add Files to Project را انتخاب کنید. از درون Folder ای که خودتان ساخته بودید، فایل volume.c را انتخاب کرده و به پروژه اضافه کنید. راه دیگر اضافه کردن فایل ها به پروژه right-click کردن روی icon پروژه در پنجره ی سمت راست صفحه و انتخاب گزینه ی Add Files to Project می باشد.

فایلی که هم اکنون به پروژه ی شما اضافه شد، حاوی کد اصلی پروژه می باشد. درون این کد، یک آرایه برای ورودی و یکی برای خروجی تعریف شده است. ابتدا پیغام "Volume Example Started" روی صفحه نمایش داده می شود. سپس درون یک حلقه، دائماً تابعی به نام DataIO فراخوانده می شود که البته این تابع کاری انجام نمی دهد زیرا در این پروژه در راستای اهداف آموزشی، عملیات خواندن ورودی از یک فایل آماده، توسط یک Probe که در ادامه توضیح داده خواهد شد انجام می گردد. سپس تمامی نمونه های آرایه ی ورودی در یک عدد ثابت ضرب شده و در آرایه خروجی ذخیره می گردند.

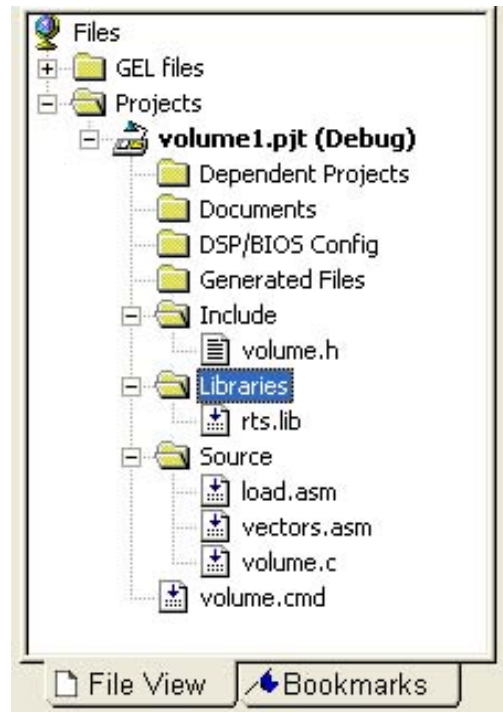
هنوز لازمست فایل های دیگری به پروژه اضافه شود. پس دوباره گزینه ی Add Files to Project را انتخاب کنید و این بار در جعبه ی Files of Type (گزینه ی Asm Source Files) را انتخاب کنید. دو فایل vectors.asm و load.asm از Folder خود را به پروژه اضافه کنید. این دو فایل که به زبان اسمبلی نوشته شده اند، برخی عملیات مربوط به فعال کردن وقفه ی reset و عملیاتی که باید در این وقفه انجام شود تا برنامه به مسیر درست خود هدایت شود، را به عهده دارند.

اکنون، از درون پنجره ی Add Files to Project گزینه ی Linker Command Files یا فایل های .cmd و .lcf را انتخاب کنید. فایل volume.cmd را به پروژه اضافه کنید. این فایل قسمت های مختلف memory را مپ می کند. در آزمایش های بعدی با طریقه نوشتن این فایل ها آشنا خواهید شد.

حال باید فایل های .o و .lib را به پروژه اضافه کنید. به آدرس زیر بروید:
C:\CCStudio_v3.10\c6000\cgtools\lib
در جعبه ی Files of Type گزینه ی Object and Library Files یا فایل های .o و .lib را انتخاب کنید. فایل rst6400.lib را به پروژه اضافه کنید. این فایل باید در تمام پروژه ها اضافه گردد زیرا توابع اولیه و ضروری را برای کار برد فراهم می کند.

حال در پنجره ی Project View Window در سمت چپ صفحه ،روی icon پروژه یعنی volume1.pjt کلیک کنید و گزینه ی Scan All File Dependencies را انتخاب کنید. با این کار هر Header File که در کد ضمیمه شده باشد، به طور اتوماتیک شناسایی شده و به پروژه اضافه می گردد. در این مورد فایل Volume.h به پروژه افزوده خواهد گشت.

در پایان عملیات افزودن فایل ها به پروژه، باید پنجره ی Project View Window به صورت زیر در آمده باشد:



شکل ۳

اگر نیاز داشتید فایلی را از پروژه ای حذف کنید، با right-click کردن روی آن در صفحه‌ی Project View Window و انتخاب گزینه Remove from project می توانید این کار را انجام دهید.

● بخش دوم – کامپایل کردن و اجرای پروژه

در این قسمت، می خواهیم مراحل کامپایل کردن و در نهایت اجرای پروژه را پیگیری کنیم. بدین منظور، از منوی Project گزینه ی Rebuild All را انتخاب کنید. این گزینه برنامه را Compile و Assemble و Link خواهد کرد. پنجره ی Build در پایین صفحه پیغام های هشدار یا خطایی که در حین این کار پیش خواهد آمد را نشان می دهد.

به طور پیش فرض، فایل out. ایجاد شده از پروژه، در Folder ای به نام Debug که درون Folder پروژه ی شما می باشد، قرار می گیرد. شما می توانید این مسیر را به وسیله ی toolbar بالای صفحه به شکل زیر تغییر دهید.



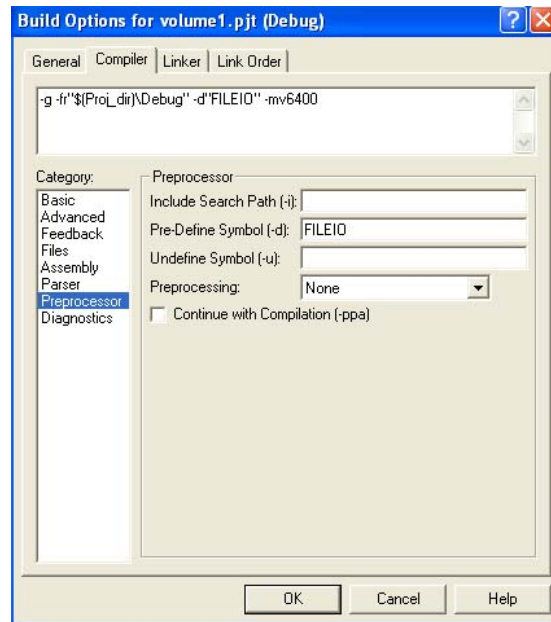
شکل ۴ – toolbar مربوط به تغییر مکان ذخیره ی فایل out.

حال از منوی File، گزینه ی Load Program را انتخاب کنید و فایل out. که از پروژه ی شما ایجاد گردیده است را باز کنید. با این کار یک فایل Disassembly از پروژه ی شما، روی صفحه ظاهر می شود. برای اجرای این فایل، از منوی Debug گزینه ی Go Main را انتخاب کنید. انتخاب این گزینه باعث می شود که اجرای برنامه از تابع اصلی main آغاز شود. حال از منوی Debug گزینه ی Run را انتخاب کنید. پیغام “Volume example started” باید در پنجره ی Stdout پایین صفحه ظاهر شود. با انتخاب گزینه ی Halt از منوی Debug می توانید اجرای این دستور را متوقف کنید.

● بخش سوم – تعریف سمبل FILEIO

همانطور که در هنگام اجرای برنامه ی قبل متوجه شدید، قسمتی از برنامه که مربوط به عملیات چاپ پیغام “begin processing” و سپس فراخوانی تابع DATAIO بود و بین دو دستورالعمل شرطی #ifdef و #endif قرار داشت، اجرا نشد. دلیل این امر این بود که ما هنوز، سمبل FILEIO را تعریف نکرده بودیم. در ادامه، به تعریف آن می پردازیم.

از منوی project گزینه ی Build Options را انتخاب کنید. در Tab مربوط به compiler، گزینه ی Preprocessor را از لیست سمت چپ انتخاب کنید. در قسمت Pre-Define Symbol نام FILEIO را تایپ کنید. اکنون با تعریف این سمبل، قسمتی از کد که میان دو دستورالعمل شرطی بود، اجرا خواهد شد. کلید OK را فشار دهید و از این صفحه خارج شوید.



شکل ۵

اگر دوباره پروژه را Rebuild کرده و run کنید، این بار خط های کد بین این دو دستور اجرا خواهد شد و پیغام مطلوب در پنجره ی Stdout نمایش داده خواهد شد. البته هنوز عملیات واقعی خواندن ورودی انجام نمی گیرد زیرا تابع DATAIO فعلاً خالی از هرگونه دستورالعملی می باشد... در بخش های بعد چگونگی انجام خواندن ورودی را با یک Probe فرا خواهیم گرفت.


• بخش چهارم-آشنایی با طریقه ی استفاده Breakpoint ها و پنجره ی Watch


در این بخش می خواهیم به طریقی قسمت های مختلف برنامه را چک کنیم. معمولاً در طول اجرا شدن (run شدن) یک برنامه، نیاز به چک کردن مقدار یک متغیر در محل خاصی از برنامه پیدا می کنیم. در این قسمت چگونگی انجام این امر را به شیوه ی استفاده از Probepoint ها و Watch Window فرا خواهیم گرفت.


ابتدا توجه کنید که محدودیتی روی تعداد Breakpoint هایی که در یک برنامه می توان قرار داد، وجود ندارد. به منظور گذاشتن یک Breakpoint در برنامه، ابتدا فایل Volume.c را باز کنید. Cursor را در کنار خط DataIO() قرار دهید و از درون toolbar بالای صفحه، icon ای که یک دست را


نمایش می دهد و Toggle Breakpoint نام دارد، انتخاب کنید. همین کار را می توانید تنها با فشردن کلید F9 انجام دهید. با قرار دادن این Breakpoint، برنامه تا این محل اجرا شده و سپس متوقف خواهد گشت. با استفاده از پنجره ی Watch می توانیم در این نقطه، مقدار متغیرهایی که می خواهیم را بخوانیم. از منوی View، گزینه ی Watch Window را انتخاب کنید. پنجره ی کوچکی در پایین صفحه باز خواهد شد. در هنگام اجرای برنامه ها، این پنجره می تواند مقدار متغیرها را به ما نشان دهد. به طور پیش فرض، تمام متغیرهایی که در تابع در حال اجرا به صورت Local وجود دارند، قابل مشاهده در این پنجره می باشند. دوباره گزینه ی Go Main را از منوی Debug انتخاب کنید و برنامه را Run کنید. حال در مقابل اسامی متغیرهای Input و Output در پنجره ی Watch دو عدد ثبت خواهد شد. اگر در حین اجرای برنامه، می خواستید متغیر دیگری را در این پنجره، Watch کنید، از پایین آن پنجره گزینه ی Watch1 را انتخاب نمایید و در قسمت Name نام متغیری که می خواهید از مقدار درون آن اطلاع حاصل کنید، بنویسید.


توجه کنید که با انتخاب کردن گزینه های مختلف روی toolbar بالای صفحه، می توان از دستورات step برای دنبال کردن قدم به قدم برنامه، به صورت های زیر استفاده کرد:

با انتخاب  icon می توان از روی خط مربوط به فراخوانی تابع DataIO پرید.

با انتخاب  icon می توان وارد یک تابع شد.

با انتخاب  icon می توان از یک تابع خارج شد.

با انتخاب  icon می توان برنامه را تا محل cursor اجرا کرد.

پیش از رفتن به قسمت بعد، بر روی  icon از toolbar، تمام Breakpoint ها را بردارید.

• بخش پنجم – آشنایی با ProbePoints جهت انجام عملیات FILEIO

Probepoint ها جهت خواندن ورودی یا نوشتن خروجی روی PC به کار می روند.

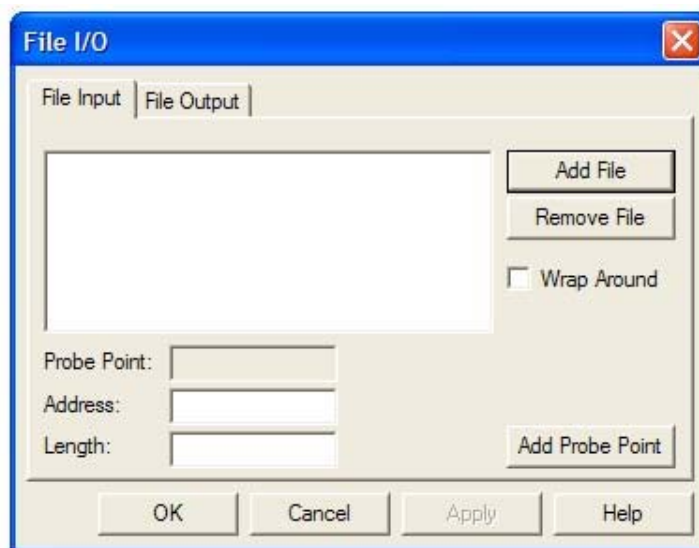
در ابتدا به بیان تفاوت های میان Probepoint ها و Breakpoint ها می پردازیم.

۱- Probepoint ها برنامه را تنها به طور لحظه ای متوقف کرده و یک عمل خاص که می تواند خواندن یا نوشتن روی PC یا نمایش یک متغیر در یک Graph باشد، را انجام می دهند و دوباره اجرای برنامه ادامه پیدا می کند

۲- Breakpoint ها برنامه را در نقطه ی مشخصی متوقف کرده و تمام پنجره ها را Update می کنند و تا وقتی به طور دستی اجرای برنامه از سر گرفته نشود، برنامه متوقف است.

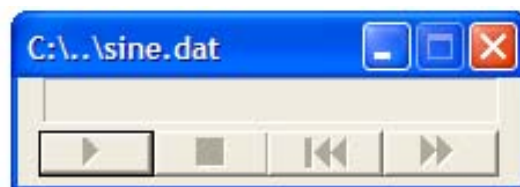
در این بخش ما از Probepoint ها جهت خواندن یک ورودی از یک فایل جهت شبیه سازی استفاده می کنیم.

Cursor را دوباره در کنار خط DataIO قرار دهید. icon را از Toolbar انتخاب کنید. حال شما در این خط یک Probepoint دارید جهت متصل کردن این پراب به فایل ورودی مورد نظر خود، از منوی File گزینه ی File I/O را انتخاب کنید. پنجره ی زیر برای شما باز خواهد شد.



شکل ۶

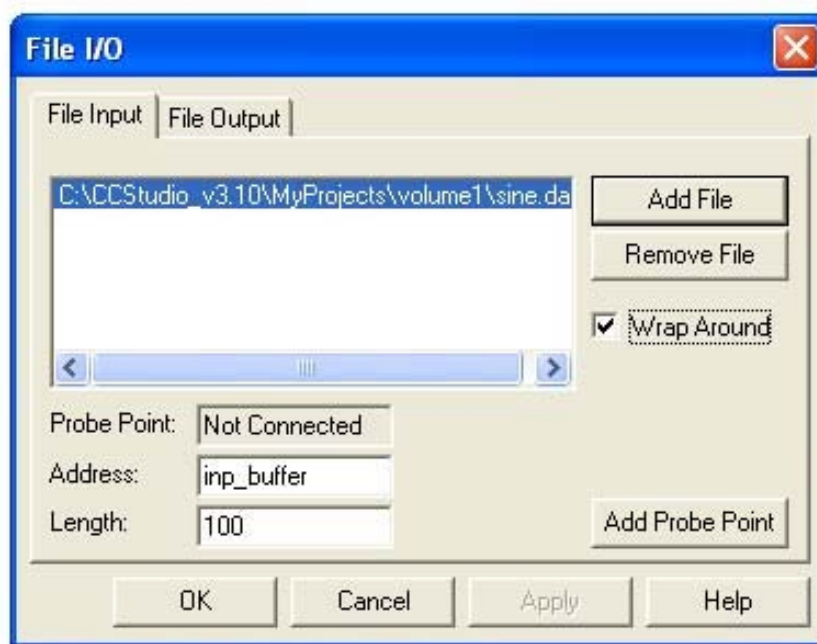
حال در Tab مربوط به File Input ، گزینه ی Add File را انتخاب کنید. از درون Folder پروژه ی خود، فایل آماده ی Sine.dat را که حاوی نمونه هایی از یک سیگنال سینوسی است را انتخاب نمایید. یک پنجره جهت کنترل این فایل به صورت زیر ظاهر خواهد شد. بعداً در طول اجرای برنامه، شما می توانید با استفاده از این پنجره، حرکت درون این فایل داده را سریع یا آرام یا متوقف کنید.



شکل ۷

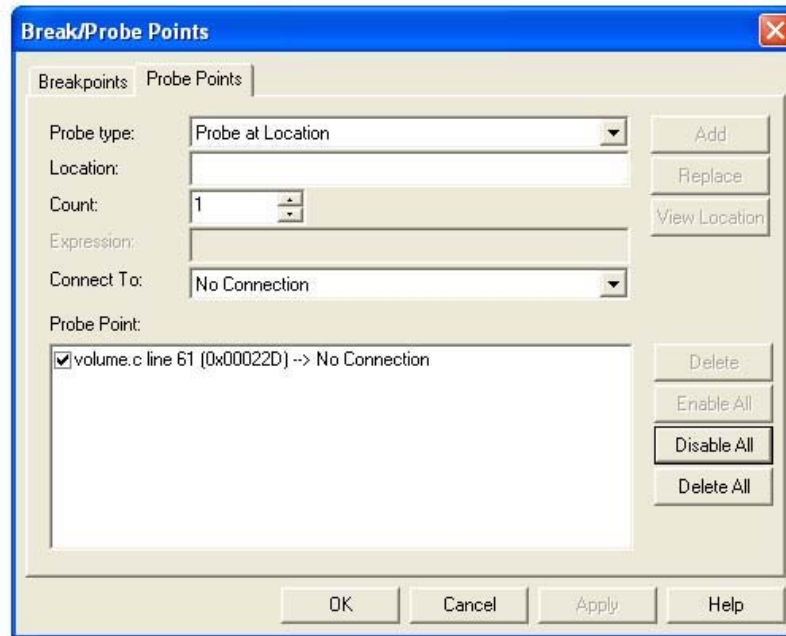
در پنجره ی File I/O در جعبه ی Address ، نام inp_buffer را به عنوان متغیری که می خواهید نمونه های خوانده شده از فایل درون آن ریخته شود، بنویسید. و برای این که مشخص کنید در

هر بار مواجه شدن با این پروب، چند نمونه باید از فایل به عنوان ورودی خوانده شود، درجعه ی Length عدد ۱۰۰ را ثبت کنید زیرا آرایه ی inp_buffer ما که در برنامه تعریف شده بود، یک آرایه ی ۱۰۰ تایی بود. در کنار گزینه ی Wrap around در این پنجره نیز تیک بزنید. این گزینه به برنامه می گوید که هرگاه به انتهای نمونه های درون فایل رسید، دوباره از ابتدا نمونه ها را بخواند. با این کار در واقع اطلاعات موجود در فایل Sine.dat به چشم یک سیگنال پیوسته به چشم خواهد آمد، حتی با اینکه این فایل به خودی خو دارای تعداد محدودی (۱۰۰۰) نمونه از یک سیگنال سینوسی بود.



شکل ۸

حال از درون همین پنجره، گزینه Add Probe Point را انتخاب کنید. پنجره ی جدیدی به شکل بعد ظاهر خواهد شد:

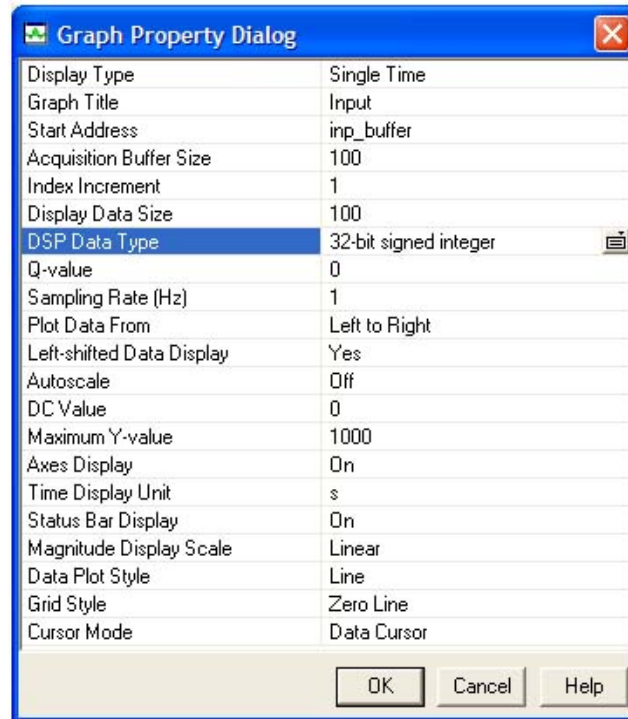


شکل ۹

در لیست Probe Point ها، عبارت Volume.c line 61 را Highlight کنید. در قسمت Connect to ، فایل C:\...\sine.dat را انتخاب کنید. کلید Replace را فشار دهید تا پروب به حالت Connected تغییر یابد. با فشردن کلید OK پنجره را ببندید. اگر دوباره برنامه را Build و اجرا کنید، این بار عملیات مربوط به خواندن داده از ورودی و پردازش آن انجام می گردد. در قسمت بعد به مشاهده ی نمودار سیگنال ورودی و خروجی می پردازیم.

• بخش ششم – رسم نمودار

در این قسمت به مشاهده ی نمودار تغییرات سیگنال ورودی بر حسب زمان می پردازیم. از منوی view ، گزینه ی Graph و سپس گزینه ی Time/Frequency را انتخاب کنید. در پنجره ی باز شده، می توانید مشخصات نمودار را به دلخواه تغییر دهید. قسمت های Graph Title و Start Address و Acquisition Buffer size و Display data size و DSP data Type و Autoscale و Maximum Y-Value را مطابق شکل زیر تغییر دهید.



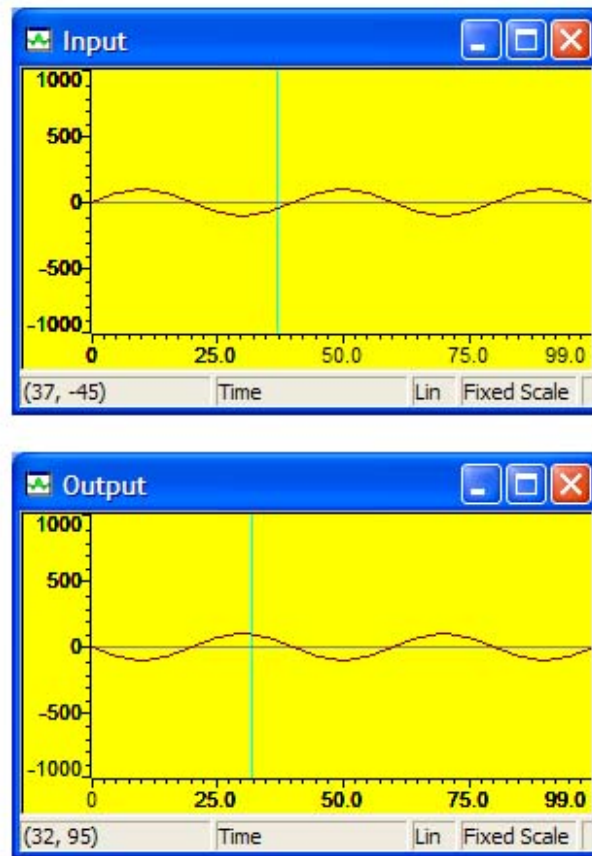
شکل ۱۰

حال با فشردن کلید OK ، یک صفحه ی خالی جهت نمایش نمودار متغیر inp_buffer ظاهر خواهد شد. بر روی این صفحه کلیک کرده و گزینه ی Clear Display را انتخاب کنید. می توانید تمام عملیات بالا را برای متغیر out_buffer تکرار کنید تا در یک صفحه ی مجزا نمودار خروجی نیز قابل مشاهده باشد.

تا به اینجا ما یک پروب به محل مناسبی از برنامه متصل نموده ایم. وقتی اجرای برنامه به اینجا رسید، برنامه از فایل مورد نظر، تعدادی نمونه به عنوان ورودی می خواند و به کار خود ادامه می دهد. اما توجه کنید که یک Probe Point نمی تواند Graph ها را update کند و بدین منظور باید حتماً از یک Breakpoint در این نقطه استفاده کنیم و برای اینکه برنامه با رسیدن به این نقطه به جای متوقف شدن تنها Graph ها را به روز کرده و به کار خود ادامه دهد، این بار به جای Run کردن برنامه، آن را Animate می کنیم. (با فشردن کلید F12 این عمل انجام خواهد شد.) زیرا در این حالت ، برنامه با رسیدن به Breakpoint تمام پنجره ها را به روز می کند اما بر خلاف حالت Run سبب متوقف شدن برنامه نخواهد شد.

هر بار که برنامه به Probe Point می رسد، برنامه ۱۰۰ نمونه از فایل مورد نظر می خواند و آنها را در آرایه ی inp_buffer ذخیره می کند. در همین زمان، به دلیل وجود یک Breakpoint در کنار

Probepoint، تمام نمودار ها به روز می شوند اما اجرای برنامه متوقف نمی گردد. در این هنگام نمودارها به صورت زیر در خواهند آمد.



شکل ۱۱

با انتخاب گزینه ی Halt از منوی Debug، اجرای برنامه متوقف خواهد گردید.

• بخش هفتم – تغییر مقدار متغیر Gain بدون تغییر دادن کد

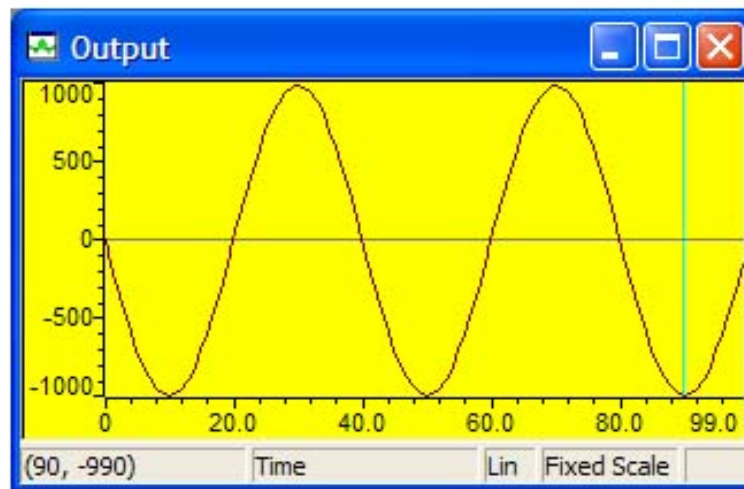
همانطور که گفتیم، این برنامه از یک فایل، سیگنال ورودی را دریافت کرده و آنها را در یک عدد ثابت به نام Gain ضرب می کند و در آرایه ی خروجی ذخیره می نماید. این عملیات با اجرای مداوم دستور زیر در یک حلقه انجام می پذیرد:

`*output++ = *input++ * gain`

حال فرض کنید، در حین اجرای برنامه، به این نتیجه برسیم که مقدار Gain باید تغییر کند. یکی از روش های انجام این کار، استفاده از پنجره ی Watch می باشد.

با انتخاب گزینه ی Watch Window از منوی View، پنجره ی Watch باز می شود. درون این پنجره و در قسمت Watch1، نام متغیر gain را وارد کنید. با این کار مقدار آن ظاهر خواهد گشت.

حال می توانید به طور مستقیم در همین قسمت، مقدار متغیر gain را تغییر دهید. مثلاً به جای مقدار ۱ مقدار ۱۰ را رو به روی آن وارد کنید. اگر برنامه دوباره اجرا شود، این بار نمودار خروجی به شکل زیر در می آید:



شکل ۱۲

یک روش دیگر نیز برای تغییر مقدار متغیر gain در حین اجرای برنامه وجود دارد. در این روش از یک نوار لغزنده که توسط یک GEL File تولید می شود، استفاده می کنیم.

ابتدا باید ببینیم GEL File ها چه هستند. GEL یا General Extention Language زبان برنامه نویسی خاصی مشابه با زبان C می باشد که به ما کمک می کند با نوشتن تنها چند خط کد، کارهای خاصی مانند تغییر متغیرها در حین اجرای برنامه با روشی راحت تر، را انجام دهیم. این زبان امکانات دیگری نیز در اختیار ما می گذارد که در بخش بعد به بررسی آن می پردازیم. در اینجا به تعریف نوار لغزنده برای تغییر متغیر Gain در حین اجرای برنامه می پردازیم. ابتدا باید مطمئن شویم که متغیر مورد نظر در برنامه به صورت یک متغیر Global تعریف شده باشد. حال از منوی File گزینه ی New Source File را انتخاب کرده و در صفحه ی باز شده، کد

زیر را بنویسید:

```
/*gaincontrol.gel Create slider and vary gain of sinewave*/
menuitem "Sine Gain"

slider Gain(10,35,5,1,gain_parameter) /*incr by 5, up to 35*/
{
gain = gain_parameter; /*vary gain of sine*/
}
```

این کد، به ما می گوید که می خواهیم در منوی بالای صفحه، گزینه ای به نام Sine Gain اضافه شود که با فشردن آن Slider ما ظاهر شود. از طرفی، خط بعدی کد به ما می گوید که می خواهیم، مقدار مینیمم این Slider عدد ۱۰ و مقدار ماکزیمم آن عدد ۳۵ باشد و فاصله ی بین این دو عدد با فاصله های ۵ تایی تقسیم بندی شود. پارامتر چهارم درون پرانتز را همواره یک قرار می دهیم و پارامتر آخر نیز باید نام یک متغیر موقتی به غیر از متغیر مطلوب تعریف شده درون کد اصلی ما باشد. حال در بدنه ی این کد، تنها می نویسیم که متغیر Gain را برابر این متغیر موقتی قرار بده.

اکنون این فایل را با نام gaincontrol.gel در Folder پروژه ی خود ذخیره کنید. حال از منوی File گزینه ی Load GEL را انتخاب کرده و همین فایل Gaincontrol را به این ترتیب به پروژه اضافه کنید. دقت کنید که اکنون به منوی GEL در بالای صفحه، یک گزینه ی Sine Gain اضافه شده است، و درون این منو گزینه ی gain وجود دارد که با انتخاب آن، Slider گین روی صفحه ظاهر می گردد که شکلی به صورت زیر دارد:



چنانچه برنامه را rebuild و سپس load کنید و دوباره run نمایید، می توانید در حال اجرای برنامه با بالا یا پایین کردن نشان لغزنده، مقدار gain را به دلخواه تنظیم کنید. برای اطمینان از اینکه تغییرات نوار لغزنده روی متغیر gain تاثیر دارد، می توانید این متغیر را با پنجره ی Watch چک کنید.

• بخش هشتم – کاربردهای دیگر GEL File ها

به طور کلی می توان از طریق ایجاد GEL File ها، برای انجام کارهایی که در حین اجرای برنامه، زیاد انجام خواهند گرفت، راه میانبری ایجاد کنیم. یکی از این کاربردها، همان تغییر مقدار متغیرها در حین اجرای برنامه بود که در بخش قبل فرا گرفتیم. حال می خواهیم کاربرد دیگری از این نوع فایل ها را بیاموزیم.

در این کاربرد، می خواهیم به منوی Gel از منوی بالای صفحه، گزینه های دلخواهی اضافه کنیم که کاری را که ما می خواهیم انجام دهیم، با انتخاب کردن این گزینه در حین اجرای برنامه به سرعت انجام شود. فرض کنید این کار، نشان دادن یک پیغام در صفحه ی stdout باشد. به این منظور دوباره source file جدیدی باز کرده و کد زیر را در آن بنویسید:

```
menuitem "GEL Welcome Tool";  
hotmenu Welcome_To_GEL_Function()  
{  
    GEL_TextOut("GEL is a solid tool.\n");  
}
```

حال این فایل را نیز با نام welcome.gel ذخیره کنید و آن را load کنید با هر بار load کردن یک GEL فایل، خود برنامه آنرا کامپایل کرده و اگر خطایی وجود داشته باشد به شما گزارش خواهد داد. پس هر بار که تغییری در این نوع فایل ها اعمال می کنید، آن را Save کرده و سپس دوباره reload کنید تا این تغییر در آن ثبت شود.

با این چند خط کد به منوی GEL شما، گزینه ی GEL Welcome Tool اضافه خواهد شد که با هر بار انتخاب آن، پیغام مطلوب به کاربر نشان داده می شود.

آنچه در بالا آمد، شرح مختصری بر قابلیت های مختلف محیط برنامه نویسی Code Composer بود که جهت ایجاد آشنایی اولیه با این نرم افزار قدرتمند در این جلسه آورده شده بود. واضح است که این نرم افزار دارای قابلیت های بسیار گسترده تری از آنچه در این جا مطرح شد می باشد. برخی از این قابلیت ها، به تدریج در حین انجام آزمایش های دیگر و در جلسات بعدی به شما معرفی خواهد شد.

جلسه دوم:

- **عنوان آزمایش:** نمونه برداری از سیگنال صدا و ولوم دادن به آن در خروجی
 - **هدف آزمایش:** فراگیری مفاهیم وقفه های نرم افزاری و چگونگی کار با آنها، آشنایی با توابع API، آشنایی با چگونگی ایجاد یک پروژه ی کامل در محیط CCS جهت پروگرام کردن برد و فراگیری چگونگی پروگرام کردن برد به طور عملی و اجرا کردن برنامه ی ساده ی ولوم دادن به صدای ورودی روی آن
 - **شرح آزمایش:** در این آزمایش، ابتدا می خواهیم با چگونگی نمونه برداری از یک سیگنال صوتی توسط امکانات برد آشنا شویم. این کار را از طریق آشنایی با وقفه های نرم افزاری انجام خواهیم داد. سپس با نمونه برداری از سیگنال صدای ورودی، یک عملیات ساده ی ولوم دادن به آن را انجام داده و دوباره در خروجی پخش می کنیم. چون این اولین آزمایشی خواهد بود که به طور عملی روی برد پیاده سازی خواهد شد و لازم است تمام ملاحظات عملی در نوشتن کد آن در نظر گرفته شود، نوشتن کدهای این آزمایش بیشتر جنبه ی آموزشی خواهد داشت و در نتیجه به طور مرحله به مرحله در این دستورکار شرح داده خواهد شد تا بتوانند راهنمایی برای نوشتن کدهای پیچیده تر در جلسات بعدی باشند.
- پس از اینکه کد این برنامه را با در نظر گرفتن تمام ملاحظات عملی، برای برد نوشتیم، با وصل کردن برد به کامپیوتر، آن را پروگرام کرده و نتیجه را به طور عملی تست می کنیم.
- ابتدا لازم است به مرور کلیتی راجع به امکانات برد که می توانند جهت نمونه برداری و سپس ذخیره نمونه ها و انجام پردازش های مختلف روی آن به کار روند، پردازیم.
- همانطور که گفتیم، برد DSK6416 دارای دو ورودی Line-in و Microphone می باشد که از این طریق می توانیم صدای خود یا یک سیگنال صدای دیگر را به عنوان ورودی به برد وصل کنیم. برد مجهز به یک codec از نوع AIC23 می باشد که می تواند از سیگنال ورودی با نرخ های مشخصی بین ۸ تا ۹۶ کیلو هرتز، نمونه برداری نماید. ارتباط این codec با پروسور اصلی برد یا همان تراشه ی TMS320C6416 از طریق پورت های سریال McBSP برقرار می شود و نمونه ها از این طریق برای پروسور اصلی فرستاده می شوند تا پردازش روی آن ها انجام گیرد. برد دارای دو پورت سریال McBSP1 و McBSP2 می باشد، که McBSP1 به منظور initialize کردن تنظیمات داخلی codec به کار می رود و پارامترهای مطلوب ما از این طریق برای codec فرستاده می شود و ما با آن سر و کار نخواهیم داشت. پورت McBSP2 برای رد و بدل کرد داده های صوتی بین codec و

پروسسور به کار می رود و ما نیز از همین پورت برای فرستادن نمونه ها به پروسور استفاده خواهیم کرد.

خواهیم دید که برنامه ی ما از چهار قسمت عمده تشکیل خواهد شد:

۱- initialize. کردن هریک از این ادوات جانبی. این عملیات می تواند به سهولت ، با به کار گیری توابعی به نام Application Programming Interface یا توابع API انجام پذیرند. این توابع که از مزیت ها و ویژگی های جدید و منحصر به فرد بردهای نسل جدید شرکت TI می باشند، در library هایی به نام های Chip Support Library (CSL) و Board Support Library (BSL) تعبیه شده اند و کار کردن با آنها بسیار ساده می باشد.

۲- نوشتن کدهای لازم برای کار با وقفه ها.

۳- نوشتن بدنه ی اصلی برنامه با توجه به اینکه قرار است برنامه با تکیه بر روش interrupt-based نوشته شود.

۴- نوشتن Interrupt Service Routine که در بخش های بعد به توضیح آن می پردازیم.

۵- نوشتن فایل های لازم دیگر و افزودن آنها به پروژه جهت اجرای آن روی برد
حال هریک از این بخشها را به تفصیل بررسی می کنیم و سپس کد را به صورت کامل در انتهای دستور کار خواهیم دید:

• بخش اول - initialize کردن ادوات جانبی

در هر برنامه ای که از codec برد برای نمونه برداری از یک سیگنال ورودی آنالوگ استفاده می کند، لازم است ابتدا initialization های خاصی انجام پذیرند. که از میان آنها می توان به initialize کردن خود برد، codec و پورت های ارتباط سریال اشاره کرد. هر یک از این initialization ها به سهولت و به وسیله ی توابع خاصی در library های آماده ی برد، انجام می گیرند که در ادامه به توضیح هر یک از آن ها می پردازیم. توجه کنید که یادگیری عملکرد این توابع و چگونگی استفاده از آنها جهت initialize کردن برد و ادوات جانبی روی آن، از اهمیت خاصی برخوردار است زیرا از این پس، باید تمام این مراحل را در تمام آزمایش های آینده تکرار کنیم. در ادامه ی این بخش، کد این قسمت به بخش های مختلفی تقسیم شده و پس از آوردن هر بخش به توضیح آن می پردازیم.

گفتیم که اکثر توابعی که در اینجا به کار خواهیم برد، در library به نام CSL قرار دارند. کار کردن با توابع درون این library، علاوه بر افزودن آن به پروژه از طریق include کردن آن در کد، نیاز به تعریف کردن نام چیپ در اولین خط کد دارد. به این معنی که نخستین خط کد در source file برنامه

حتماً باید به تعریف نوع چیپ توسط دستور `#Define` پردازیم. پس برای تعریف برد ما، باید کد زیر به ابتدای برنامه افزوده شود:

#Define CHIP_6416

البته این سمبل می تواند از طریق گزینه ی `preprocessor` در پنجره ی `Build Options`، همانطور که در جلسه ی قبل به تعریف سمبل `FILEIO` پرداختیم، نیز تعریف شود. پس از این مرحله، باید تمام `header file` هایی که به طریقی از یکی از توابع آماده ی درون آنها یا از یک متغیر تعریف شده درون آنها استفاده خواهیم کرد، به کد `include` کنیم. پس خطوط زیر را به کد برنامه می افزایم:

```
#include <stdio.h>
#include <c6416.h>
#include <csl.h>
#include <csl_mcbbsp.h>
#include <csl_irq.h>

#include "dsk6416.h"
#include "dsk6416_aic23.h"
```

حال ابتدا به تعریف دو متغیر که بعداً برای `initialize` کردن `codec` به ما کمک خواهند کرد، می پردازیم. متغیر اول متغیری از کلاس `DSK6416_AIC23_CodecHandle` می باشد که در `header file` ای که در ابتدا به برنامه `include` کردیم و `dsk6416_aic23` نام داشت، تعریف شده است. این متغیر که آن را به دلخواه `hCodec` می نامیم، یک `Handle` برای `initialize` کردن پارامترهای مختلف `codec` خواهد بود. متغیر دوم که آن را به دلخواه، `config` می نامیم، یک متغیر از کلاس `DSK6416_AIC23_Config` می باشد که وضعیت مطلوب پارامترهای `codec` در آن ریخته می شود و از طریق توابع خاصی برای `initialize` کردن `codec` به کار می رود. جهت سادگی بیشتر، پس از تعریف این متغیر، مقدار آن را برابر مقدار پیش فرضی که در همین `header file` برای یک تنظیم پیش فرض تعریف شده است، قرار می دهیم. پس در نهایت تعریف این دو متغیر به ترتیب زیر انجام می شود:

```
DSK6416_AIC23_CodecHandle hCodec;
DSK6416_AIC23_Config config = DSK6416_AIC23_DEFAULTCONFIG;
```

حال باید در بدنه ی اصلی کد، به `initialize` کردن ادوات مختلف با استفاده از متغیرهای تعریف شده و توابع آماده پردازیم. به این منظور کدهای زیر را به بدنه ی اصلی برنامه اضافه می کنیم.

ابتدا باید Board support library یا BSL ما initialize شود. به این منظور باید بنویسیم:

DSK6416_init();

حال همانطور که گفتیم، باید از دو متغیر hCodec و Config برای initialize کردن codec استفاده کنیم. به این منظور از تابع آماده ای به نام openCodec استفاده می کنیم. طریقه ی استفاده از این تابع که آدرس یک متغیر از کلاس Config را به عنوان ورودی گرفته و پارامترهای یک Handle را به عنوان خروجی به ما می دهد به قرار زیرست:

hCodec = DSK6416_AIC23_openCodec(0 , &config);

ضمناً فرکانس نمونه برداری که در این آزمایش ۸ کیلو هرتز در نظر گرفته می شود، توسط یک تابع API دیگر و یک متغیر که در header file های برنامه تعریف شده است، به شرح زیر تنظیم می گردد:

DSK6416_AIC23_setFreq(hCodec , DSK6416_AIC23_FREQ_8KHZ);

اکنون می خواهیم به initialize کردن پورت سریال پردازیم. این کار را با استفاده از چند تابع آماده به نام McBSP_FSETS انجام می دهیم. از آنجایی که قرار است این ارتباط سریال بر اساس وقفه های نرم افزاری نوشته شود، ابتدا لازم است که وقفه های مربوط به ارسال و دریافت سریال پورت شماره ی ۲ به طریقی فعال شوند به این منظور به وسیله تابع FSETS ، مد وقفه ی دریافت/ارسال سریال (XINTM/RINTM) مربوط به پورت ۲ (SPCR2) را تغییر می دهیم به طوریکه در انتهای فرستادن هر frame داده، یک وقفه نرم افزاری ایجاد شود. به این منظور، کدهای زیر باید نوشته شود:

McBSP_FSETS(SPCR2 , RINTM , FRM);
McBSP_FSETS(SPCR2 , XINTM , FRM);

حال توجه کنید که به طور معمول، نمونه های خوانده شده توسط codec ، به صورت فریم های ۳۲ بیتی هستند که ۱۶ بیت آن مربوط به کانال چپ ورودی و ۱۶ بیت آن مربوط به داده ی کانال راست ورودی می

باشد. پی لازم است که طول فریم های تعریف شده برای پورت سریال نیز ۳۲ بیت باشد تا بتواند نمونه ها را به درستی از codec دریافت کرده و در انتهای هر فریم، وقفه ی مناسب را فعال کند. برای تعریف طول فریم داده برای پورت سریال، از توابع FSETS به شیوه ی زیر استفاده می کنیم:

```
McBSP_FSETS(RCR2 , RWDLEN1 , 32BIT );  
McBSP_FSETS(XCR2 , XWDLEN1 , 32BIT );
```

پیش از اینکه به بررسی بخش های بعدی پردازش سیگنال را شرح دهیم ابتدا لازم است روش کار با وقفه های نرم افزاری و شیوه ی استفاده از آنها در کارهای پردازش سیگنال را شرح دهیم. روش استفاده از وقفه های نرم افزاری، شیوه ای متداول در پردازش real-time سیگنال ها می باشد. علت کاربرد گسترده این روش، مزیت آن بر روش ها و الگوریتم های پیچیده ی سنکرون کردن ورود داده و پردازش می باشد. در این آزمایش، هر بار که یک نمونه ی جدید از طریق ارتباط سریال در پروسسور دریافت می گردد، یک وقفه ی نرم افزاری رخ می دهد. و وقفه ی ایجاد شده، مسیر برنامه را به سمت یک زیر برنامه ی سرویس دهی (Interrupt Service Routine یا ISR) منحرف می کند. در این زیر برنامه ما پس از خواندن نمونه ی دریافتی از بافر پورت، عملیات مربوط به پردازش نمونه ها را که در این مثال تنها شامل ضرب کردن در یک گین و فرستادن آن به D/A می باشد، انجام می دهیم. پس از اتمام این زیر برنامه، اجرای برنامه به همان مسیر عادی خود باز می گردد و دوباره ما منتظر دریافت نمونه ی بعدی و یا رخ دادن وقفه ی بعدی می شویم.

برای اینکه این عملیات انجام شوند، لازم است از قابلیت های برد DSK استفاده کنیم، تا یک Event که در اینجا همان وقفه ی مربوط به دریافت سریال می باشد را به یک وقفه ی خالی CPU مپ کنیم (به این معنا که هر بار این Event رخ داد، پوینتر اجرای برنامه به جدول بردارهای وقفه در حافظه رفته و از خانه ی مربوط به این وقفه ی خاص CPU آدرسی را که زیر برنامه ی سرویس وقفه در آن قرار دارد را بخواند) و سپس آدرس ISR پردازش سیگنال را در جدول بردارهای وقفه در خانه ی مربوط به این وقفه قرار دهیم. توجه کنید که عمل reset کردن برد نیز خود یک وقفه برای برد محسوب می شود و به صورت پیش فرض آدرس 0 به ISR این وقفه اختصاص می یابد و در نتیجه با روشن کردن برد، پوینتر برنامه به آدرس 0 رفته و کدی که در آنجا قرار گرفته است را انجام می دهد. سپس با هر بار رخ دادن یک وقفه نرم افزاری، پوینتر برنامه مسیر عادی اجرای کد را رها کرده و از درون جدول وقفه ها، آدرس ISR ای را که به این وقفه جدید مربوط است و باید اجرا شود را به خود می گیرد. پس از اتمام این زیر برنامه،

دوباره پوینتر اجرای برنامه، مقدار پیشین خود را بازیابی کرده و اجرای برنامه را از جایی که رها شده بود، از سر می گیرد.

پس همانطور که ملاحظه می شود، اولین وظیفه ی ما نوشتن برنامه ی لازم برای مپ کردن این Event به وقفه ی لازم و سپس قرار دادن آدرس ISR پردازش سیگنال در جدول سرویس وقفه ها (interrupt service table) می باشد. توجه کنید که CPU ما دارای قابلیت handle کردن ۱۶ وقفه ی نرم افزاری را دارد که در اینجا ما Event مطلوب را به وقفه ی شماره ۱۵ CPU مپ خواهیم کرد. Event ای که در اینجا منبع ایجاد وقفه می باشد، وقفه ی مربوط به دریافت سریال پورت McBSP2 می باشد (RINT2) . عمل مپ کردن این Event به وقفه ی مطلوب توسط یک زیر برنامه با نام hook_int() در source file برنامه انجام می شود. عمل قرار دادن آدرس ISR دریافت سریال (serialPortRcvISR()) در جدول وقفه ها نیز توسط فایلی به نام vectors.asm که به زبان اسمبلی نوشته می شود انجام می گیرد.

حال ابتدا به نوشتن زیر برنامه ی hook_int() در source file برنامه و سپس به بررسی فایل اسمبلی لازم جهت قرار دادن آدرس ISR در جدول بردار وقفه می پردازیم.

• بخش دوم – نوشتن زیر برنامه ی مپ کردن Event دریافت سریال به وقفه ی ۱۵ پروسور و فایل اسمبلی لازم جهت قرار دادن آدرس ISR وقفه دریافت سریال در خانه ی مربوط به وقفه ی شماره ۱۵ CPU در جدول بردارهای وقفه

ابتدا نوشتن زیر برنامه ی hook_int() در source file برنامه می پردازیم که قرار است Event مربوط به دریافت سریال را به وقفه ی ۱۵ پروسور مپ کند. در این زیر برنامه، ابتدا تمام وقفه ها به جز وقفه ی non-maskable را غیر فعال می کنیم تا اگر در همین ابتدای برنامه پیش از مپ کردن صحیح وقفه ها، وقفه ای آمد، به آن توجهی نشود. سپس Event مربوطه را مپ می کنیم. این کار با استفاده از یکی از توابع آماده ی API به نام IRQ_map انجام می پذیرد و در آن Event مربوط به RINT2 به وقفه ی ۱۵ پروسور مپ می گردد. سپس وقفه ها را فعال کرده و از زیر برنامه خارج می شویم. مراحل انجام این عملیات در کد زیر با توضیحات لازم آورده شده است:

```

void hook_int()
{
    IRQ_globalDisable();           // Globally disables interrupts
    IRQ_nmiEnable();               // Enables the NMI interrupt
    IRQ_map(IRQ_EVT_RINT2,15);     // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT2);     // Enables the event
    IRQ_globalEnable();            // Globally enables interrupts
}

```

حال به نوشتن فایل اسمبلی لازم جهت تنظیم جدول بردار وقفه ها می پردازیم. در نوشتن این کد که در فایل به نام vectors.asm نوشته خواهد شد، در مقابل نام هر وقفه ی نرم افزاری، یا دستور NOP (به معنای No operation) را قرار می دهیم و یا آدرسی را که با رخ دادن این وقفه، پوینتر اجرای برنامه باید به خود بگیرد را به طریقی در خانه های حافظه ی مربوطه ثبت می کنیم. در هر دو حالت نیز یک Delay به اندازه زمان اجرای ۸ دستور NOP ایجاد می کنیم تا CPU زمان کافی برای انجام کارهای خواسته شده را پیدا کند.

کد برنامه به همراه توضیحات لازم در ادامه آورده می شود، توجه کنید که برای مشخص کردن آدرس 0 (مربوط به وقفه ی reset) و همچنین آدرسی که ISR مربوط به وقفه ی دریافت سریال در آن قرار گرفته است، از دو reference به نام های _c_int00 و _serialPortRcvISR استفاده کرده ایم:

```

; vectors.asm
.ref    _c_int00
.ref    _serialPortRcvISR ; refer the addr of ISR defined in C code
.sect   "vectors"

RESET_RST:  MVKL .S2 _c_int00, B0
            MVKH .S2 _c_int00, B0
            B    .S2 B0
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
.NMI_RST:   .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 ;branch to ISR
NOP
NOP
NOP
NOP
NOP

```

• بخش سوم – بدنه ی اصلی برنامه

با در نظر گرفتن این نکته که CPU در هنگامی که منتظر دریافت نمونه ی جدید است، کار خاصی انجام نمی دهد، در بدنه ی اصلی برنامه از یک loop بینهایت استفاده می کنیم تا برنامه را همواره در حال اجرا نگه دارد و هر بار که وقفه ی مطلوب آمد، پس از پردازش آن، دوباره در این loop منتظر دریافت نمونه ی بعدی بماند. این لوپ را با دستور { } (1) while پیاده سازی می کنیم.

• بخش چهارم – نوشتن ISR دریافت سریال

در این زیر برنامه که با دریافت هر نمونه ی جدید اجرا خواهد شد، ابتدا نمونه ی ۳۲ بیتی دریافتی را با تابع آماده ی McBSP_read خوانده و سپس آن را در متغیری به نام temp می ریزیم. سپس اسن متغیر را در گین مطلوب ضرب کرده و با تابع McBSP_write آن را دوباره از طریق سریال به D/A می دهیم. توابع McBSP_read و McBSP_write به عنوان آرگومان ورودی خود از یک Handle آماده به نام DSK6416_AIC23_DATAHANDLE استفاده می نمایند. کد این زیر برنامه به قرار زیر می باشد:

```
interrupt void serialPortRcvISR()
{
    Uint32 temp;

    temp = MCBSP_read(DSK6416_AIC23_DATAHANDLE);
    temp = temp * volumeGain;
    MCBSP_write(DSK6416_AIC23_DATAHANDLE, temp);
}
```

• بخش پنجم – نوشتن فایل های لازم دیگر برای پروژه

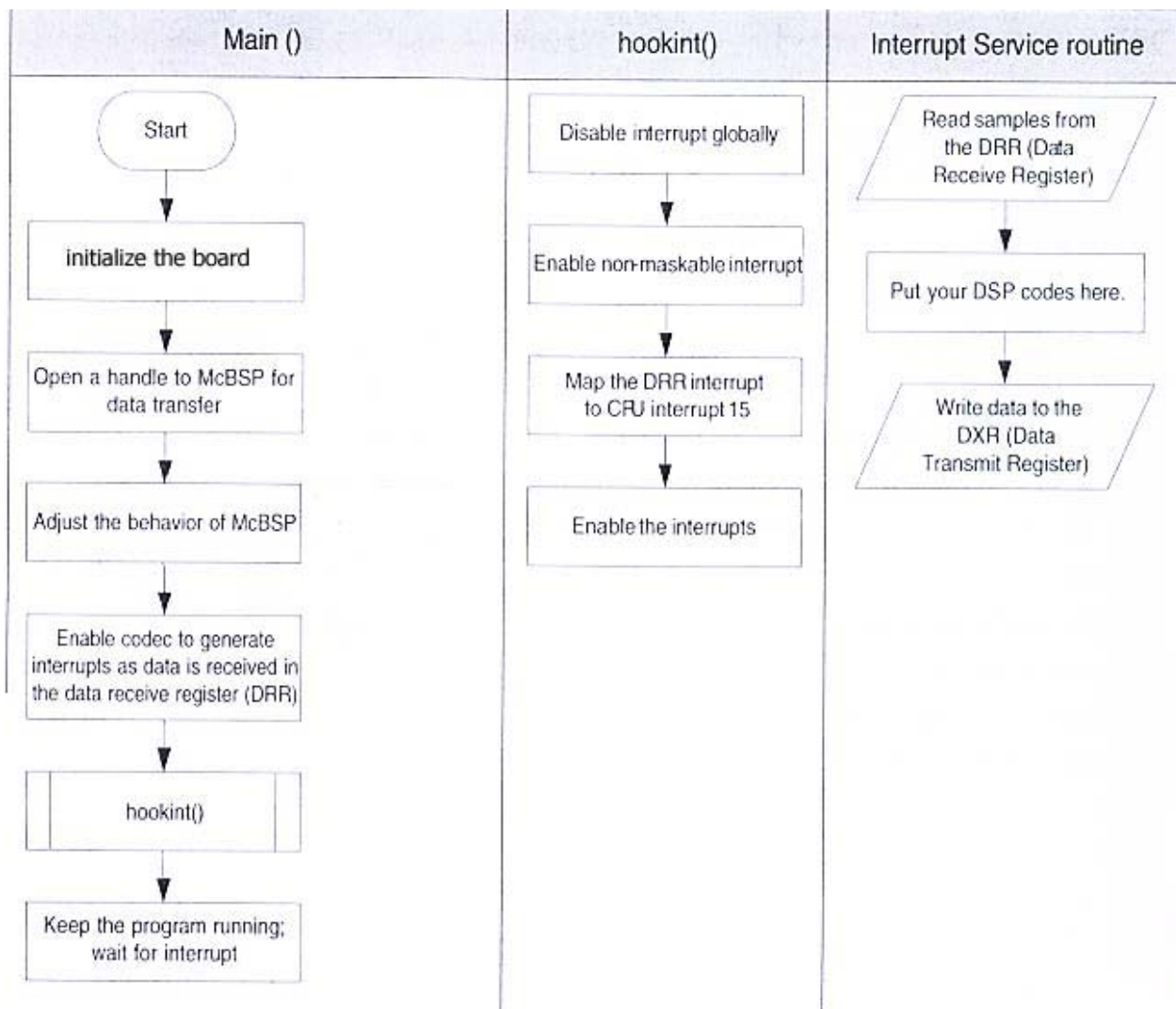
همانطور که در جلسه ی اول توضیح داده شد، هر پروژه برای کار خود، نیاز به یک فایل cmd دارد که در آن باید طریقه ی مپ کردن حافظه مشخص گردد. به همین ترتیب برای این پروژه نیز چنین فایلی به زبان اسمبلی نوشته و به نام lab2.cmd ذخیره می کنیم و سپس به پروژه اضافه می نماییم. این فایل ابتدا کل حافظه را به سه قسمت تقسیم می کند. بخش اول vecs نام دارد و از آدرس 00000000h به طول 00000200h خانه قرار دارد. این بخش حاوی همان جدول بردارهای وقفه خواهد بود. بخش دوم، IRAM نام دارد و از ادامه ی بخش vecs و به طول 0000FE00h ادامه دارد. در نهایت بخش سوم، مربوط به حافظه خارجی برد است و CE0 نام دارد و از ادامه ی بخش قبل به طول 01000000h قرار دارد.

در قسمت دوم این فایل cmd ، بخش ها (section ها) ی مختلفی را در حافظه تعریف کرده و مشخص می کنیم که هر section در کدام یک از سه بخش حافظه قرار خواهند گرفت. در نهایت کد این فایل به قرار زیر خواهد بود:

```
MEMORY
{
    vecs:          o = 00000000h      l = 00000200h
    IRAM:          o = 00000200h      l = 0000FE00h
    CE0:           o = 80000000h      l = 01000000h
}
```

```
SECTIONS
{
    "vectors" >      vecs
    .cinit >          IRAM
    .text >           IRAM
    .stack >          IRAM
    .bss >            IRAM
    .const >          IRAM
    .data >           IRAM
    .far >            IRAM
    .switch >         IRAM
    .sysmem >         IRAM
    .tables >         IRAM
    .cio >            IRAM
}
```

حال پس از توضیح قسمت های مختلف برنامه، ابتدا توضیحات بالا را در یک فلوجارت خلاصه کرده و سپس کد source file برنامه را به طور کامل در ادامه می بینیم.



فلوچارت مربوط به *Source File* برنامه

کد *Source file* نیز به قرار زیرست:


```

#define CHIP_6416

#include <stdio.h>
#include <c6x.h>
#include <csl.h>
#include <csl_mcbasp.h>
#include <csl_irq.h>

#include "dsk6416.h"
#include "dsk6416_aic23.h"

DSK6416_AIC23_CodecHandle hCodec;
DSK6416_AIC23_Config config = DSK6416_AIC23_DEFAULTCONFIG;
// Codec configuration with default settings

interrupt void serialPortRcvISR(void);
void hook_int();
int volumeGain;

void main()
{
    DSK6416_init();           // Initialize the board support library
    hCodec = DSK6416_AIC23_openCodec(0, &config);

    MCBSP_FSETS(SPCR2, RINTM, FRM);
    MCBSP_FSETS(SPCR2, XINTM, FRM);
    MCBSP_FSETS(RCR2, RWDLEN1, 32BIT);
    MCBSP_FSETS(XCR2, XWDLEN1, 32BIT);

    DSK6416_AIC23_setFreq(hCodec, DSK6416_AIC23_FREQ_48KHZ);

    volumeGain = 1;

    hook_int();

    while(1)
    {

```

بخش اول کد Source File برنامہ

```

while(!MCBSP_xrdy(hMcbbsp)); // Write to Control Register of AD535
MCBSP_write(hMcbbsp, data);

while(!MCBSP_rdy(hMcbbsp)); // Read
MCBSP_read(hMcbbsp);
}

interrupt void serialPortRcvISR()
{
    int temp;

    temp = MCBSP_read(hMcbbsp);
    temp = ( temp * volumeGain ) & 0xFFFE;

    MCBSP_write(hMcbbsp,temp);
}

```

بخش درم که Source File برنامه

کارهایی که باید در این جلسه انجام پذیرند:

۱- از درون setup برنامه ی CCS ، گزینه ی DSK6416 را انتخاب کرده و سپس از setup با انتخاب گزینه ی save & quit خارج شوید و برنامه ی CCS را اجرا کنید تا نسخه ی Emulator برنامه اجرا شود و بتوانید پس از نوشتن برنامه، آن را به صورت عملی روی برد نیز پیاده کنید.

۲- برنامه ی لازم جهت دریافت یک سیگنال صدا از ورودی و سپس پخش نسخه ی گین داده شده ی آن در خروجی را با توجه به توضیحات بالابنویسید.

۳- برد را ابتدا با کابل USB به کامپیوتر وصل کرده و سپس آن را به برق وصل کنید.

۴- از منوی Debug، گزینه ی connect را انتخاب کنید تا برد به برنامه متصل و قابل پروگرام کردن گردد. پس از Build کردن برنامه، فایل out. آن را load کنید و سپس گزینه ی run را انتخاب کنید. با دادن سیگنال صدا به ورودی برد، باید قادر باشید از headphone متصل به خروجی برد، صدای گین یافته را بشنوید.

۵- با ایجاد Slider ای برای تغییر متغیر VolumeGain، مقدار گین را تغییر داده و تاثیر آن را روی صدایی که می شنوید بررسی کنید. [برای ایجاد Slider به توضیحات آزمایش اول بخش GEL File ها رجوع کنید.]

۶- پس از اتمام آزمایش ابتدا برد را از برنامه ی CCS ، Disconnect کرده، سپس آن را از برق خارج کرده و از کامپیوتر جدا کنید.