

پروژه بورد سخنگوی طبقات (FSB)

آذر 95

چکیده

بورد FSB یا سخنگوی طبقات وظیفه تولید پیام / موسیقی برای آسانسور را دارد. کارکرد این بورد به این صورت است که با بررسی سیگنال­های کنترلی و شمارنده دریافتی از بورد فرمان آسانسور، وضعیت فعلی آن را تعیین کرده و سپس پیام / موسیقی متناسب با آن را از حافظه داخلی بورد انتخاب کرده و پخش می­کند. پخش موسیقی در بورد در قالب wav می­باشد و بنابراین بورد سخنگوی طبقات در اصل یک wav player می­باشد. بورد FSB300 که در این گزارش توسعه یافته است، علاوه بر حذف مشکلات اصلی بورد FSB200a، دارای چند مزیت اضافی می­باشد که از جمله آنها 1- افزایش فضای ذخیره موسیقی با استفاده از تکنیک­های فشرده­سازی موسیقی و حذف حافظه خارجی میکرو SD نسبت به بورد FSB200a 2- کاهش سخت افزار مورد استفاده در بورد 3- کاهش زمان مورد نیاز برای تست بورد 4- افزایش حافظه داخلی تا 8 مگابایت 5- امنیت بالاتر در مقابل کپی­برداری و ... می­باشد. همچنین بورد FSB310 که بر روی بورد CIB216 سوار می­شود، خواصی مشابه بورد FSB300 دارد با این تفاوت که ارتباط با آن از طریق ارتباط I2C صورت می­پذیرد. در این گزارش پس از توضیح مفاهیم نظری مورد استفاده جهت پیاده­سازی بورد، نرم­افزار متلب (MATLAB) ارائه شده برای تولید موسیقی منطبق با قالب موسیقی مورد استفاده در بورد معرفی شده و سپس بخش­های نرم­افزاری و سخت­افزاری بورد به صورت کامل تشریح می­شوند.

فهرست مطالب

[1- تعریف پروژه 1](#_Toc481078532)

[1-1- تعریف پروژه بورد سخنگوی طبقات (FSB) 2](#_Toc481078533)

[1-1-1- مدهای کاری بورد FSB 3](#_Toc481078534)

[1-1-2- سیگنالهای دریافتی در بورد FSB 3](#_Toc481078535)

[1-1-3- وضعیتهای بورد FSB 4](#_Toc481078536)

[1-1-4- سایر تعاریف در بورد FSB 5](#_Toc481078537)

[1-1-5- قابلیتهای جانبی بورد FSB 6](#_Toc481078538)

[1-1-6- پارامترهای قابل تغییر در نرمافزار بورد FSB 7](#_Toc481078539)

[1-2- مشکلات نسخه قبلی بورد (بورد FSB200a) 7](#_Toc481078540)

[2- مفاهیم نظری مورد استفاده در پروژه 10](#_Toc481078541)

[2-1- مقدمه 11](#_Toc481078542)

[2-2- بلوک دیاگرام سختافزار بورد 11](#_Toc481078543)

[2-2-1- متمایز کردن بورد FSB300 از بورد FSB310 در بعد سختافزاری 12](#_Toc481078544)

[2-3- بلوک دیاگرام نرمافزار بورد FSB 12](#_Toc481078545)

[2-4- سازوکار تولید صوت در بورد FSB 13](#_Toc481078546)

[2-4-1- قضیه نمونهبرداری و بازتولید سیگنال آنالوگ از نمونههای آن 14](#_Toc481078547)

[2-4-2- تولید صوت در بورد FSB 16](#_Toc481078548)

[2-4-3- تولید نمونههای آنالوگ و فیلترینگ در بورد FSB 17](#_Toc481078549)

[2-4-4- تغییر سطح صدا با تغییر اندازه زمان وظیفه موج PWM 18](#_Toc481078550)

[2-4-5- استفاده از تایمر/کانتر یک برای تولید PWM 19](#_Toc481078551)

[2-5- فشردهسازی فایلهای موسیقی با استفاده از زیرنمونهبرداری 20](#_Toc481078552)

[2-6- استفاده از حافظه فلش برای ذخیره فایلهای wav 21](#_Toc481078553)

[2-6-1- فلش AT45DB321E 22](#_Toc481078554)

[2-6-2- چگونگی خواندن داده از فلشهای فوق 23](#_Toc481078555)

[2-6-3- مشکل ایجاد شده در بوردهای FSB200a به دلیل رعایت نکردن قالب آدرسدهی – تنظیم اندازه صفحات فلش به 528 بایت 25](#_Toc481078556)

[2-6-4- ارتباط SPI و چگونگی خواندن داده از AT45DB321E 26](#_Toc481078557)

[2-6-5- ارتباط میان فرکانس نمونهبرداری صوت، تیک زمانی برنامه و سرعت ارتباط SPI جهت خواندن داده از حافظه فلش 27](#_Toc481078558)

[2-6-6- استفاده از دو حافظه فلش در بورد FSB 29](#_Toc481078559)

[2-7- تعریف سیگنالهای ورودی در بورد FSB 29](#_Toc481078560)

[2-7-1- سیگنالهای کنترلی و چگونگی دریافت آنها در بورد FSB300 29](#_Toc481078561)

[2-7-2- سیگنالهای شمارنده در بورد FSB300 33](#_Toc481078562)

[2-8- مفهوم دیبانس سیگنالهای ورودی 36](#_Toc481078563)

[2-8-1- دیبانس سیگنالهای کنترلی (به غیر از سطح صدا) و شمارنده در بورد FSB300 37](#_Toc481078564)

[2-8-2- دیبانس سیگنال تنظیم سطح صدا/ تغییر آلبوم 37](#_Toc481078565)

[2-9- استفاده از پروتکل I2C برای دریافت سیگنالهای ورودی به صورت سریال در بورد FSB310 37](#_Toc481078566)

[2-9-1- سیگنالهای کنترلی در بورد LCB216 38](#_Toc481078567)

[2-9-2- سیگنال شمارنده در بورد LCB216 39](#_Toc481078568)

[2-9-3- دریافت سیگنالهای کنترلی و شمارنده در بورد FSB310 42](#_Toc481078569)

[2-10- کنترل کیفیت (QC) بورد 43](#_Toc481078570)

[2-10-1- مد تست بورد FSB300 44](#_Toc481078571)

[2-10-2- آزمون QC در بورد FSB310 45](#_Toc481078572)

[2-11- تطبیق بورد بورد FSB300 با ورودیهای AL و AH (مد یادگیری) 45](#_Toc481078573)

[2-11-1- مد یادگیری 46](#_Toc481078574)

[2-11-2- ترتیب اجرای مد تست و مد یادگیری 47](#_Toc481078575)

[3- تولید فایلهای موسیقی 48](#_Toc481078576)

[3-1- مقدمه 49](#_Toc481078577)

[3-2- واسط گرافیکی ویرایش فایلهای موسیقی برای بورد FSB 49](#_Toc481078578)

[3-2-1- شمارهگذاری فایلهای موسیقی مورد استفاده در بورد FSB 52](#_Toc481078579)

[3-3- پردازشها برای تولید فایلهای موسیقی با استفاده از نرمافزار متلب 55](#_Toc481078580)

[3-3-1- کد Music\_Compression 55](#_Toc481078581)

[3-3-2- کد Attach\_Music 57](#_Toc481078582)

[3-4- رمزدار کردن فایلهای موسیقی ذخیره شده بر روی بورد 57](#_Toc481078583)

[4- بخش نرمافزاری 59](#_Toc481078584)

[4-1- مقدمه 60](#_Toc481078585)

[4-2- دیاگرام وضعیت در بورد FSB 61](#_Toc481078586)

[4-2-1- وضعیت اورژانسی (EMR) 62](#_Toc481078587)

[4-2-2- وضعیت موسیقی به علاوه دو (MUS\_2) 62](#_Toc481078588)

[4-2-3- وضعیت سکوت به علاوه یک (SILENCE\_1) 63](#_Toc481078589)

[4-2-4- وضعیت سکوت (SILENCE) 63](#_Toc481078590)

[4-2-5- وضعیت اضافه وزن (OVL) 65](#_Toc481078591)

[4-2-6- وضعیت باز بودن درب (DOB) 65](#_Toc481078592)

[4-2-7- وضعیت موسیقی (MUS) 66](#_Toc481078593)

[4-2-1- وضعیت موسیقی به علاوه یک (MUS\_1) 67](#_Toc481078594)

[4-2-2- وضعیت دیبانس سیگنال شمارنده (DEBOUNCE\_NUM\_SIGNAL) 68](#_Toc481078595)

[4-2-3- وضعیتهای DING و DING DONG 69](#_Toc481078596)

[4-2-4- وضعیت اعلان وضعیت توقف (DECLARE STOP STATUS) 69](#_Toc481078597)

[4-2-5- وضعیتهای CHANGE ALBUM و موسیقی به علاوه سه (MUS\_3) 69](#_Toc481078598)

[4-2-6- وضعیت تست (TEST MODE) و سکوت به علاوه دو (SILENCE\_2) 70](#_Toc481078599)

[4-2-7- وضعیت یادگیری (LEARN\_MODE) 71](#_Toc481078600)

[4-2-8- مد زمان توقف برنامه در وضعیتهای مختلف 72](#_Toc481078601)

[4-3- پیادهسازی نرمافزار بورد (ماژولهای کد) 73](#_Toc481078602)

[4-3-1- هدر General.h 73](#_Toc481078603)

[4-3-2- ماژول Main 76](#_Toc481078604)

[4-3-3- ماژول ورودی/ خروجی بورد 86](#_Toc481078605)

[4-4- ماژول خواندن داده به صورت امن از EEPROM 92](#_Toc481078606)

[4-4-2- ماژول خواندن داده از فلش AT45DB321E 93](#_Toc481078607)

[4-4-3- ماژول تنظیمات اولیه فلش AT45DB321E 99](#_Toc481078608)

[4-4-4- ماژول مد یادگیری در بورد FSB300 101](#_Toc481078609)

[4-4-5- ماژول مد تست در بورد FSB300 102](#_Toc481078610)

[4-4-6- ماژولهای ارتباط I2C 103](#_Toc481078611)

[5- بخش سختافزاری 104](#_Toc481078612)

[5-1- مقدمه 105](#_Toc481078613)

[5-2- سختافزار بورد FSB300 105](#_Toc481078614)

[5-2-1- استفاده از رگولاتور ولتاژ LM1117-3.3 به جای LF33 در تغذیه مدار 106](#_Toc481078615)

[5-2-2- تغییر در چینش ورودیها و حذف جامپر NC و NO 106](#_Toc481078616)

[5-2-3- کاهش تعداد هدرهای پروگرام بورد و حذف جامپرهای خارجی آن 106](#_Toc481078617)

[5-2-4- تقویت کننده توان LM386 107](#_Toc481078618)

[5-2-5- فیلتر (میانگذر) مورد استفاده برای بازتولید سیگنال آنالوگ از PWM 108](#_Toc481078619)

[5-2-6- سیگنالدهی دو چیپ AT45DB321E توسط ارتباط SPI 110](#_Toc481078620)

[5-3- سختافزار بورد FSB310 111](#_Toc481078621)

[مراجع 112](#_Toc481078622)



# تعریف پروژه

## تعریف پروژه بورد سخنگوی طبقات (FSB)

بورد سخنگوی طبقات (FSB)[[1]](#footnote-1) که با نام FSB300 نام­گذاری شده است[[2]](#footnote-2) بوردی است که با دریافت   
سیگنال­های کنترلی[[3]](#footnote-3) و شمارنده[[4]](#footnote-4) (شمارنده طبقات) به صورت موازی (Parallel) از بورد اصلی کنترل کننده آسانسور (بورد LCB)، پیام صوتی مناسب را با توجه به وضعیت فعلی آسانسور پخش می­کند. این بورد یک پخش کننده   
فایل­های موسیقی با فرمت wav می­باشد (wave player) که تنها قابلیت پخش فایل­های با نمونه­های هشت بیتی و فرکانس نمونه­برداری 16 کیلوهرتز را دارد که نرخ بیت 128kbps را نتیجه می­دهد. بورد این قابلیت را دارا می­باشد که با استفاده از تکنیک­های ساده پردازش صوت، کیفیت موسیقی را تا نرخ بیت 256kbps افزایش بدهد.

بورد FSB که با نام FSB310 نام­گذاری شده است از لحاظ ساختاری کاملاً شبیه بورد FSB300   
می­باشد. مهمترین تفاوت این نسخه با FSB300، دریافت تمامی سیگنال­ها (کنترلی و شمارنده) به صورت سریال از بورد LCB است. دریافت سیگنال در این بورد از طریق ارتباط I2C صورت می­پذیرد؛ بنابراین لازم است در هنگام پیاده­سازی بورد LCB به چگونگی پیاده­سازی ارتباط I2C نیز توجه شود[[5]](#footnote-5). بورد FSB310 در اصل برای پیاده­سازی برروی بورد CIB216 طراحی شده است.

تمامی مشخصاتی که در ادامه در مورد بورد FSB خواهد آمد برای هر دو نسخه FSB300 و FSB310 معتبر می­باشد، مگر اینکه به طور صریح به غیر از این مطلب ذکر شود. در صورتی که یک مشخصه برای هر دو بورد یکسان باشد، از عبارت بورد FSB در ادامه استفاده خواهد شد؛ اما برای مشخصه­های متفاوت به صورت صریح نام هر یک از دو بورد خواهد آمد.

بورد­ FSB پایه دارای تنها یک حافظه فلش چهار مگابایتی است که این حافظه بر روی بورد تعبیه شده است و نمی­توان آن را از سخت افزار جدا نمود. برای اضافه کردن موسیقی دلخواه به این بورد لازم است این حافظه با کمک یک پروگرامر برنامه­ریزی شود. بورد همچنین این قابلیت را دارد که بدون ایجاد تغییرات اساسی در آن، یک حافظه چهارمگابایتی داخلی دیگر را نیز به آن اضافه نمود، اما بورد قابلیت اضافه کردن حافظه­های خارجی را ندارد. این مطلب در تضاد با بورد FSB200a (نسخه قبلی بورد FSB) می­باشد که در آن یک حافظه میکرو SD اضافی نیز وجود دارد که می­توان آن را از بورد جدا نموده و موسیقی دلخواه را با استفاده از کامپیوتر بر روی آن قرار داد.

یکی از مهمترین خواص نرم­افزار نوشته شده برای بورد، تطبیق یافتن آن با سخت افزار بورد FSB300 یا FSB310 بدون نیاز به هیچگونه تغییرات در آن است. این قابلیت به بورد اجازه می­دهد که با شناسایی سخت­افزار، پروتکل ارتباطی خود را از حالت موازی به I2C تغییر بدهد. همچنین نرم­افزار بورد قابلیت تطبیق­دهی خود با سایر تغییرات خارجی (تغییرات در موسیقی، سیگنال­های ورودی و ...) را دارد که در ادامه در مورد آن بحث خواهد شد.

### مدهای کاری بورد FSB

علاوه بر مد اصلی کاری بورد که در آن پیام­های هشدار، موسیقی، شماره طبقات و ... اعلان می­شود، بورد دارای سه مد کاری دیگر نیز می­باشد:

* **مد تنها دینگ:** منظور از این مد، حالتی می­باشد که بورد تنها پیام­ها و صدای دینگ/ دینگ دانگ را پخش می­کند و هیچگونه موسیقی پخش نمی­شود. برای تعریف این مد در بورد FSB، باید دست کم یک آلبوم خالی (فاقد موسیقی) تعریف و در بورد انتخاب شود. در صورتی که پخش موسیقی حالت اورژانسی نیز مورد نظر نباشد، این موسیقی نیز نباید در بورد پروگرام شود.
* **مد یادگیری:** مد یادگیری تنها مخصوص بورد FSB300 است. هدف از مد یادگیری در بورد، تطبیق دادن نرم­افزار بورد با سطح منطقی سیگنال­های باز بودن درب و trigger است. در این مد (که تنها در POR[[6]](#footnote-6)­ قابل دسترسی است) بورد برای مدت زمان مشخصی سیگنال­های کنترلی را (بدون اعمال هیچگونه ورودی) می­خواند و بر این اساس در مورد سطح منطقی آنها تصمیم­گیری می­کند. با استفاده از مد یادگیری می­توان سطح سیگنال­ها را در نرم­افزار بدون نیاز به اعمال تغییرات مستقیم در آن اعمال کرد، ضمن اینکه این روش باعث حذف جامپرهای NC و NO در مقایسه با بورد FSB200a   
  می­شود.

**یادآوری:** سطح منطقی تعریف شده برای سیگنال­های کنترلی و شمارنده در بورد FSB310 بر اساس سیگنال­دهی متناظر تعریف شده در بورد LCB216 (پیاده­سازی شده در شرکت صانیک) است. به این ترتیب تغییر در تعاریف سیگنال­های کنترلی و شمارشی در بورد FSB برای تطبیق دادن آن با سایر بوردهای LCB تنها با تغییر در نرم­افزار بورد FSB ممکن می­باشد[[7]](#footnote-7).

* **مد تست:** مد تست تنها مخصوص بورد FSB300 است. هدف از مد تست مدار، بررسی صحت کارکرد بورد بدون نیاز به پخش تمامی پیام­های موجود در آن از طریق بررسی تمامی ورودی­ها است. در این مد (که تنها در POR قابل دسترسی است)، اعمال یک سیگنال منطقی مخالف با سطح منطقی پیش فرض یک پین مشخص، باعث می­شود یک شماره طبقه منحصر به فرد (متناظر با شماره پینی که تحریک شده است) پخش ­شود. به این ترتیب می­توان از صحت کارکرد سخت­افزار بورد در کنار صحت نحوه پروگرام میکروکنترلر و همینطور حافظه(های) فلش آن اطمینان حاصل کرد.

### سیگنال­های دریافتی در بورد FSB

جدول ‏1‑1، انواع سیگنال­های کنترلی و اولویت آنها را نمایش می­دهد. منظور از اولویت، اهمیت پخش پیام / عملیات متناظر با یک سیگنال­های دریافتی در بورد است و پیام/ عملیات متناظر با سیگنال دارای بالاترین اولویت بدون توجه به سایر پیام­ها انجام/ پخش و تکرار خواهد شد. بنابراین اگر به عنوان مثال بورد در وضعیت trigger باشد، به وجود سیگنال­های باز بودن درب و یا اضافه وزن واکنشی نشان نمی­دهد که این مطلب با توجه به اولویت بالاتر سیگنال trigger نسبت به سایر سیگنال­ها ممکن شده است.

جدول ‏1‑1- سیگنال­های کنترلی و اولویت متناظر

|  |  |
| --- | --- |
| سیگنال کنترلی | اولویت |
| سیگنال اورژانسی | اولویت اول |
| سیگنال تغییر آلبوم/ تغییر سطح صدا | اولویت دوم |
| سیگنال trigger | اولویت سوم |
| سیگنال اضافه وزن | اولویت چهارم |
| سیگنال بازبودن درب | اولویت پنجم |

علاوه بر سیگنال­های کنترلی، سیگنال­های شمارنده نیز وظیفه اطلاع رسانی در مورد موقعیت فعلی آسانسور را دارند. دو مفهوم در ارتباط با حرکت آسانسور در میان طبقات دارای اهمیت است:

* **شماره توقف آسانسور:** منظور از توقف، نقاط ایستا شدن آسانسور پس از حرکت است. مفهوم توقف فراتر از مفهوم طبقه است و می­تواند (به عنوان مثال) نیم طبقه را نیز پوشش بدهد.
* **وضعیت توقف فعلی:** منظور از وضعیت توقف فعلی، اعلانی است که به این توقف نسبت داده شده است که می­تواند شامل لابی، همکف، زیرزمین، طبقه اول و ... باشد.

طبق تعریف سیگنال شمارنده باید حاوی شماره توقف و همینطور وضعیت آن باشد. اما تمامی بورد­های CIB لزوماً هر دو این اطلاعات را ارسال نمی­کنند و بنابراین تعاریف در دو مدل بورد FSB تفاوت دارند. بورد FSB310 سیگنال­های شمارنده شامل سیگنال شماره توقف فعلی و سیگنال وضعیت آن توقف را از بورد LCB216 دریافت می­کند. در بورد LCB216 می­توان به هر شماره توقف یک وضعیت را نسبت داد که این وضعیت توسط بورد FSB310 پخش   
می­شود. اما بورد FSB300 قابلیت دریافت شماره توقف را ندارد و تنها وضعیت توقف در این بورد دریافت می­شود. به همین دلیل در بورد FSB300، به هر سیگنال شمارنده دریافتی یک شماره توقف در نرم­افزار نسبت داده شده است. به عنوان مثال پارکینگ نه به عنوان اولین توقف، لابی و همکف به عنوان توقف دوازدهم، و طبقه بیست و نهم به عنوان توقف 31 تعریف شده­اند (به جدول ‏2‑6 رجوع شود). بنابراین در صورتی که قرار باشد از وضعیت طبقاتی به غیر از وضعیت­های پیش فرض تعریف شده در جدول ‏2‑6 استفاده شود (مثلاً به جای پارکینگ زیرزمین اعلان شود)، باید چینش وضعیت­ها بر اساس شماره توقف انجام شود.

### وضعیت­های بورد FSB

منظور از وضعیت­ در بورد FSB، یکی از حالت­های زیر می­باشد:

* **وضعیت اورژانسی**: وضعیتی است که در آن تغذیه آسانسور قطع شده و سیگنال 24 ولتی اورژانسی مدار را تغذیه می­کند و سیگنال PDN نیز در بورد FSB300 وجود دارد. این وضعیت در بورد FSB تنها در حالت آغاز به کار مجدد بورد (POR) و تنها یکبار اتفاق می­افتد. در این وضعیت باید ابتدا پیام اورژانسی پخش شود و پس از آن یک موسیقی (آرامش­ بخش) پخش شده و سپس سکوت برقرار شود. خروج از این وضعیت تنها در صورتی ممکن است که سیگنال PDN قطع شود.
* **وضعیت trigger**: وضعیت trigger با دریافت سیگنال trigger (شروع حرکت آسانسور) آغاز می­شود. در این وضعیت موسیقی توسط بورد پخش می­شود. پخش موسیقی با قطع سیگنال trigger متوقف خواهد شد.
* **وضعیت اضافه وزن**: این وضعیت تنها در حالت توقف آسانسور و در صورت وجود سیگنال اضافه وزن رخ   
  می­دهد. در صورت وجود سیگنال اضافه وزن پس از یک مدت زمان معین (که در تمام این مدت بورد هیچگونه پیامی را پخش نمی­کند)، پیام اضافه وزن به صورت کامل پخش می­شود.
* **وضعیت باز بودن درب**: این وضعیت تنها در حالت توقف آسانسور و در صورت وجود سیگنال بازبودن درب رخ می­دهد. در صورت وجود سیگنال بازبودن درب پس از یک مدت زمان معین (که در تمام این مدت بورد هیچگونه پیامی را پخش نمی­کند)، پیام بازبودن درب به صورت کامل پخش می­شود. با توجه به اولویت سیگنال­دهی­ها، پخش پیام اضافه وزن بر پیام بازبودن درب اولویت دارد و در صورت وجود هر دو سیگنال، تنها پیام اضافه وزن پخش خواهد شد.
* **وضعیت توقف­**: این وضعیت در پی قطع شدن سیگنال trigger و بلافاصله پس از وضعیت trigger رخ می­دهد و در آن وضعیت توقف فعلی با توجه به سیگنال­های شمارنده اعلان خواهد شد. قبل از اعلان وضعیت توقف، صدای دینگ و یا دینگ دانگ نیز پخش می­شود که دینگ به معنای افزایش شماره توقف و دینگ دانگ به معنای کاهش آن خواهد بود.
* **وضعیت سکوت**: وضعیتی است که در آن هیچ سیگنال (با مفهومی) در بورد دریافت نشود. در این حالت هیچ صوتی توسط بورد پخش نمی­شود و سیگنال­های ورودی مرتباً بررسی می­شوند.
* **وضعیت تغییر آلبوم**: این وضعیت در زمان پخش موسیقی (وضعیت trigger) و یا سکوت بورد و پس از فشردن همزمان دو کلید فشاری بورد رخ می­دهد. هدف از این وضعیت، تغییر آلبوم موسیقی مورد استفاده در بورد می­باشد. در صورتی که بورد وارد وضعیت تغییر آلبوم شود، بورد بدون توجه به سایر سیگنال­های ورودی شروع به پخش آلبوم بعدی می­کند و پخش آلبوم تا انتهای آن ادامه دارد. در صورتی که در حین پخش آلبوم مجدداً دو کلید برای تغییر آلبوم فشرده نشود، آلبوم فعلی به عنوان آلبوم موسیقی مورد استفاده در بورد انتخاب خواهد شد؛ در غیر اینصورت، با فشردن کلیدها آلبوم عوض خواهد شد و مجدداً پخش آلبوم جدید از ابتدا آغاز خواهد شد.

توجه داریم پخش تمامی پیام­ها و صوت طبقه­ها (غیر از موسیقی) به گونه­ای است که حتی در صورت قطع شدن سیگنال متناظر، پخش صوت متناظر تا انتها ادامه خواهد داشت.

### سایر تعاریف در بورد FSB

سایر تعاریف در بورد FSB به شرح زیر می­باشند:

* **آلبوم:** منظور از یک آلبوم، مجموعه یک یا چند موسیقی می­باشد که این موسیقی­ها در وضعیت trigger پخش می­شوند. در بورد FSB سه آلبوم تعریف شده است که این آلبوم­ها با فشردن همزمان دو کلید فشاری قابل تغییر هستند.
* **تعداد طبقات پشتیبانی شده در بورد:** بورد FSB300 می­تواند تا 19 طبقه بالای همکف، 10 پارکینگ، لابی، همکف و زیرزمین را پوشش دهد. بورد FSB3100 می­تواند تا 121 توقف مختلف را پوشش بدهد و تعداد وضعیت­های توقفی که در شرایط فعلی توسط این بورد پوشش داده می­شود شامل 73 وضعیت مختلف است. توجه داریم با اعمال تغییرات مناسب در نرم­افزار این مقادیر را می­توان تا 141 عدد نیز رساند.
* **بلندگوی مورد استفاده در بورد:** با توجه به IC تقویت توان LM386 مورد استفاده در بورد، بهتر است بلندگوی مورد استفاده در بورد یک بلندگوی 0.5W و باشد.
* **چگونگی تغییر سطح صدا در بورد:** تغییر سطح صدا در بورد FSB با فشردن کلیدهای فشاری برروی بورد انجام می­شود و از پتانسیومتر استفاده نمی­شود. پس از فشردن کلیدها، افزایش و یا کاهش صدا تنها پس از رها شدن کلید متناظر رخ می­دهد. شش سطح صدا در بورد تعریف شده است.

### قابلیت­های جانبی بورد FSB

موارد زیر به عنوان قابلیت­های جانبی بورد FSB تعریف می­شوند. منظور از آنلاین یا آفلاین بودن یک خاصیت، امکان تنظیم آن خاصیت در حین کارکرد بورد (آنلاین) یا نیاز به خدمات جانبی (آفلاین) است:

* **قابل تغییر بودن محتوای پیام­ها و موسیقی (آنلاین و آفلاین)**: بورد به گونه­ای طراحی شده است که بنا به خواست مشتری بتوان محتوای پیام­ها، پیام­های وضعیت توقف و نوع موسیقی را تعویض کرد. این تعویض در بورد FSB به صورت آفلاین قابل پیاده­سازی می­باشد (به این مفهوم که لازم است یکبار موسیقی­ها بر روی حافظه پروگرام شوند)، اما تعویض آلبوم­های موسیقی قرار داده شده در حافظه فلش به صورت آنلاین ممکن می­باشد.
* **قابلیت تنظیم میزان سکوت قبل و بعد از پخش پیام­های هشدار (آفلاین)**: این قابلیت به بورد اجازه می­دهد تا فرکانس تکرار یک پیام مشخص (مثل باز بودن درب) را به صورت مناسبی تنظیم کند. برای این کار لازم است ثابت تکرار آن پیام در هدر Genera.h نرم­افزار تغییر کند[[8]](#footnote-8).
* **سازگار کردن بورد با ورودی­های باز بودن درب و trigger با منطق دلخواه (آفلاین):** این خاصیت مخصوص بورد FSB300 می­باشد. نرم افزار بورد به گونه­ای نوشته شده است که در صورت تغییر سطح منطقی ورودی­ها (در بورد فرمان)، بورد هیچ گونه مشکل تطبیق­پذیری وجود نداشته باشد. برای تطبیق دادن بورد لازم است علاوه بر تنظیم سطح سیگنال­ها در مد یادگیری، جامپرهای تعبیه شده بر روی بورد نیز به صورت مناسبی تنظیم شوند.
* **دارا بودن بسته­ نرم­افزاری برای تولید فایل­های موسیقی منطبق با بورد (آفلاین):** بورد FSB قابلیت پخش تنها یک فرمت موسیقی را دارد (که در شرایط فعلی فرمت wav با نمونه­های هشت­بیتی و فرکانس نمونه­برداری 16 کیلوهرتزی است که نرخ بیت 128kbps را نتیجه می­دهد). بنابراین یک بسته نرم­افزاری در کنار بورد اصلی تعریف شده است که می­تواند فایل­های موسیقی دلخواه را به قالب مورد استفاده در بورد FSB تبدیل کند.

**یادآوری:** این بسته نرم­افزاری برای هر دو بورد FSB300 و FSB310 قابل استفاده است که برای توضیحات بیشتر به فصل سوم این گزارش رجوع شود.

### پارامترهای قابل تغییر در نرم­افزار بورد FSB

علاوه بر موسیقی مورد استفاده در بورد FSB که می­توان آن را به صورت دلخواه توسط نرم­افزار جانبی همراه با بورد تنظیم نمود، بعضی دیگر از پارامترهای مرتبط با تنظیمات داخلی بورد نیز به صورت آفلاین و در نرم­افزار بورد قابل تنظیم مجدد می­باشند. مهمترین پارامترهای قابل تنظیم عبارتند از:

* زمان بالا آمدن نرم­افزار
* تاخیر زمانی تکرار پیام باز بودن درب
* تاخیر زمانی تکرار پیام اضافه وزن
* مدت زمان پخش موسیقی پس از رسیدن به یک توقف
* زمان تخصیص یافته به دیبانس سیگنال­های شمارنده[[9]](#footnote-9)
* آدرس محل ذخیره سطح صدای بورد، وضعیت دارای خطا قبل از ریست شدن بورد، شماره آلبوم، سطح منطقی سیگنال­های کنترلی
* سطح منطقی تعریف شده برای تمامی سیگنال­های شمارنده
* سطح منطقی تعریف شده برای تمامی سیگنال­های کنترلی
* تعداد دفعات دیبانس سیگنال کنترلی، سیگنال شمارنده، سیگنال­های تنظیم سطح صدا (تنها در بورد FSB300)
* تعداد دفعات دیبانس سیگنال­های باز بودن درب و trigger در مد آموزش (تنها در بورد FSB300)

تمامی این پارامترها در هدر General.h (بخش­ ‏4-3-1-) و یا هدر IO\_Module.h (بخش ‏4-3-3-) نرم­افزار بورد FSB قابل دسترسی می­باشند.

## مشکلات نسخه قبلی بورد (بورد FSB200a)

به صورت خلاصه، مشکلات بورد FSB قبلی (بورد FSB200a) شامل موارد زیر است:

1. نرم­افزار قبلی نوشته شده برای بورد دارای موارد ابهام بسیار زیاد، پارامترهای فاقد استفاده و سایر مواردی است که آن را عملاً ناخوانا کرده و در نتیجه پیاده­سازی اصلاحات مورد نیاز در نسخه­های بعدی را بر روی این کد عملاً ناممکن می­سازد.
2. در بورد قبلی امکان پخش دینگ و یا دینگ دانگ بسته به افزایش و یا کاهش شماره توقف وجود ندارد.
3. در بورد FSB200a امکان تعویض موسیقی طبقات به تنهایی وجود ندارد. در واقع یکی از مهمترین دلایلی که از حافظه میکرو SD استفاده شده است این مطلب است که به مشتری اجازه داده شود تا تنها موسیقی میان طبقات را تعویض کند، در حالی که به عنوان مثال شماره طبقات، پیام­ها و غیره بدون تغییر باقی  
   می­مانند. اما چنین امکانی در FSB200a وجود ندارد و لازم است برای تمامی صوت­های پخش شده از بورد یک فایل متناظر فراهم شود. این مشکل در بورد نسخه جدید به این صورت حل می­شود که سه آلبوم موسیقی مستقل در حافظه سوار شده بر روی بورد تعریف می­شوند و امکان تعویض آنها با استفاده از کلیدهای آلبوم وجود دارد. عوض کردن این آلبوم­های موسیقی بدون تعویض صدای طبقات و پیام­ها خواهد بود.
4. از مهمترین مشکلات نسخه قبلی، عدم امکان اصلاح اندازه صفحات حافظه فلش است که باعث بروز مشکلات زیادی در تولید بورد شده بود[[10]](#footnote-10). این مشکل در نسخه فعلی با استفاده از سازوکارهای مناسب مرتفع شده است.
5. استفاده از ولوم آنالوگ در بورد باعث ایجاد هزینه اضافی برای تولید بورد در نسخه قبلی شده است. ضمن اینکه ولوم دیجیتال موجود در بورد دارای تنها سه سطح صدا می­باشد که کافی نیست. در نسخه جدید تنها از ولوم دیجیتالی استفاده می­شود که دارای تعداد سطوح تنظیم صوت بیشتری است.
6. فیلتر آنالوگ مورد استفاده در خروجی PWM برای ساخت DAC مورد استفاده در بورد از ساختارهای استاندارد فیلترهای غیرفعال (passive) استفاده نمی­کند. ضمن اینکه ارتباط میان فرکانس نمونه­برداری صوت و پهنای باند فیلتر به خوبی رعایت نشده است.
7. استفاده از کریستال 16 مگاهرتزی برای تامین فرکانس پالس میکروکنترلر غیر ضروری بوده و این خود باعث ایجاد مشکلاتی در هنگام تولید بورد شده است. در نسخه فعلی از اسیلاتور داخلی میکرو استفاده   
   می­شود که این امر تاثیری بر کارکرد بورد ندارد.
8. عدم استفاده از روش­های فشرده­سازی سیگنال صوت (Music compression) منجر به عدم استفاده بهینه از حافظه فلش در بورد نسخه قبلی شده است. برای مقایسه، استفاده از تکنیک­های Upsampling و downsampling باعث کاهش نصف شدن حجم فایل­های صوتی یکسان در بورد جدید در مقایسه با بورد قبلی می­شود.
9. تعداد هدرهای مورد استفاده در بورد (14 عدد) بیش از حد زیاد بوده، ضمن اینکه نصب جامپر برای اتصال آنها به یکدیگر نیز مشکلات خاص خود را ایجاد می­کند. در بورد نسخه جدید تعداد این هدرها به پنج کاهش یافته است و از جامپر نیز استفاده نمی­شود.
10. نیاز به زمان قابل توجه برای پروگرام بورد، نصب جامپرها و تست کیفیت.
11. موارد زیر از مشکلات بورد FSB200a در زمان پخش صوت­های مختلف است:

* به صورت کلی پخش پیام­ها به وجود و یا عدم وجود سیگنال متناظر آنها وابسته است. به عنوان مثال اگر پیام اورژانسی در حال پخش باشد و سیگنال آن حتی برای لحظه­ای قطع شود، پخش پیام آن متوقف خواهد شد و با وصل شدن سیگنال مجدداً آغاز می­شود. لازم است این وابستگی به گونه­ای رفع شود که پخش یک پیام به صورت کامل انجام شود و پس از آن مجدداً وجود سیگنال متناظر بررسی شود.
* وضعیت اورژانسی بدون توجه به POR­ و در هر زمانی که سیگنال اورژانسی به عنوان ورودی به بورد داده شود برقرار می­شود. در بورد جدید وضعیت اورژانسی تنها می­تواند در زمان روشن شدن بورد رخ بدهد.
* اگر بورد در وضعیت اورژانسی باشد و قبل از آن در یک وضعیت معین (مثل اضافه وزن) و در حال پخش پیام آن باشد، پس از قطع سیگنال اورژانسی پخش پیام قبلی از سر گرفته خواهد شد[[11]](#footnote-11).
* قبل و پس از پخش پیام­های اضافه وزن و باز بودن درب، در بورد موسیقی پخش می­شود که مطلوب نمی­باشد.
* مدت زمان توقف پخش موسیقی پس از اعلان وضعیت توقف معین نیست و بعضاً بسیار طولانی   
  می­باشد.
* در صورتی که بورد در وضعیت trigger قرار داشته باشد (یعنی آسانسور در حال حرکت باشد)، پیام باز بودن درب و یا اضافه وزن در صورت دریافت سیگنال­ها پخش می­شود. این مورد یکی از مهمترین مشکلات بورد فعلی می­باشد، زیرا به عنوان مثال سیگنال باز بودن درب ممکن است به دلایل مختلف (از جمله قرار گرفتن یکی از مسافران در سر راه سنسور آن) قطع شود و پخش پیام باز بودن درب در حین حرکت معنادار نخواهد بود.
* اگر سیگنال trigger حتی برای یک لحظه­ کوتاه در بورد قطع و وصل شود، پخش موسیقی متوقف می­شود که این امر مطلوب نیست.



# مفاهیم نظری مورد استفاده در پروژه

## مقدمه

در این فصل مفاهیم نظری و تعاریف مورد استفاده در بورد FSB ارائه می­شود. این موارد شامل ارائه بلوک دیاگرام سخت­افزار بورد(ها)، بلوک دیاگرام نرم­افزاری، معرفی کامل سیگنال­های ورودی، معرفی وضعیت­های بورد، سازوکار تولید صوت توسط بورد، فیلتر کردن سیگنال­ها در بورد، حافظه جانبی فلش مورد استفاده در بورد و نحوه ارتباط­گیری میکروکنترلر با این حافظه است. از بسیاری از مفاهیم ارائه شده در این فصل به صورت مستقیم در پیاده­سازی بورد استفاده می­شود، اگر چه در موارد کمی این اطلاعات تنها به منظور کامل بودن مفاهیم ارائه شده در یک موضوع آورده شده­اند و لزوماً در نسخه فعلی مورد استفاده قرار نمی­گیرند. بسیاری از این مفاهیم در هر دو نسخه FSB300 و FSB310 یکسان می­باشند که در این حالت تمایزی میان این دو نسخه بورد ایجاد نمی­شود. در غیر این ­صورت به صورت صریح بیان خواهد شد که یک خاصیت متعلق به کدام بورد است.

## بلوک دیاگرام سخت­افزار بورد

به منظور منسجم ساختن مفاهیم ارائه شده در این گزارش، در این قسمت به صورت خلاصه در مورد بلوک دیاگرام سخت افزار بورد بحث می­شود و بحث کامل­تر در مورد مفاهیم به فصل پنجم واگذار می­شود.

بلوک دیاگرام سخت افزار بورد FSB300 در شکل ‏2‑1-الف آمده است. همانگونه که مشاهده می­شود بورد از شش بلوک اصلی تشکیل شده است که شامل پورت­های دریافت سیگنال، بلوک پردازنده ATMEGA8، حافظه فلش، فیلتر میان­گذر و در نهایت تقویت کننده توان LM386 می­باشد که این بلوک به صورت مستقیم با بلندگوی بورد در ارتباط می­باشد. دو بلوک دریافت سیگنال­های شمارنده و کنترلی به ترتیب با پورت­های D و C میکرو در ارتباط هستند. ارتباط میان حافظه فلش و پردازنده نیز یک ارتباط SPI است که در ادامه این فصل در مورد آن توضیح داده خواهد شد. برای تولید صدا از نمونه­های دیجیتال ذخیره شده در پردازنده لازم است یک DAC در بورد پیاده­سازی شود که اینکار با استفاده از وقفه­های داخلی میکرو و یک فیلتر میان­گذر پیاده­سازی می­شود که در مورد آن در بخش 2-4-2 بحث خواهد شد. در نهایت از آنجا که معمولاً سطح توان سیگنال در خروجی فیلتر پایین می­باشد، از IC تقویت کننده توان LM386 قبل از بلندگو استفاده شده است که چگونگی راه­اندازی آن در فصل پنجم خواهد آمد.

بلوک دیاگرام سخت­افزار FSB310 در شکل ‏2‑1-ب آمده است. همانگونه که مشاهده می­شود تنها تفاوت این بلوک دیاگرام با شکل الف، دریافت سیگنال­ها به صورت سریال (و از طریق ارتباط I2C) است. در سایر موارد اما بلوک دیاگرام کاملاً مشابه بلوک دیاگرام بورد FSB300 است.



(الف) بورد FSB300



(ب) بورد FSB310

شکل ‏2‑1: بلوک دیاگرام سخت­افزار بورد FSB

### متمایز کردن بورد FSB300 از بورد FSB310 در بعد سخت­افزاری

به جهت اینکه بتوان تنها از یک نرم­افزار برای هر دو بورد FSB300 و FSB310 استفاده کرد، یک وجه تمایز مشخص سخت­افزاری میان این دو بورد ایجاد شده است. در واقع مقاومت Pull-up داخلی پورت PB6 میکروکنترلر در هر دو بورد فعال می­شود؛ اما در بورد FSB300 این پایه با یک مقاومت Pull-down زمین می­شود. به این ترتیب نرم­افزار بورد با سرکشی به این پورت می­تواند تعیین کند که سخت­افزار متعلق به بورد FSB300 یا FSB310 است و از این طریق خود را با سخت­افزار تطبیق دهد.

## بلوک دیاگرام نرم­افزار بورد FSB

بلوک دیاگرام نرم­افزار بورد FSB در بالاترین سطح در شکل ‏2‑2 آمده است. همانگونه که مشاهده می­شود نرم افزار شامل پنج بخش کلی تنظمیات اولیه، تشخیص سیگنال ورودی، تعیین وضعیت برمبنای سیگنال­های ورودی، خواندن داده از فلش مموری و پخش صوت می­باشد. ضمن اینکه پخش صوت نیز به صورت همروند[[12]](#footnote-12) با سایر   
قسمت­های حلقه اصلی برنامه و توسط یک وقفه انجام می­شود که این امر اهمیت زمان­بندی قسمت­های مختلف برنامه را دو چندان می­کند. پیاده­سازی کد نرم­افزاری بر مبنای گزارش «*استانداردسازی نوشتار کد*» صورت پذیرفته است که توصیه می­شود قبل از بررسی جزئیات برنامه، مطالب این گزارش نیز مورد بررسی قرار بگیرد.

**یادآوری:** نرم­افزار نوشته شده برای هر دو بورد FSB300 و FSB310 به صورت مشترک استفاده می­شود و تنها در بلوک تشخیص سیگنال ورودی تفاوت دارد. این خاصیت باعث می­شود که این نرم­افزار به صورت نصب و اجرا (Plug & Play) باشد.

**یادآوری:** منظور از تیک زمانی برنامه (برنامه میکروکنترلر) در ادامه این گزارش، مدت زمانی است که برنامه در حلقه اصلی سپری می­کند قبل از آنکه وقفه بررسی وضعیت برنامه (یعنی وضعیت­های WAIT، BUSY، READY در حلقه اصلی) صورت پذیرد. در بوردهای شرکت صانیک معمولاً میزان این وقفه برابر 10   
میلی­ثانیه در نظر گرفته می­شود، اما در بورد FSB (همانگونه که در بخش 2-6-4 توضیح داده خواهد شد) این مقدار برابر 16 میلی­ثانیه است. در گزارش «*استانداردسازی نوشتار کد*» به تفضیل در مورد تیک زمانی برنامه بحث شده است.



شکل ‏2‑2: بلوک دیاگرام نرم­افزار بورد FSB

برای درک بهتر محتوای نرم­افزار و همینطور چگونگی ارتباط آن با سخت افزار برنامه، لازم است مفاهیم اولیه مورد استفاده در بورد FSB مورد بررسی قرار بگیرد. این مفاهیم در ادامه این فصل ارائه خواهند شد.

## سازوکار تولید صوت در بورد FSB

در این بخش چگونگی ایجاد صوت در بورد FSB با استفاده از داده­های ذخیره شده در حافظه تشریح می­شود. همانگونه که می­دانیم، پس از ذخیره یک فایل صوتی '.wav' درون حافظه دیجیتال، تنها نمونه­های صوت در حافظه ذخیره می­شوند و تولید مجدد صوت از این نمونه­ها نیازمند تولید یک سیگنال آنالوگ است. برای درک چگونگی تولید این سیگنال آنالوگ لازم است ابتدا توضیح اجمالی در مورد قضیه نمونه­برداری ارائه گردد و سپس سازوکار استفاده شده برای تولید موسیقی در بورد تشریح شود.

### قضیه نمونه­برداری و بازتولید سیگنال آنالوگ از نمونه­های آن

منظور از نمونه­برداری از یک سیگنال آنالوگ، ثبت مقدار سیگنال در لحظات مشخصی از زمان می­باشد. هدف از نمونه­برداری، ذخیره اطلاعات سیگنال در یک فضای محدود جهت بازسازی مجدد آن بدون نیاز به ثبت سیگنال آنالوگ در تمامی لحظات است (که برای ذخیره آن یک حافظه با اندازه بی­نهایت مورد نیاز می­­باشد).

نمونه­برداری از سیگنال آنالوگ با استفاده از A/D مناسب، در فواصل زمانی یکسان و با رعایت نرخ نایکوئیست صورت می­پذیرد. اگر فرض کنیم بالاترین فرکانس موجود در طیف فرکانسی سیگنال برابر باشد، آنگاه نرخ نایکوئیست ایجاب می­کند که فرکانس نمونه­برداری یا دو برابر باشد. به عبارت دیگر یا فاصله بین   
نمونه­ها از رابطه زیر محاسبه خواهد شد:

|  |  |
| --- | --- |
| (‏2‑1) |  |

پس از استخراج نمونه­های آنالوگ نوبت به ذخیره­سازی آنها به صورت دیجیتال می­رسد. از آنجا که ذخیره­سازی به فرمت بیتی صورت می­پذیرد، ذخیره هر نمونه­ با مقداری خطا انجام خواهد شد که این خطا به نام خطای کوانتیزاسیون معروف است و با تعداد بیت­های A/D (رزولوشن A/D) نسبت عکس دارد.

**یادآوری:** همانگونه که می­دانیم سیگنال صوت بر اثر فشار هوا ایجاد می­شود و این سیگنال با تغییر در به عنوان مثال صفحات خازن یک میکروفون به یک سیگنال الکتریکی تبدیل می­شود. بنابراین می­توان برای نمونه­های سیگنال الکتریکی صوت تنها مقادیر مثبت و یا مقادیر مثبت و منفی در نظر گرفت. این امر مشکلی در هنگام بازسازی صوت ایجاد نمی­کند. معمولاً صوت 8 بیتی تنها با مقادیر مثبت و صوت 16 بیتی با مقادیر مثبت و منفی (مکمل 2) نمایش داده می­شود.

اگر فرض کنیم سیگنال گسسته متناظر با نمونه­های دیجیتال با نمایش داده شود، اگر طیف فرکانسی سیگنال آنالوگ یا را به صورت شکل ‏2‑3-الف در نظر بگیریم، طیف فرکانسی سیگنال دیجیتال به صورت شکل ‏2‑3-ب خواهد شد.



(الف) طیف فرکانسی سیگنال آنالوگ



(ب) طیف سیگنال دیجیتال

شکل ‏2‑3: مقایسه طیف سیگنال­ آنالوگ و سیگنال نمونه­برداری شده

بازسازی سیگنال از نمونه­ها: به صورت نظری، بازسازی سیگنال آنالوگ از نمونه­های دیجیتال با فیلتر کردن سیگنال دیجیتال با استفاده از یک فیلتر آنالوگ ایده­آل صورت می­پذیرد. اگر فرض کنیم فرکانس قطع فیلتر آنالوگ پایین­گذر ایده­آل برابر است که ، آنگاه طیف فرکانسی سیگنال خروجی به صورت ارائه شده در شکل ‏2‑7 خواهد شد که دقیقاً برابر طیف سیگنال آنالوگ است. در نتیجه شکل سیگنال در حوزه زمان نیز با شکل سیگنال آنالوگ اصلی یکسان خواهد شد.



شکل ‏2‑4: بازسازی سیگنال آنالوگ اصلی با استفاده از نمونه­های آن

معمولاً در منابع توصیه شده است که انتخاب شود که انتخاب دوم از این بابت اهمیت دارد که باعث می­شود نویز ورودی به سیستم افزایش پیدا نکند. توجه داریم اگر فیلتر آنالوگ مورد استفاده برای بازسازی سیگنال ایده­آل نباشد (که این امر اجتناب ناپذیر می­باشد)، آنگاه بازسازی سیگنال نیز دارای خطا خواهد شد. بنابراین دو عامل:

* ذخیره کردن نمونه­های سیگنال به صورت دیجیتال
* ایده­آل نبودن فیلتر پایین­گذر مورد استفاده برای بازسازی سیگنال

عوامل اصلی ایجاد خطا در نمونه­برداری و بازسازی سیگنال می­باشند. بنابراین نرخ قطع فیلتر (Cut-off rate) در حوزه فرکانس نیز اهمیت زیادی دارد و هرچه مقدار آن بالاتر باشد (یعنی گذار فیلتر از بهره بالا به بهره بسیار پایین در حوزه فرکانس سریعتر باشد و فیلتر به شکل مربعی در حوزه فرکانس نزدیک­تر باشد)، شکل موج خروجی فرم بهتری خواهد داشت.

از دیدگاه عملی برای بازتولید سیگنال از نمونه­های آنالوگ به این صورت عمل می­شود که ابتدا پالس­هایی با دامنه برابر دامنه نمونه­های سیگنال و با عرضی برابر دوره نمونه­برداری تولید می­شوند و سپس از فیلتر پایین­گذر عبور داده می­شوند (شکل ‏2‑5). در این صورت سیگنال خروجی فیلتر مشابه سیگنال آنالوگ اصلی می­باشد.



شکل ‏2‑5: بازسازی سیگنال آنالوگ اصلی با استفاده از نمونه­های آن

### تولید صوت در بورد FSB

تولید صوت در بورد FSB مشابه روش ارائه شده در بخش قبل و با استفاده از قضیه نمونه­برداری صورت   
می­پذیرد. در واقع فایل '.wav' که در فلش مموری بورد ذخیره شده است حاوی نمونه­های دیجیتال سیگنال صوت است و بنابراین لازم است تنها مرحله بازسازی سیگنال آنالوگ در بورد صورت بپذیرد. برای تبدیل این نمونه­ها به صوت، لازم است اولاً این نمونه­ها به قالب آنالوگ تبدیل شوند و ثانیاً از فیلتر پایین­گذر مناسبی عبور کنند. در ادامه چگونگی انجام این دو عمل در بورد FSB خواهد آمد. اما قبل از ورود به بحث فوق لازم است مختصری در مورد فایل­های wav توضیح داده شود.

#### فرمت ذخیره­سازی صوت wav

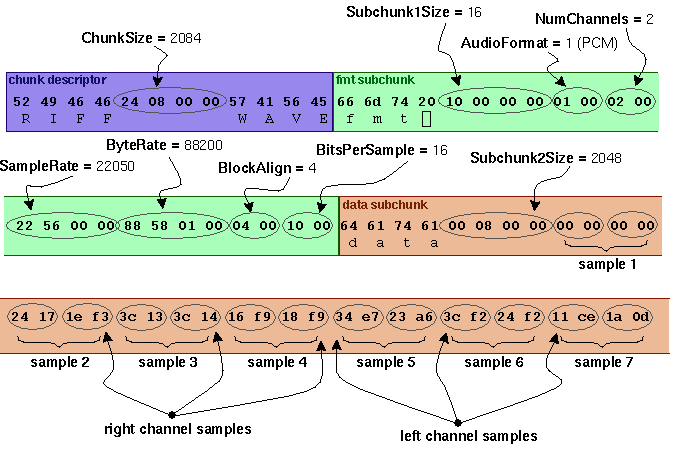
فرمت wav یکی از محبوب­ترین فرمت­های ذخیره­سازی صوت می­باشد. در این فرمت نمونه­های سیگنال صوت به صورت دیجیتال و با اندازه­های 8، 16 و 32 بیتی ذخیره می­شوند. ضمن اینکه فرکانس نمونه­برداری نیز معمولاً شامل مقادیر استاندارد 8، 16، 32 و 64 کیلوهرتز (و یا بیشتر) است. به این ترتیب نرخ بیت (bit-rate یا تعداد بیت­های ذخیره شده در هر ثانیه) بین مقدار 64kbps تا 2048kbps متغیر است.

هر فایل wav استاندارد دارای یک هدر (Header) 44 بایتی است که شامل بعضی اطلاعات مهم در مورد فایل می­باشد. یک هدر استاندارد در شکل ‏2‑6 نمایش داده شده است[[13]](#footnote-13). همانگونه که مشاهده می­شود چهار بیت اول نشان دهنده کد اسکی کلمه RIFF می­باشند. چهار بیت دوم نشان دهنده تعداد بایت­های اطلاعات موجود در این فایل موسیقی می­باشند. این چهار بایت به ترتیب از چپ به راست خوانده می­شوند و به عنوان مثال هدر زیر نشان می­دهد که فایل موسیقی متناظر دارای:

|  |  |
| --- | --- |
| (‏2‑2) |  |

اطلاعات است. حال اگر فرض کنیم موسیقی در قالب هشت بیتی نیز ذخیره شده باشد، آنگاه تعداد نمونه­های ذخیره شده برابر 2048 می­باشد، اما اگر تعداد بیت­ها به ازای نمونه برابر 16 باشد، آنگاه سیگنال دارای 1024 نمونه است.

در بورد FSB ترجیج بر استفاده از فایل­های صوت با 8 بیت به ازای نمونه است که فضای ذخیره­سازی را کمینه می­کنند. بنابراین تعداد نمونه­های هر فایل صوت برابر عدد خوانده شده از این هشت بایت هستند. البته می­توان تعداد نمونه­ها را با استفاده از نرم­افزارهای مناسب (مثل MATLAB) و بدون استفاده از این اطلاعات نیز محاسبه کرد.



شکل ‏2‑6: هدر استاندارد فایل wav

### تولید نمونه­های آنالوگ و فیلترینگ در بورد FSB

مهمترین مشکلی برای تولید صوت در بورد FSB وجود دارد، ممکن نبودن تولید نمونه­های آنالوگ از مقادیر هشت بیتی ذخیره شده در حافظه فلش با استفاده از میکروکنترلر است. در واقع میکروکنترلر ATmega8 قابلیت تولید سطح ولتاژهای مختلف را به عنوان خروجی ندارد و باید ساختار جایگزینی را برای این امر استفاده کرد.

برای حل مشکل فوق، می­توان به جای تولید پالس­های با دامنه­های متناسب با اندازه نمونه (شکل ‏2‑4)، عرض پالس و به عبارتی زمان وظیفه (یا DC[[14]](#footnote-14)) را با هر نمونه مرتبط ساخت که این عمل اصطلاحاً مدولاسیون عرض پالس (PWM)[[15]](#footnote-15) نام دارد. در این مدولاسیون، هر چه اندازه نمونه بزرگ­تر باشد به همان نسبت DC نیز بزرگ­تر انتخاب خواهد شد. به طور خاص اگر دامنه برابر (بزرگترین دامنه پس از نمونه­برداری و کوانتیزاسیون) باشد، آنگاه DC بزرگترین مقدار (عموماً 100%) را خواهد داشت. از آنجا که تولید PWM توسط یکی از وقفه­های TIMER/COUNTER میکروکنترلر صورت می­پذیرد، مقدار DC به یک عدد با دقت 8 بیت و یا 16 بیت محدود می­شود.

شکل موج خروجی PWM پس از عبور از فیلتر پایین­گذر در شکل ‏2‑7 آورده شده و با شکل موج شکل ‏2‑4 مقایسه شده است. همانگونه که مشاهده می­شود شکل موج خروجی با تقریب خوبی مشابه شکل موج اصلی برابر شده است و تنها در ریپل­های کوچکی با آن تفاوت دارد.

همانگونه که مشاهده می­شود ارتباط نزدیکی میان فرکانس نمونه­برداری، عرض فیلتر مورد استفاده و همینطور DC وجود دارد و همین امر انتخاب فرکانس نمونه­برداری موسیقی را محدود می­کند. همچنین از آنجا که ترجیح بر استفاده از موسیقی­های با کمترین میزان حجم حافظه است، اندازه نمونه­های موسیقی برابر 8 بیت انتخاب می­شود. بنابراین بورد FSB قابلیت سازگاری تنها با یک قالب فایل wav را دارد. در پیاده­سازی فعلی، نرخ   
نمونه­برداری برابر 16 کیلوهرتز است که با توجه به 8 بیتی بودن نمونه­ها، نرخ بیت 128kbps را نتیجه می­دهد. البته توجه داریم این نرخ مخصوص فایل­های ذخیره شده بر روی بورد است و موسیقی در اصل با نرخ بیت 256kbps در بورد پخش خواهد شد (به بخش 2-4-5 رجوع شود).



شکل ‏2‑7: تولید سیگنال آنالوگ از نمونه­های دیجیتال با استفاده از مدولاسیون عرض پالس و فیلتر پایین­گذر

### تغییر سطح صدا با تغییر اندازه زمان وظیفه موج PWM

پیش از بررسی چگونگی تغییر سطح صدا در بورد FSB، لازم است به این نکته اشاره شود که تغییر صدا به معنای افزایش گستره پویایی[[16]](#footnote-16) سیگنال آنالوگ صوت است. برای توضیح توجه داریم که اگر شکل موج صوت به صورت الف در شکل ‏2‑8 باشد، آنگاه سیگنال ب که در آن سیگنال الف در یک عدد ثابت ضرب شده است دارای سطح صدای بالاتری می­باشد؛ اما توجه داریم سیگنال شکل پ که شیفت یافته سیگنال شکل الف می­باشد دارای سطح صدای بالاتری نمی­باشد، چون گستره پویایی صوت افزایش نیافته و تنها دامنه سیگنال در تمامی لحظات افزایش یافته است. از دید حوزه فرکانس نیز ضرب در دو کردن دامنه سیگنال به معنای دو برابر شدن دامنه سیگنال در تمامی فرکانس­ها می­باشد (افزایش سطح صدا)، در حالی که افزایش سطح سیگنال با یک مقدار ثابت تنها به معنای افزایش دامنه طیف در فرکانس صفر می­باشد که باعث افزایش سطح صدا نخواهد شد.



شکل ‏2‑8: منظور از افزایش سطح صدا، افزایش گستره پویایی سیگنال خروجی است.

همانگونه که در بخش قبل اشاره شد، تولید سیگنال آنالوگ با فیلترینگ موج PWM صورت می­پذیرد. اگر فرض کنیم دامنه سیگنال خروجی تقریباً به صورت خطی با سیگنال ورودی ارتباط دارد، بنابراین دامنه سیگنال در هر دوره به DC در آن دوره بستگی دارد. به این ترتیب اگر به عنوان مثال DC در تمامی دوره­ها در یک مقدار ثابت ضرب شود، آنگاه سطح سیگنال خروجی نیز در تمامی دوره­ها در یک عدد ثابت ضرب می­شود و در نتیجه دامنه سیگنال خروجی افزایش خواهد یافت (مشابه شکل موج ب در شکل ‏2‑8)؛ اما توجه داریم اگر تمامی DCها به اندازه یک عدد ثابت افزایش پیدا کنند، تمامی دامنه­ها با یک عدد ثابت جمع خواهند شد (مشابه شکل موج پ در شکل ‏2‑8) که این مطلب معادل افزایش سطح صوت نمی­باشد.

توجه داریم سازوکار ضرب DCها در یک عدد ثابت، یک راه­حل ساده برای افزایش سطح صدا در بورد می­باشد که دارای مزایا و معایب خاص خود می­باشد. مهمترین مزیت سازوکار فوق، عدم نیاز به پتانسیومتر جهت تغییر سطح صدا می­باشد که احتمال خرابی بالایی در پیاده­سازی و استفاده طولانی مدت دارد. مهمترین عیب روش نیز کم بودن تعداد سطوح ممکن جهت تغییر سطح صدا می­باشد؛ به خصوص اینکه افزایش DC با مقادیر غیر از ضریب دو (یعنی دو برابر، چهار برابر و ...) به دلیل محدودیت­های ذاتی میکروکنترلر و همینطور مبنای دو بودن محاسبات پیچیده می­باشد.

### استفاده از تایمر/کانتر یک برای تولید PWM

برای تولید PWM مورد نیاز پخش صوت می­توان از مد Fast PWM تایمر کانتر یک میکرو استفاده نمود. در این مد، بسته به تنظیمات کاربر، شمارنده شروع به شمارش می­کند و هنگامی که مقدار شمارش برابر مقدار قرار داده شده در رجیستر OCR1A شود، یک وقفه تولید می­شود که در آن منطق شکل موج خروجی تغییر می­کند. با تنظیم رجیستری TCCR1A، PWM تولیدی این تایمر/کانتر در پایه OC1A یا OC1B دریافت می­شود که در این پروژه خروجی در پایه OC1A دریافت می­شود. ارتباط میان فرکانس PWM و فرکانس ساعت پردازنده در این مد به صورت زیر تعریف می­شود:

|  |  |
| --- | --- |
| (‏2‑3) |  |

که در آن N مقسم (prescalar) می­باشد که می­­تواند مقادیر 1، 8، 64، 256 و 1024 را اختیار کند و TOP مقداری است که پردازنده قبل از تولید وقفه سرریز تایمر/کانتر می­شمارد. همانطور که می­دانیم شمارنده تولید وقفه در این تایمر/کانتر 16 بیتی است، بنابراین می­توان رزولوشن تایمر کانتر را به صورت هشت بیتی، نه بیتی و یا ده بیتی تنظیم نمود، یا اینکه این مقدار را توسط رجیستر ICR1 به صورت دلخواه تنظیم کرد. برای فراهم کردن آزادی عمل به جهت تغییر فرکانس PWM در نسخه­های آینده نرم­افزار، ترجیح بر استفاده از رجیستر ICR1 است.

همانگونه که در قبل نیز اشاره شد، فرکانس نمونه­برداری فایل­های صوت مورد استفاده در این بورد برابر 32kHz است، بنابراین . از آنجا که برای تامین سیگنال پالس ساعت ترجیح بر استفاده از نوسان­ساز داخلی میکروکنترلر می­باشد، با انتخاب مقادیر زیر:

|  |  |
| --- | --- |
| (‏2‑4) |  |

می­توان به مطلوب دست پیدا کرد. توجه داریم با انتخاب ، به صورت ضمنی از شمارنده هشت بیتی استفاده می­شود و این بدان معناست که مقدار DC موج PWM توسط هشت بیت تنظیم خواهد شد. بنابراین از آنجا که سطح هر نمونه سیگنال صوت نیز برابر هشت بیت است، DC دقیقاً به مقدار نمونه متناظر نگاشت خواهد شد. توجه داریم اگر شمارنده به عنوان مثال تا شمارش می­کرد، آنگاه شمارنده و در نتیجه DC نه بیتی می­شد و بنابراین لازم بود 8 بیت اندازه نمونه به نحوی در این 9 بیت قرار بگیرند (به عنوان مثال از سمت MSB یا LSB هشت بیت را پر کنند).

توجه داریم پخش موسیقی به صورت همروند در کنار اجرای برنامه اصلی میکروکنترلر انجام می­شود. این بدان معناست که در هر بار رخ دادن وقفه تایمر/کانتر، میزان رجیستر OCR1A برابر مقدار نمونه هشت بیتی قرار داده می­شود و پس از اتمام وقفه، میکرو شروع به تولید PWM می­کند و به صورت همزمان برنامه روند عادی خود (یعنی تعیین سیگنال­های ورودی، تعیین وضعیت بورد و غیره) را انجام می­دهد.

## فشرده­سازی فایل­های موسیقی با استفاده از زیرنمونه­برداری

به صورت کلی روش­های فشرده­سازی و ذخیره دیجیتال سیگنال صوت به دو دسته فاقد تلف[[17]](#footnote-17) و دارای تلف[[18]](#footnote-18) طبقه­بندی می­شوند. در روش­های فاقد تلف، نمونه­های سیگنال به صورت مستقیم در حافظه ذخیره می­شوند. معروف­ترین پسوند متناظر با این طبقه نیز فایل­های wav می­باشند. اما در روش­های ذخیره دارای تلف، ممکن است بخش گسترده­ای از نمونه­ها با استفاده از ابزارهای پردازش سیگنال کنار گذاشته شوند و به همین دلیل این روش­ها به روش­های دارای تلف معروف هستند. معروف­ترین پسوند­های متناظر با این طبقه نیز فایل­های mp3، mpeg و .... می­باشند. از آنجا که استفاده از روش­های پیچیده فشرده­سازی به دلیل محدودیت حافظه پردازشی ممکن نمی­باشد، در این پروژه از تکنیک­های ساده زیرنمونه­برداری (downsampling) و نمونه­گذاری (upsampling) استفاده می­شود. این تکنیک فرمت فایل موسیقی مورد استفاده در بورد (یعنی wav) را تغییر نمی­دهد، اما باعث کاهش حجم داده خواهد شد و به همین دلیل جزو روش­های دارای تلف طبقه­بندی می­شود.

منظور از زیرنمونه­برداری، کاهش نرخ نمونه­برداری است که منجر به کاهش تعداد نمونه­های سیگنال در یک بازه زمانی مشخص خواهد شد. اگر فرض کنیم سیگنال آنالوگ با نشان داده شود و سیگنال دیجیتال متناظر با نشان داده شود، و نرخ نمونه­برداری نیز با نمایش داده شود، آنگاه با تعریف که در آن و یک عدد صحیح است، سیگنال دیجیتال زیر نمونه­بردای شده یا به صورت زیر تعریف خواهد شد:

|  |  |
| --- | --- |
| (‏2‑5) |  |

بنابراین استفاده از زیرنمونه­برداری یک روش کاهش تعداد نمونه­ها و در نتیجه کاهش فایل است. در نتیجه با زیرنمونه­برداری از فایل­های موسیقی می­توان حجم آنها را به اندازه قابل توجهی کاهش داد. توجه داریم استفاده از زیرنمونه­برداری علی­رغم وجود افزودگی (redundancy) بسیار زیاد سیگنال موسیقی و نزدیک بودن اندازه دو نمونه متوالی به یکدیگر، در نرخ نمونه­برداری بسیار پایین (کمتر از 32kHz) می­تواند باعث کاهش کیفیت صوت شود که این امر استفاده از آن را محدود می­کند.

یک روش غلبه بر کاهش کیفیت صوت پس از زیر نمونه­برداری، استفاده از خاصیت افزودگی سیگنال صوت و بازتولید نمونه­های حذف شده و از این طریق بازگرداندن نرخ نمونه­برداری به مقدار قبلی است. برای این منظور،   
می­توان سیگنال را که تقریب بسیار خوبی از سیگنال است به صورت زیر مجدداً از سیگنال (که زیر نمونه­برداری شده از سیگنال با نرخ است) تولید کرد:

|  |  |
| --- | --- |
| (‏2‑6) |  |

از آنجا عملیات فوق صرفاً پردازشی است و نمونه­ها تنها با استفاده از یک عملیات ریاضی تولید می­شوند، عملیات نمونه­گذاری مقرون به صرفه می­باشد.

در بورد FSB، یک سیگنال صوت ابتدا با نرخ 2 و توسط بسته نرم­افزاری متلبی که توضیح آن در فصل سوم خواهد آمد زیرنمونه­برداری ­می­شود، سپس در هنگام بازتولید صوت، سیگنال صوت در نرم­افزار با نرخ 2   
نمونه­گذاری می­شود. نمونه­گذاری نیز به این صورت است که هر نمونه میانی حذف شده با نمونه قبلی خود جایگزین می­شود. استفاده از این سازوکار با وجود نصف کردن حجم حافظه مورد استفاده باعث کاهش کیفیت قابل توجه موسیقی پخش شده نمی­شود.

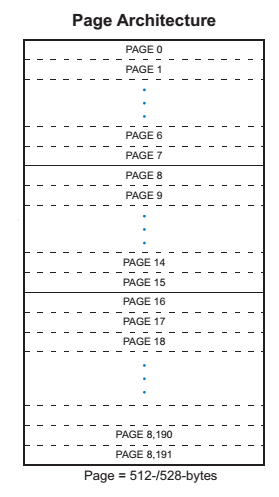
## استفاده از حافظه فلش برای ذخیره فایل­های wav

یکی از ملزومات پیاده­سازی بورد FSB، ذخیره داده­های موسیقی است. برای این منظور از حافظه­های فلش on chip و به طور خاص حافظه AT45DB321E استفاده می­شود. این نوع حافظه دارای اندازه کوچک، دسترسی پذیری و برنامه­ریزی ساده و قیمت نسبتاً مناسبی می­باشد که استفاده از آن را توجیه می­کند. در ادامه مختصری در مورد این حافظه و چگونگی استفاده از آن توسط دستورهای عملیاتی مناسب صحبت خواهد شد، سپس در مورد ارتباط SPI که وسیله برقراری ارتباط میان فلش و میکرو است صحبت خواهد شد.

### فلش AT45DB321E

این فلش قابلیت ذخیره داده تا چهار مگابیت (4MByte) را دارد. می­توان از ارتباط SPI (که در بخش بعدی توضیح داده می­شود) برای خواندن و نوشتن داده بر روی این فلش­ها استفاده نمود. تغذیه مورد نیاز این فلش 6/3 است.

نحوه تقسیم بندی حافظه در این فلش­ در شکل ‏2‑9 داده شده است:



شکل ‏2‑9: چگونگی ذخیره داده در AT45DB321E

همانگونه که مشاهده می­شود، حافظه به 8191 صفحه (page) با اندازه­های مساوی تقسیم می­شود که این صفحات می­توانند 512 یا 528 بایتی باشند. با تعاریف فوق واضح است که برای آدرس­دهی هر صفحه به 13 بیت احتیاج است. همچنین برای آدرس­دهی هر بایت درون یک صفحه با اندازه 528 بایت، 10 بیت مورد نیاز می­باشد. در مورد صفحات 512 بایتی نیز 9 بیت مورد نیاز است.

به طور پیش فرض اندازه هر صفحه یک IC از نوع فوق 528 بایت است، اما می­توان فلش­هایی را با اندازه صفحه پیش فرض 512 بایت نیز تهیه نمود. در جدول ‏2‑1، دو برچسب IC که می­توان از آنها در بورد FSB استفاده نمود آمده است.

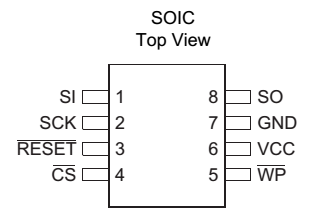
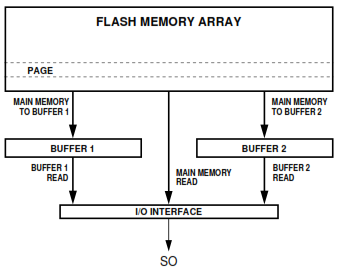
جدول ‏2‑1: AT45DB321E که می­توان از آنها در بورد FSB استفاده نمود

|  |  |  |
| --- | --- | --- |
| **برچسب** | **اندازه پیش فرض صفحات** | **توضیحات** |
| AT45DB321E-SHF-B | 528 Byte | **دارای هشت پایه و SOIC است.** |
| AT45DB321E-SHF-T | 528 Byte | **دارای هشت پایه و SOIC است.** |

توجه داریم در صورتی که اندازه پیش فرض صفحات 528 بایت (512 بایت) باشد، در سری E می­توان 10000 بار صفحات حافظه را از اندازه 528 به 512 و بر عکس تغییر داد[[19]](#footnote-19).

### چگونگی خواندن داده از فلش­های فوق

شماتیک قطعه و بلوک دیاگرام واسط خواندن داده از فلش در شکل زیر نمایش داده شده است:



(الف) نمای پایه­های قطعه (ب) بلوک دیاگرام بخش خواندن از حافظه

شکل ‏2‑10: فلش AT45DB321E

همانگونه که مشاهده می­شود، فلش فوق دارای هشت پایه می­باشد که شرح آنها در جدول زیر خلاصه شده است:

جدول ‏2‑2: پایه­های AT45DB321D و کارکردهای آن

|  |  |
| --- | --- |
| **پایه** | **عملکرد** |
| SCK | پالس ساعت را برای قطعه فراهم می­کند |
| CS | برای انتخاب قطعه (Chip Select) استفاده می­شود. در صورت که سطح آن صفر باشد، قطعه انتخاب شده و عملیات بر روی قطعه قابل انجام است. با یک شدن این پایه، قطعه از حالت آماده به کار خارج می­شود. |
| SI | Slave Input است و داده­ای که قرار است بر روی حافظه نوشته شود و سایر دستورهای ورودی به فلش از طریق این پایه وارد قطعه می­شوند. |
| SO | Slave Output است و داده­­های خروجی از فلش بر روی این پایه قرار می­گیرند. |

همانطور که در شکل ‏2‑16-ب نیز مشاهده می­شود، به صورت کلی دو روش برای خواندن داده از این فلش وجود دارد. یا می­توان داده را به صورت مستقیم از حافظه خواند، یا آنها را در داخل یکی از بافرهای داخلی قطعه قرار داده و سپس از آنها استفاده نمود.

برای خواندن داده از فلش، باید در ابتدا opcode مناسب از طریق پایه SI قطعه به آن ارسال شود. چهار روش خواندن مستقیم داده از فلش در جدول ‏2‑3 خلاصه شده­اند.

جدول ‏2‑3: کدهای عملیاتی (opcode) مختلف فلش AT45DB321E برای خواندن داده

|  |  |
| --- | --- |
| **کد هگز عملیات (opcode)** | **چگونگی خواندن** |
| E8H | خواندن داده به صورت آرایه پیوسته تا فرکانس 66 MHz. در صورت رسیدن به انتهای یک صفحه، از انتهای هر صفحه به ابتدای صفحه بعد می­رود (بدون تاخیر).  در صورت اتمام صفحات به ابتدای حافظه بازمی­گردد. از بافرهای قطعه استفاده نمی­کند. |
| 0BH | خواندن داده با استفاده از واسط سریال تا فرکانس 66 MHz. در صورت رسیدن به انتهای یک صفحه، از انتهای هر صفحه به ابتدای صفحه بعد می­رود (بدون تاخیر).  در صورت اتمام صفحات به ابتدای حافظه بازمی­گردد. از بافرهای قطعه استفاده نمی­کند. |
| 03H | خواندن داده بدون استفاده از بایت­های بی­اهمیت (Don’t care) تا فرکانس 33 MHz. در صورت رسیدن به انتهای یک صفحه، از انتهای هر صفحه به ابتدای صفحه بعد می­رود (بدون تاخیر).  در صورت اتمام صفحات به ابتدای حافظه بازمی­گردد. از بافرهای قطعه استفاده نمی­کند. |
| D2H | خواندن داده از یک صفحه دلخواه تا فرکانس 66 MHz. در صورت رسیدن به انتهای یک صفحه، به ابتدای همان صفحه باز می­گردد (بدون تاخیر). از بافرهای قطعه استفاده نمی­کند. |

در نسخه فعلی از دستور D2H برای خواندن داده از فلش استفاده می­شود، زیرا قرار است در تیک زمانی   
نرم­افزار تنها یک صفحه از داده از فلش خوانده شود (به فصل چهارم رجوع شود). نحوه خواندن داده با این دستور به این صورت است که ابتدا توسط 8 پالس ساعت، این دستور عملیاتی برروی پایه SI ارسال می­شود. سپس آدرس صفحه مورد نظر و همینطور بایتی که قرار است خواندن حافظه از آن آغاز شود و در نهایت چهار بایت بی­اهمیت بر روی این خط ارسال می­شود. از این نقطه به بعد با ارسال هر بایت (که محتوای آن اهمیتی ندارد) یک بایت از حافظه قرائت می­شود.

حال فرض کنیم حافظه دارای صفحات 528 بایتی است. شکل زیر نحوه چینش 13 بیت آدرس صفحه و 10 بیت آدرس بایت را نمایش می­دهد:



شکل ‏2‑11: چگونگی آدرس­دهی حافظه در فلش با فرض صفحات 528 بایتی

همانگونه که مشاهده می­شود، به جهت اینکه مجموع 10 و 13 از 24 بیت (3 بایت) کمتر می­شود و از آنجا که در هر مرحله ارسال هشت بیت داده توسط فلش خوانده می­شود، لازم است یک بیت صفر اضافی به رشته فوق اضافه شود تا اطمینان حاصل شود در هر هشت پالس ساعت تولیدی توسط میکرو، یک بایت اطلاعات وارد فلش می­شود. اضافه کردن بیت صفر به ابتدای رشته باعث می­شود ارزش عددی رشته تغییر نکند که این به نوبه خود باعث حل مشکل آدرس­دهی می­شود. توجه داریم به هنگام ارسال و دریافت هر رشته­ای به/از حافظه (مثل رشته آدرس­دهی که در بالا تعریف شد)، ابتدا باید با اهمیت­ترین بیت رشته (MSB) به سمت فلش ارسال شود (البته می­توان با تنظیم رجیستر­های مناسب از کم­اهمیت­ترین بیت نیز ارسال و دریافت را آغاز نمود که از بررسی آن صرف نظر می­شود). بنابراین ارسال رشته فوق از بیت اضافی صفر آغاز و به هشت بیت پایانی ختم می­شود. توجه داریم در مد خواندن از قطعه، در تمامی لحظات فرستادن رشته آدرس، بیت­های بی­اهمیت پس از آن و همینطور هنگام خواندن داده از فلش، پایه CS باید منطق صفر داشته باشد. با اتمام خواندن تمامی صفحات مورد نظر نیز لازم است CS مجدداً یک شود تا بتوان خواندن جدیدی را از حافظه آغاز نمود.

حال فرض کنیم حافظه دارای صفحات با اندازه 512 بایت باشد. در این صورت رشته آدرس به صورت زیر تغییر پیدا می­کند:



شکل ‏2‑12: چگونگی آدرس­دهی حافظه در فلش با فرض صفحات 512 بایتی

همانگونه که مشاهده می­شود مجموع بیت­های محتوی آدرس در رشته فوق برابر 22 = 13 + 9 است و برای تبدیل آن به مضربی از 24 بیت، لازم است دو بیت صفر به ابتدای رشته فوق اضافه شود. البته برای ساده­سازی پیاده­سازی، سازوکاری در نظر گرفته شده است که اندازه صفحات فلش حتماً برابر 528 بایتی تنظیم شوند که در این مورد در بخش بعدی توضیح داده خواهد شد.

### مشکل ایجاد شده در بوردهای FSB200a به دلیل رعایت نکردن قالب آدرس­دهی – تنظیم اندازه صفحات فلش به 528 بایت

عدم توجه به اندازه صفحات فلش سبب ایجاد مشکل در تولید بوردهای FSB200a شده است. در واقع در ICهای استفاده شده برای بوردها به صورت تصادفی تعدادی فلش با اندازه صفحه 528 و تعدادی نیز با اندازه 512 بایتی وجود داشته است، در حالی که برنامه بورد FSB200a تنها برای صفحات با اندازه 528 بایتی نوشته شده است. به این ترتیب و با توجه به اینکه قالب آدرس­دهی برای دو صفحه فوق متفاوت است، بوردها در هنگام پخش صوت دارای مشکل شده­اند و صوت به صورت نامفهوم پخش می­شود. تعیین اندازه صفحات حافظه فلش با خواندن رجیستر وضعیت حافظه فلش ممکن می­باشد. همچنین برای تغییر اندازه صفحات به 528 بایت، دو قدم مجزا لازم است (بلوک دیاگرام شکل ‏2‑13):

1- پاک کردن تمامی داده­های موجود در حافظه AT45DB321E.

2- تنظیم صفحات حافظه به 528 با تنظیم رجیستری فلش

برای انجام عملیات اول می­توان از روش Chip Erase اشاره شده در دیتاشیت AT45DB321E (بخش 10-6) استفاده کرد. در این روش ابتدا دستور عملیاتی 0xC7، و سپس دستورهای 0x94، 0x80، 0x9A به IC ارسال   
می­شوند. پس از خاتمه ارسال این دستورها عملیات پاک­سازی حافظه آغاز می­شود که پایان آن با تغییر وضعیت بیت RDY/BUSY در رجیستر Status Register حافظه مشخص می­شود. عملیات فوق در تابع Chip\_Erase() در برنامه میکروکنترلر پیاده­سازی شده است (به فصل چهارم رجوع شود).

برای انجام عملیات دوم همانگونه که در بخش 10 دیتاشیت اشاره شده است، لازم است چهار دستور عملیاتی 0x3D، 0x2A، 0x80 و 0xA7 به IC ارسال شوند و مشابه بخش قبل تغییر وضعیت بیت RDY/BUSY در رجیستر وضعیت بررسی شود. پس از تغییر وضعیت این بیت، عملیات خاتمه یافته است. این عملیات نیز در تابع Set\_Flash\_Register() میکرو پیاده­سازی شده است (به فصل چهارم رجوع شود).

پس از انجام دو عملیات فوق که به صورت خودکار پس از روشن شدن بورد FSB انجام می­شوند، اندازه صفحات به 528 بایتی تنظیم می­شوند؛ اما توجه داریم از آنجا که در قدم اول تمامی محتویات حافظه پاک می­شود و از آنجا که در هنگام پروگرام بورد ابتدا حافظه فلش پروگرام می­شود، لازم است یکبار دیگر حافظه پروگرام شود تا بورد کارکرد عادی خود را داشته باشد.



شکل ‏2‑13: بلوک دیاگرام تغییر اندازه صفحات حافظه فلش به 528 بایت

### ارتباط SPI و چگونگی خواندن داده از AT45DB321E

ارتباط SPI یک نوع ارتباط Master (به صورت مخفف M) و Slave (به صورت مخفف S) است. این به معنای آن است که فعال­سازی ارتباط (Enable) و ارسال پالس ساعت توسط M صورت می­پذیرد. شکل زیر، نمای کلی این ارتباط را نمایش می­دهد:

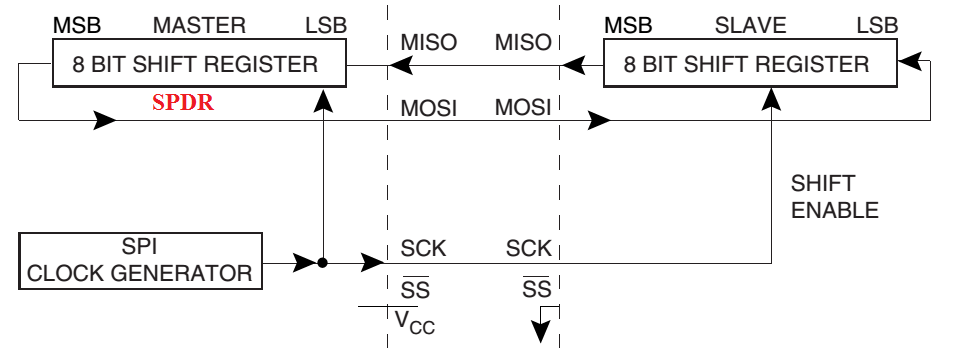


شکل ‏2‑14: نمای ارتباط SPI

همانطور که مشاهده می­شود، ارتباط فوق دارای چهار خط است:

* SCLK: پالس ساعت توسط M بر روی این خط برای S فراهم می­شود.
* MOSI: پورت خروجی M و ورودی S است
* MISO: پورت ورودی M و خروجی S است
* SS: پورت فعال­سازی ارتباط است. با صفر کردن این خط، S انتخاب می­شود.

پس از فعال شدن ارتباط SPI، S و M دیتایی را که قرار است با یکدیگر مبادله کنند در رجیسترهای خود قرار   
می­دهند. این رجستر در سمت M (که در اینجا ATmega8 است)، SPDR نام دارد. ارتباط دو رجیستر در شکل زیر نمایش داده شده است:



شکل ‏2‑15: ارتباط بین رجیسترهای M و S در ارتباط SPI

همانگونه که اشاره شد داده به صورت پیش فرض از MSB میان دو شیفت رجیستر جابجا می­شود. توجه داریم ارتباط SPI یک ارتباط دوطرفه کامل (Full-Duplex) است. این به معنای آن است که حرکت داده دو طرفه است و اصطلاحاً یک حلقه چرخش داده میان دو سمت برقرار می­شود. به صورت دقیق­تر، نحوه ارسال و دریافت داده در این ارتباط به این صورت است که ابتدا M، داده­ای که قرار است برای S بفرستد را بر روی رجیستر SPDR قرار می­دهد. سپس با شروع از MSB، این داده­ها به سمت S و برروی خط MOSI حرکت داده می­شوند و در همان زمان داده دریافتی از رجیستر S بر روی خط MISO ارسال شده و در رجیستر SPDR قرار می­گیرد. پس از سپری شدن هشت پالس ساعت، گذار داده انجام شده است و با خواندن داده از رجیستر SPDR می­توان از داده دریافتی از فلش استفاده نمود. توجه داریم با خاتمه ارسال و دریافت داده، بیت MSB رجیستر SPSR برابر یک می­شود که این نشانه خاتمه چرخه فوق است.

رجیستر SPCR رجیستر مهم دیگر مرتبط با این ارتباط می­باشد که با استفاده از آن می­توان فعال یا غیر فعال بودن وقفه مرتبط با این ارتباط، فعال کردن خود ارتباط، ترتیب فرستادن داده (از MSB به LSB و یا برعکس)، انتخاب Master و یا Slave، و پلاریته پالس ساعت و همینطوری لبه­ای که با آن عملیات SPI فعال می­شود را معین نمود. ضمن اینکه ارتباط پالس ساعت با فرکانس نوسان­ساز در این ارتباط توسط سه بیت تعیین می­شود که عبارتند از دو بیت LSB این رجیستر و بیت LSB رجیستر SPSR. نحوه تنظیم این رجیسترها به صورت مفصل در دیتاشیت میکروکنترلر آمده است.

### ارتباط میان فرکانس نمونه­برداری صوت، تیک زمانی برنامه و سرعت ارتباط SPI جهت خواندن داده از حافظه فلش

یکی از مهمترین مسائلی که باید در هنگام پیاده­سازی نرم­افزار بورد FSB رعایت شود، توجه به زمان­بندی خواندن داده از فلش با استفاده از ارتباط SPI و پخش آنها توسط وقفه میکروکنترلر است و همچنین ارتباط هر دوی این موارد با تیک زمانی برنامه است.

به جهت اینکه از پیچیده شدن نرم­افزار بورد جلوگیری شود، می­توان تیک زمانی برنامه را معادل زمانی تعریف کرد که یک صفحه داده فلش توسط وقفه تایمر/کانتر پخش می­شود. این امر باعث کاهش تعداد تایمر کانترهای مورد استفاده در بورد نیز میشود. توجه داریم از آنجا که زیرنمونه­برداری با نرخ 2 صورت پذیرفته است، هر بایت یک صفحه فلش معادل دو نمونه پشت سر هم از موسیقی است. بنابراین اگر فرکانس نمونه­برداری برابر 32kHz در نظر گرفته شود، هر صفحه داده معادل و به عبارتی 32ms موسیقی است. اگر تیک زمانی برنامه برابر 32ms در نظر گرفته شود، آنگاه باید خواندن این صفحه از موسیقی در زمانی بسیار کمتر از 32ms صورت بپذیرد تا داده­ها در میکرو در اختیار باشند تا بتوان آنها را توسط وقفه اجرا نمود. به صورت تجربی مشاهده شد که حتی اگر فرکانس پالس ساعت ارتباط بیشترین مقدار خود (یعنی نصف فرکانس پالس ساعت میکرو) را داشته باشد، باز هم عملیات خواندن داده­های یک صفحه و پخش آن در طول یک تیک زمانی پایان نمی­پذیرد. در نتیجه کیفیت صدای خروجی به دلیل از دست رفتن تعدادی از بیت­ها کاهش می­یابد.

راه­حلی جایگزینی که به صورت تجربی در هنگام پیاده­سازی به دست آمد، خواندن نصف یک صفحه از حافظه و تعریف تیک زمانی برنامه برابر زمانی است که این نصف صفحه توسط میکرو پخش می­شود. مشابه استدلالی که در بالا شد، برای پخش یک نصف صفحه به نیاز است و بنابراین تیک زمانی برنامه برابر 16.5ms خواهد شد. با این تعریف، در هر تیک زمانی برنامه تنها تا بایت 264 یک صفحه داده خوانده می­شود که از لحاظ زمان­بندی میان خواندن داده و پخش آن در یک تیک مشکلی پیش نخواهد آمد.

شکل ‏2‑16 نشان دهنده زمان­بندی میان خواندن داده از فلش توسط ارتباط SPI، پخش آن توسط وقفه و همینطور تیک زمانی برنامه است. همانگونه که در شکل مشاهده می­شود زمان وقفه تایمر/کانتر یک به عنوان معیار همزمان­سازی زمان­بندی قسمت­های مختلف استفاده می­شود. همانگونه که اشاره شد برای خواندن هر نیمه از صفحات حافظه 528 وقفه لازم می­باشد. برای اینکه داده­های تازه همیشه در حین اجرای این وقفه­ها در دسترس باشند لازم است خواندن داده از فلش قبل از این 528 وقفه آغاز شود. به همین دلیل تیک زمانی برنامه برابر 500 وقفه تنظیم می­شود تا بیست و هشت بایت قبل از پایان یافتن داده­ها، خواندن داده­های جدید آغاز شود. توجه داریم در صورتی که تیک زمانی برنامه برابر 528 وقفه تعریف شود، آنگاه چند شکل موج اول PWM از صفحه قبلی حافظه خواهند بود که این امر باعث ایجاد نویز در هنگام پخش صوت خواهد شد.



شکل ‏2‑16: زمان­بندی تیک زمانی برنامه، خواندن از فلش (ارتباط SPI) و همچنین وقفه­های تایمر/کانتر یک

با توجه به شکل و توضیحاتی که در بالا اشاره شد، می­توان تنها با استفاده از تایمر/کانتر یک تیک زمانی­ برنامه را تنظیم نمود و در عین حال موج PWM را نیز تولید کرد. به همین دلیل در نرم­افزار تنها از یک تایمر/کانتر استفاده می­شود، که این امر یک بهبود نسبت به بورد نسخه FSB200a است که از دو تایمر/کانتر برای عمل فوق استفاده شده است.

### استفاده از دو حافظه فلش در بورد FSB

همانگونه که در شکل ‏2‑14 نیز مشاهده می­شود، با استفاده از ارتباط SPI تنها می­توان یک Slave را مورد خطاب قرار داد، زیرا سیگنال CS تنها می­تواند یک Master را فعال کند. توجه داریم استفاده از یک حافظه فلش عموماً برای جای دادن موسیقی­ها و پیام­ها کافی است، اما در بعضی موارد و به خصوص اگر اندازه موسیقی مورد استفاده در بورد بزرگ باشد و همینطور اگر تعداد طبقات در بورد FSB310 زیاد باشد، استفاده از دو حافظه فلش لزوم پیدا می­کند.

استفاده از دو چیپ AT45DB321E از آنجایی پیچیده است که ارتباط SPI تنها قادر به فعال­سازی یک چیپ با استفاده از سیگنال CS است. برای حل مشکل ارسال یک سیگنال CS به چند چیپ، می­توان از چیپ­های demultiplexer استفاده کرد. این چیپ­ها قادرند یک سیگنال ورودی را به یکی از پایه­های خروجی خود (که توسط پایه­های انتخاب چیپ آن تعیین می­شود) ارسال کنند. اما از آنجا که هدف در بورد FSB، سیگنال­دهی به تنها دو چیپ است، می­توان از پیکربندی­های ساده­تری نیز استفاده کرد که بحث کامل­تر در این مورد در فصل پنجم این گزارش (بخش ‏5-2-6-) آمده است. با پیاده­سازی این سخت افزار و اعمال تغییرات نرم­افزاری مناسب   
(بخش ‏4-4-2-5-)، می­توان به کمک پایه PB0 میکرو یک چیپ demultiplexer پیاده­سازی کرد و از طریق آن CS را به چیپ دلخواه فرستاد. به این ترتیب خواندن داده نیز از یکی از دو چیپ AT45 ممکن خواهد شد.

## تعریف سیگنال­های ورودی در بورد FSB

در فصل اول این گزارش به صورت مختصر در مورد سیگنال­های ورودی به بورد بحث شد. در این قسمت به صورت جزئی­تر در مورد سیگنال­های ورودی، سطح منطقی آنها و همینطور پورت­های خواندن سیگنال در میکرو پرداخته می­شود.

به صورت کلی دو دسته سیگنال به عنوان ورودی به پورت­های میکروکنترلر وارد می­شوند که عبارتند از سیگنال­های کنترلی و سیگنال­های شمارنده. در ادامه در مورد هر دسته از این سیگنال­ها در ادامه بحث خواهد شد.

### سیگنال­های کنترلی و چگونگی دریافت آنها در بورد FSB300

سیگنال­های کنترلی[[20]](#footnote-20) سیگنال­هایی هستند که نشان دهنده وضعیت کلی بورد FSB می­باشند. در بورد FSB300 این سیگنال­ها از پورت C میکروکنترلر و به صورت موازی خوانده می­شوند. به همین دلیل پورت C میکروکنترلر در بورد FSB300 اصطلاحاً پورت کنترلی نام دارد. شش سیگنال کنترلی در بورد FSB تعریف می­شوند:

* **سیگنال اورژانسی:** این سیگنال نشان دهنده وضعیت EMR در آسانسور می­باشد و با نماد EMR\_SIGNAL در کد نشان داده شده است. این سیگنال یک سیگنال DC و 24 ولتی است و در بورد FSB300 از پورت PC2 میکرو دریافت می­شود.
* **سیگنال فعال­سازی یا trigger:** این سیگنال نشان دهنده وضعیت TRG یا شروع حرکت آسانسور می­باشد و با نماد TRG\_SIGNAL در کد نشان داده می­شود. اگر آسانسور شروع به حرکت کند، سطح منطقی سیگنال تغییر می­کند (معمولاً از یک به صفر می­رود) و در هنگام دورگیری قبل از توقف و در حالت سکون آسانسور نیز منطق آن تغییر خواهد کرد (معمولاً از صفر به یک می­رود). با تعریف فوق، سیگنال TRG منطق Active High یا به صورت خلاصه AH خواهد داشت. در صورتی که سطح منطقی سیگنال در هنگام حرکت از صفر به یک برود، سیگنال Active Low یا AL خواهد شد. در بورد FSB300 این سیگنال از پورت PC5 میکرو دریافت می­شود.
* **سیگنال باز بودن درب:** این سیگنال نشان دهنده وضعیت DOB در آسانسور می­باشد و با نماد DOB\_SIGNAL در کد نشان داده شده است. سیگنال DOB به صورت مستقیم از سنسور باز بودن درب دریافت می­شود و به همین دلیل عموماً هنگام باز بودن درب به دلیل وجود یک مانع، یک سیگنال به سطح منطقی صفر تولید می­کند. بنابراین سیگنال DOB عموماً یک سیگنال AL است و با همین منطق در بورد پیاده­سازی شده است، اگر چه با اعمال جامپرهای مناسب می­توان آن را AH نیز نمود (به فصل پنجم رجوع شود). دو وضعیت دیگر در ارتباط با سیگنال DOB، Normally Open (NO) و Normally Closed (NC) بودن آن است که نشان دهنده دریافت سیگنال در حالت باز بودن درب و یا بسته بودن آن می­باشد. در بورد FSB300 این سیگنال از پورت PC3 بورد وارد می­شود.
* **سیگنال اضافه وزن:** این سیگنال نشان دهنده وضعیت OVL می­باشد و با نماد OVL\_SIGNAL در کد نشان داده شده است. در صورت تشخیص وجود اضافه وزن با استفاده از سنسور اضافه وزن، این سیگنال منطق صفر پیدا خواهد کرد. در بورد FSB300 این سیگنال از پورت PC4 دریافت می­شود.
* **سیگنال­های تغییر سطح صدا:** این سیگنال­ها از دو کلید تعبیه شده بر روی بورد دریافت می­شوند و برای تغییر سطح صدای بورد استفاده می­شوند. سیگنال­ها شامل VOL\_UP (برای افزایش صدا) و VOL\_DOWN (برای کاهش صدا) هستند. در حالت عادی منطق قرار گرفته بر روی این دو پورت یک است، اما با فشرده شدن هر یک از کلید­ها این منطق صفر خواهد شد. این دو سیگنال (در FSB300 و FSB310) به ترتیب از طریق پورت­های PC0 و PC1 دریافت می­شوند. برای اینکه تغییر سطح صدای بورد در هر زمانی امکان پذیر باشد، امکان تولید این سیگنال­ها و خواندن آنها در هر لحظه­ای از زمان در بورد وجود دارد، به همین دلیل این دو سیگنال کنترلی در کنار سیگنال تغییر آلبوم به صورت مستقل از سایر سیگنال­های کنترلی بررسی و دیبانس می­شوند (به بخش ‏2-8- رجوع شود. دیبانس این سیگنال­ها در تابع VOL\_Signal\_Detection() صورت می­پذیرد که در بخش ‏4-3-3-5- تعریف شده است).
* **سیگنال تغییر آلبوم:** این سیگنال با فشرده­ شدن همزمان دو کلید فشاری بر روی بورد تولید   
  می­شود و از آن برای تغییر آلبوم موسیقی در حال پخش در بورد استفاده می­شود. به همین دلیل سطح منطقی سیگنال، پورت دریافت آن و همینطور منطق دیبانس آن در بورد FSB300 با سیگنال تغییر سطح صدا مشترک است.

با توجه به تعاریف فوق، از شش سیگنال کنترلی چهار سیگنال از بورد کنترل کننده آسانسور به بورد داده می­شوند (یعنی EMR، OVL، DOB و TRG) و دو سیگنال نیز از طریق کلیدهای فشاری تعبیه شده بر روی بورد به آن وارد می­شوند (یعنی تغییر سطح صدا و تغییر آلبوم).

#### اولویت­ سیگنال­های کنترلی

همانگونه که در فصل اول اشاره شد، منظور از اولویت سیگنال­های کنترلی اهمیت پخش پیام متناظر با آن نسبت به پیام متناظر با سایر سیگنال­ها است. با توجه به تعریف فوق، واضح است که سیگنال اورژانسی بالاترین اولویت را دارد، زیرا در صورت وجود این سیگنال باید بلافاصله پیام نجات اضطراری بدون توجه به سایر سیگنال­های ورودی پخش شود. به همین دلیل در بورد FSB300 و در زمان بررسی وجود سیگنال اورژانسی، هنگام خواندن سیگنال پورت C از رجیستر PORTC مقدار این رجیستر همیشه ضرب­ در 0x04 می­شود[[21]](#footnote-21) تا سیگنال متناظر با سایر پین­های ورودی در نظر گرفته نشوند. پس از سیگنال اورژانسی، با اولویت­ترین سیگنال­، سیگنال­های کلیدهای فشاری هستند چون می­توان آلبوم­ را در اکثر وضعیت­ها تغییر داد (به فصل چهارم رجوع شود) و تغییر سطح صدا نیز در تمامی لحظات ممکن می­باشد. به همین دلیل و همانگونه که در بالا اشاره شد، اصولاً دیبانس و آشکارسازی این سیگنال­ها در بوردهای FSB به صورت مستقل از سایر سیگنال­ها صورت می­پذیرد.

پس از سیگنال­های اورژانسی و تغییر سطح صدا/ آلبوم، بالاترین اولویت متعلق به سیگنال TRG می­باشد. علت اینکه این سیگنال اولویت بالاتری نسبت به مثلاً سیگنال OVL دارد (چون اصولاً باید پخش پیام OVL درجه اهمیت بالاتری داشته باشد تا از مسافران خواسته شود شرایط حرکت آسانسور را فراهم کنند) این مطلب است که بورد اصلی کنترل کننده آسانسور اجازه تولید سیگنال TRG را قبل برطرف شدن وضعیت­های اضافه وزن و بازبودن درب در آسانسور نمی­دهد. از طرف دیگر یکی از ضعف­های بورد FSB200a، پخش پیام DOB در حین حرکت آسانسور و در زمان پخش موسیقی است[[22]](#footnote-22) که مورد قبول نمی­باشد. بالاتر بودن اولویت سیگنال TRG نسبت به OVL یا DOB باعث می­شود تا در صورت وجود همزمان سیگنال TRG و OVL یا DOB، تنها موسیقی پخش شود و از پخش پیام­های هشدار جلوگیری به عمل آید. در آخرین سطح اولویت، سیگنال­­های DOB و OVL قرار دارند که سیگنال OVL اولویت بالاتری نسبت به DOB دارد، زیرا برطرف شدن مشکل اضافه وزن باید قبل از بسته شدن درب آسانسور صورت بپذیرد.

توجه داریم اولویت­های تعریف شده برای سیگنال­های اورژانسی در هر دو بورد FSB300 و FSB310 یکسان است.

#### سطح منطقی سیگنال­های کنترلی در بورد FSB300

جدول ‏2‑4 سطوح منطقی هر پورت را متناظر با هر سیگنال­ کنترلی با توجه به اولویت­ها نشان می­دهد. فرض شده است سیگنال­های TRG و DOB دارای منطق AL هستند و سیگنال DOB به صورت NC است. در این جدول که برای FSB300 نوشته شده است، هشت بیتی تولید شده در رجیستر PORTC پس از حذف سیگنال متناظر با پورتهایی که برای تشخیص یک سیگنال اهمیت ندارند (به عنوان مثال تمامی پورت­ها به غیر از PC2 برای سیگنال اورژانسی، یا تمامی پورت­ها به غیر از PC2 و PC5 برای سیگنال TRG) نشان داده شده است. همچنین جدول ‏2‑5 سطوح سیگنال را برای حالتی نشان می­دهد که سیگنال TRG به صورت AH باشد.

جدول ‏2‑4- سطح منطقی پورت­ها برای ساخت یک سیگنال کنترلی دلخواه با فرض ورودی­های AL برای DOB و TRG در بورد FSB300

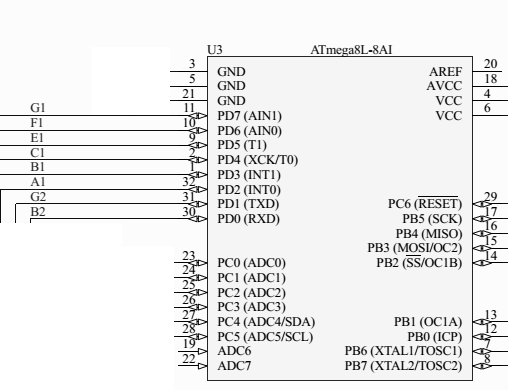
|  | | سیگنال­های پورت کنترلی به ترتیب از کم ارزش­ترین بیت (PC0) به با ارزش­ترین بیت (PC5) | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| سیگنال | سیگنال پورت کنترلی متناظر پس از حذف بیت­های بدون اهمیت | V+ | V- | EMR | DOB | OVL | TRG |
| (PC0) | (PC1) | (PC2) | (PC3) | (PC4) | (PC5) |
| سیگنال EMR | 0x04 |  |  | 1 | x | x | x |
| سیگنال TRG | 0x00 |  |  | 0 | x | x | 0 |
| سیگنال OVL | 0x20 |  |  | 0 | x | 0 | 1 |
| سیگنال DOB | 0x30 |  |  | 0 | 0 | 1 | 1 |
| سیگنال ALBUM\_CHANGE | 0x00 | 0 | 0 |  |  |  |  |
| سیگنال VOL\_UP | 0x02 | 0 | 1 |  |  |  |  |
| سیگنال VOL\_DOWN | 0x01 | 1 | 0 |  |  |  |  |

جدول ‏2‑5- سطح منطقی پورت­ها برای ساخت یک سیگنال کنترلی دلخواه با فرض ورودی­ AL برای DOB و AH برای TRG در بورد FSB300

|  | | سیگنال­های پورت کنترلی به ترتیب از کم ارزش­ترین بیت (PC0) به با ارزش­ترین بیت (PC5) | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| سیگنال | سیگنال پورت کنترلی متناظر پس از حذف بیت­های بدون اهمیت | V+ | V- | EMR | DOB | OVL | TRG |
| (PC0) | (PC1) | (PC2) | (PC3) | (PC4) | (PC5) |
| سیگنال EMR | 0x04 |  |  | 1 | x | X | x |
| سیگنال TRG | 0x20 |  |  | 0 | x | X | 1 |
| سیگنال OVL | 0x00 |  |  | 0 | x | 0 | 0 |
| سیگنال DOB | 0x10 |  |  | 0 | 0 | 1 | 0 |
| سیگنال ALBUM\_CHANGE | 0x00 | 0 | 0 |  |  |  |  |
| سیگنال VOL\_UP | 0x02 | 0 | 1 |  |  |  |  |
| سیگنال VOL\_DOWN | 0x01 | 1 | 0 |  |  |  |  |

### سیگنال­های شمارنده در بورد FSB300

منظور از سیگنال­های شمارنده[[23]](#footnote-23) در حالت کلی سیگنال­هایی است که شماره توقف و همینطور وضعیت آن را نشان می­دهند. در بورد FSB300 اصطلاحاً پورت D پورت شمارنده بورد FSB نام دارد، زیرا تمامی سیگنال­های شمارنده از آن خوانده می­شوند. این سیگنال­ها با نام­های A1، B1، C1، E1، F1، G1، B2 و G2 و به صورت AL در بورد تعریف می­شوند. همانگونه که در فصل اول اشاره شد، بورد FSB تنها وضعیت توقف را دریافت می­کند و قابلیت دریافت و پردازش شماره توقف را ندارد[[24]](#footnote-24). برای درک بهتر ارتباط میان سیگنال­های شمارنده وضعیت توقف و وضعیت توقف اعلان شده، شکل ‏2‑17 را در نظر می­گیریم. همانگونه که مشاهده می­شود هر سیگنال شمارنده متناظر با یک پایه 7Seg می­باشد. ضمن اینکه 7Seg سمت راست نشان دهنده یکان شماره طبقه (و یا لابی، همکف و ...) می­باشد و 7Seg سمت راست نشان دهنده منفی بودن طبقات و یا بیش از ده بودن آنهاست. از طرف دیگر (و به صورت قراردادی) اگر سیگنال G2 منطق یک داشته باشد آنگاه در طبقات منفی هستیم و اگر سیگنال B2 دارای منطق یک باشد آنگاه در طبقات بیش از نه و قبل از بیست قرار داریم[[25]](#footnote-25). توجه داریم سیگنال D1 به این دلیل در نظر گرفته نشده است که دقیقاً هشت سیگنال شمارنده داشته باشیم (که بتوان آنها را توسط تنها یک پورت میکرو قرائت کرد)[[26]](#footnote-26). به این ترتیب به عنوان مثال اگر داشته باشیم A1 = B1 = C1 = B2 = 1، آنگاه شماره طبقه 17 خواهد بود (شکل ‏2‑17-ج). همچنین لابی با F1 = E1 = 1 نمایش داده می­شود. توجه داریم با توصیفات فوق در شرایط فعلی تنها می­توانیم از پارکینگ منهای نه تا طبقه نوزده را نمایش دهیم. لازم به یادآوری است طبقه همکف (Ground) به صورت G ناکامل و صفر (شکل ‏2‑17-ه و شکل ‏2‑17-و) سیگنال­دهی می­شود و سطح منطقی این دو سیگنال در کد با ثوابت GROUND و GROUND0 در نظر گرفته شده است.

(الف) چگونگی ارتباط میان سیگنال پورت­های شمارنده (ب) نمایش سیگنال­های شمارنده بر روی 7Seg

و پورت D میکرو در FSB300

(ج) نمایش طبقه هفدهم توسط 7Seg به همراه سیگنال­های متناظر (د) نمایش لابی توسط 7Seg به همراه سیگنال­های متناظر

(ه) سیگنال­دهی طبقه همکف با G ناکامل (و) سیگنال­دهی طبقه همکف به صورت صفر

شکل ‏2‑17: ارتباط میان سیگنال­های شمارنده وضعیت توقف و پورت D (پورت شمارنده) میکروکنترلر در بورد FSB300 و چند نمونه   
از سیگنال وضعیت توقف­ (خطوط سبز نشان دهنده قسمت­های روشن 7seg می­باشند)

#### تخصیص شماره توقف در بورد FSB300

همانگونه که اشاره شد، در بورد FSB300، امکان ارسال شماره توقف به بورد فراهم نشده است و بنابراین مجبوریم به هر وضعیت توقف، یک شماره توقف نسبت بدهیم. جدول ‏2‑6 ارتباط میان وضعیت توقف، سیگنال شمارنده دریافتی برای آن وضعیت و همینطور شماره توقف نسبت داده شده به آن را نمایش می­دهد.

جدول ‏2‑6- وضعیت توقف، سیگنال شمارنده متناظر و شماره توقف (با فرض منطق AL برای ورودی­ها) در بورد FSB300

|  | | | سیگنال­های شمارنده  به ترتیب از کم ارزش­ترین بیت (B2 یا PD0) به با ارزش­ترین بیت (G1 یا PD7) | | | | | | | | شماره توقف نسبت داده شده |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| وضعیت توقف | | سیگنال پورت شمارنده | B2  (PD0) | G2  (PD1) | A1  (PD2) | B1  (PD3) | C1  (PD4) | E1  (PD5) | F1  (PD6) | G1  (PD7) |
| پارکینگ نهم | | 0x21 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| پارکینگ هشتم | | 0x01 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| پارکینگ هفتم | | 0xE1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 |
| پارکینگ ششم | | 0x09 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 4 |
| پارکینگ پنجم | | 0x29 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 5 |
| پارکینگ چهارم | | 0x25 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 6 |
| پارکینگ سوم | | 0x61 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 7 |
| پارکینگ دوم | | 0x51 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 8 |
| پارکینگ اول | | 0xE5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 9 |
| پارکینگ | | 0x13 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 10 |
| زیرزمین (Basement) | | 0x0F | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 11 |
| همکف | G ناکامل (Ground) | 0x8B | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 12 |
| G ناکامل (Ground0) | 0x83 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| لابی (Lobby) | | 0x9F | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| طبقه اول | | 0xE7 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 13 |
| طبقه دوم | | 0x53 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 14 |
| طبقه سوم | | 0x63 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 15 |
| طبقه چهارم | | 0x27 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 16 |
| طبقه پنجم | | 0x2B | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 17 |
| طبقه ششم | | 0x0B | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 18 |
| طبقه هفتم | | 0xE3 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 19 |
| طبقه هشتم | | 0x03 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 20 |
| طبقه نهم | | 0x23 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 21 |
| طبقه دهم | | 0x82 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 22 |
| طبقه یازدهم | | 0x86 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 23 |
| طبقه دوازدهم | | 0x52 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 24 |
| طبقه سیزدهم | | 0x62 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 25 |
| طبقه چهاردهم | | 0x26 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 26 |
| طبقه پانزدهم | | 0x2A | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 27 |
| طبقه شانزدهم | | 0x0A | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 28 |
| طبقه هفدهم | | 0xE2 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 29 |
| طبقه هجدهم | | 0x02 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| طبقه نوزدهم | | 0x22 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 31 |

## مفهوم دیبانس سیگنال­های ورودی

اگر فرض کنیم محیط کار بورد عاری از نویز باشد، می­توان سطح سیگنال یک پین ورودی را در هر لحظه دلخواه بررسی کرد و در مورد سطح منطقی آن با قرائت رجیسترهای مربوطه تصمیم­گیری نمود. اما از آنجا که بورد به دلیل همجواری با سایر بوردها و عناصر الکتریکی دچار نویز می­شود، ممکن است سطح سیگنال­های ورودی­ به صورت ناخواسته تغییر کنند که این امر روند خواندن سیگنال­های ورودی را دچار مشکل می­کند.

برای حل مشکل فوق از مفهومی به نام دیبانس[[27]](#footnote-27) سیگنال­های ورودی استفاده می­شود؛ دیبان یک سیگنال به معنای خواندن آن سیگنال در چند بازه زمانی مجزا و تصمیم­گیری در مورد سطح منطقی آن است. بدان معناست که تصمیم­گیری در مورد سطح سیگنال ورودی تنها به یکبار مشاهده سیگنال ورودی محدود نمی­شود، بلکه در صورتی که سطح سیگنال در چندبار مشاهده متوالی آن مقدار معینی داشته باشد، آنگاه آن مقدار به عنوان سطح ورودی سیگنال پذیرفته خواهد شد. به این صورت اگر مقدار یک سیگنال به دلیل حضور ناگهانی نویز تغییر کند، این مقدار به عنوان ورودی در نظر گرفته نخواهد شد. در زیر چگونگی دیبانس سیگنال­های مختلف ارائه خواهد شد.

**توجه:** در مورد سیگنال­های کنترلی و شمارنده در بورد FSB300 به دلیل دریافت این سیگنال­ها از پورت­های ورودی میکروکنترلر عمل دیبانس الزام دارد. اما در بورد FSB310 از آنجا که سیگنال­های کنترلی و شمارنده همانگونه که در ادامه اشاره خواهد شد به صورت سریال و توسط پروتکل I2C دریافت می­شوند، دیبانس سیگنال­ها معنی ندارد و خود پروتکل با تولید ACK و NACK صحت و یا عدم صحت ارسال و دریافت سیگنال­ها را تضمین می­کند (به گزارش «*پروتکل ارتباطی I2C»* رجوع شود). لازم به ذکر است که دیبانس سیگنال­های تغییر سطح صدا در هر دو بورد FSB300 و FSB310 صورت می­پذیرد، زیرا این سیگنال­ها از طریق دو کلید فشاری به بورد وارد می­شوند که ممکن است دچار نویز شوند.

### دیبانس سیگنال­های کنترلی (به غیر از سطح صدا) و شمارنده در بورد FSB300

دیبانس­ سیگنال­های کنترلی و شمارنده از یک منطق تبعیت می­کند و بدین صورت است که در هر مرحله ابتدا تمامی سیگنال­ها از پورت­های متناظر قرائت و ذخیره می­شوند و این خواندن در تیک زمانی بعدی برنامه تکرار می­شود. در صورتی که در چند خواندن با فاصله زمانی برابر تیک زمانی برنامه سیگنال مقدار یکسانی داشته باشد، این سیگنال به عنوان یک سیگنال معتبر در نظر گرفته شده و آشکار می­شود.

### دیبانس سیگنال تنظیم سطح صدا/ تغییر آلبوم

همانگونه که اشاره شد سیگنال­های تنظیم سطح صدا/ تغییر آلبوم با استفاده از دو کلید فشاری تولید   
می­شوند. مسئله­ای که باعث می­شود دیبانس این سیگنال­ها کمی پیچیده­تر شود، تصمیم­گیری در مورد نوع سیگنال ورودی به دلیل نوع استفاده کاربر از این کلیدها است. به عنوان مثال در هنگام تغییر آلبوم (که برای تولید سیگنال آن لازم است هر دو کلید به صورت همزمان پایین نگاه داشته شوند)، ممکن است کاربر یکی از کلیدها را زودتر از دیگری فشار دهد و یا آن را زودتر از کلید دیگر رها کند. در صورت عدم توجه به این مسئله، ممکن است به جای تغییر آلبوم سطح صدا افزایش یا کاهش پیدا کند که مطلوب نمی­باشد.

برای حل مشکل فوق، اولاً در نظر می­گیریم وجود هیچ سیگنالی تا قبل از رها شدن کلیدها اعلام نشود. ضمن اینکه در صورتی که پس از دیبانس سیگنال­ها نتیجه شود که دو کلید به صورت همزمان فشرده شده­اند، دیگر هیچ سیگنال ورودی از دو کلید خوانده نشود و تنها پس از رها شدن دو کلید، آلبوم تغییر کند. شرط اول به کاربر اجازه می­دهد در صورتی که در ابتدا یکی از دو کلید را زودتر از دیگری فشار داده، بتواند با فشردن کلید دوم در ادامه کار سیگنال تغییر آلبوم را تولید کند. شرط دوم نیز اجازه می­دهد که در صورتی که یک کلید زودتر از کلید دیگر رها شود، سیگنال اشتباهاً سیگنال تغییر سطح صدا تلقی نشود. پیاده­سازی دیبانس سیگنال­های سطح صدا با در نظر گرفتن منطق فوق در بخش ‏4-3-3-5- آمده است.

## استفاده از پروتکل I2C برای دریافت سیگنال­های ورودی به صورت سریال در بورد FSB310

پروتکل I2C یک پروتکل ارتباطی دو سیمه است که از آن برای ارسال و دریافت اطلاعات استفاده می­شود. این ارتباط به صورت Master و Slave عمل می­کند که Master پالس ساعت را بر روی خط SCL ارتباط فراهم   
می­کند و سپس از Slave درخواست می­کند تا اطلاعاتی را برای آن ارسال کند/ اطلاعاتی را بر روی آن بنویسید. ارسال و دریافت داده در این پروتکل در قاب­های هشت بیتی صورت می­پذیرد. برای توضیحات بیشتر در مورد این پروتکل به گزارش «*پروتکل ارتباطی I2C»* رجوع شود. پیش از توضیح در مورد چگونگی دریافت سیگنال توسط این ارتباط لازم است در مورد سیگنال کنترلی و شمارنده ارسالی در بورد LCB216 که با بورد FSB310 ارتباط دارد توضیح داده شود.

**توجه:** از آنجا که بورد FSB310 سیگنال­های کنترلی و شمارنده خود را از بورد LCB216 دریافت می­کند، بورد FSB در مد «*دریافت در حالت Slave»* ارتباط I2C فعالیت می­کند. این مد با ثابت I2C\_SR\_MODE در ماژول ارتباط I2C نشان داده می­شود.

**یادآوری:** گزارش «*ارتباط میان بورد LCB216 و FSB310*» حاوی جزئیات پیاده­سازی ارتباط I2C در بورد LCB216 با بورد FSB310 می­باشد که توصیه می­شود برای هرگونه بروزرسانی سیگنال­دهی در بورد FSB310، این گزارش نیز مطالعه شود.

**توجه:** تعاریفی که در ادامه در مورد ارتباط میان بورد FSB310 و بورد LCB216 خواهند آمد منحصر به ارتباط میان همین دو بورد می­باشند. استفاده از بورد FSB310 در کنار سایر بوردهای LCB غیر ممکن   
می­باشد.

**توجه:** سیگنال­دهی­هایی که در ادامه خواهند براساس نیازهای بورد FSB310 در بورد CIB216 تعریف   
شده­اند. از این سیگنال­دهی­ها و تعاریف تا این لحظه برای ارتباط با هیچ بورد دیگری غیر از FSB310 استفاده نشده است.

### سیگنال­های کنترلی در بورد LCB216

در بورد LCB216، به هر یک از چهار سیگنال کنترلی EMR، TRG، OVL و DOB و همچنین حالت عدم وجود هرگونه سیگنال کنترلی یک عدد هشت بیتی نسبت داده شده است که بیت MSB آنها برابر یک می­باشد. این مقادیر در جدول ‏2‑7 خلاصه شده­اند. در مورد منطق انتخاب این اعداد به گزارش «*ارتباط میان بورد LCB216 و FSB310*» رجوع شود.

جدول ‏2‑7- مقادیر اختصاص یافته به سیگنال­های کنترلی در قاب سیگنال کنترل پروتکل I2C

|  |  |
| --- | --- |
| سیگنال | مقدار اختصاص یافته |
| عدم وجود سیگنال کنترلی | 0xFB |
| سیگنال EMR | 0xFC |
| سیگنال TRG | 0xFD |
| سیگنال OVL | 0xFE |
| سیگنال DOB | 0xFF |

### سیگنال شمارنده در بورد LCB216

بورد LCB216 قادر است شماره توقف فعلی را نیز از طریق ارتباط I2C به بورد FSB310 ارسال کند. شماره توقف­ها در قالب یک هشت بیتی به بورد FSB310 ارسال می­شوند که بیت MSB آنها همیشه برابر یک است. بورد قادر است 122 توقف مختلف را آدرس­دهی کند، که اولین توقف با 0x81 و آخرین توقف این مجموعه نیز با 0xFA نمایش داده می­شود. جدول ‏2‑8 این اطلاعات را خلاصه می­کند. توجه داریم نمایش هگز پنج سیگنال کنترلی ارائه شده در بخش قبل (یعنی 0xFB برای حالت عدم وجود سیگنال کنترلی و ...) بعد از نمایش هگز شماره توقف 122 (یعنی 0xFA) قرار می­گیرند.

جدول ‏2‑8- شماره توقف ارسالی از بورد CIB

|  |  |
| --- | --- |
| سیگنال | مقدار اختصاص یافته |
| توقف یک | 0x81 |
| توقف دو | 0x82 |
|  |  |
| توقف 122 | 0xFA |

علاوه بر ارسال شماره توقف، بورد LCB216 قادر است وضعیت توقف را نیز برای بورد FSB310 ارسال کند. یکی از مهمترین تفاوت­های سیگنال وضعیت توقف ارسالی در بورد LCB216 به FSB310 با استانداری که در بخش ‏2-7-3- توصیف شد، استفاده از دو سون سگمنت کامل برای ارسال و دریافت سیگنال است. هر کدام از این سون سگمنت­ها متناظر با یک بایت در نظر گرفته می­شوند که بایت اول برای سون سگمنت سمت راست و بایت دوم برای سون سگمنت سمت چپ استفاده می­شود. همچنین بیت MSB هر دو بایت نیز برابر صفر انتخاب می­شود تا تفاوت مشخصی میان بایت­های سیگنال­های کنترلی، سیگنال شماره توقف و سیگنال وضعیت توقف وجود داشته باشد. شکل ‏2‑18 چند نمونه از سیگنال­های تولیدی توسط این بورد را نمایش می­دهد. این بورد قادر است نه پارکینگ به علاوه یک پارکینگ مجزا، نه زیرزمین به علاوه یک زیرزمین مجزا، زیرزمین، همکف، لابی، هایپراستار، مدیریت، بایگانی، و ... و همینطور 99 وضعیت توقف دیگر را سیگنال­دهی کند. جدول ‏2‑8، فهرست کاملی از سیگنال­های شمارنده در این بورد را نمایش می­دهد.

**توجه:** ممکن است علاوه بر وضعیت طبقاتی که در جدول ‏2‑8 توصیف شده­ است، در آینده نیز وضعیت طبقاتی دیگری به بورد FSB310 اضافه شود. در این صورت لازم است سیگنال­دهی متناظر با این وضعیت­ها در بورد LCB216 نیز اعمال شود.

(الف) پارکینگ پنجم (ب) پارکینگ

(ج) زیرزمین (د) مدیریت

(ه) هایپراستار (و) طبقه بیستم

شکل ‏2‑18: چند نمونه سیگنال شمارنده تولید شده در بورد LCB216. منظور از بایت اول و دوم در شکل­های فوق، بایت­های متناظر در سیگنال شمارنده است.

جدول ‏2‑8- مقادیر اختصاص یافته به سیگنال­های شمارنده در قاب سیگنال شمارنده پروتکل I2C

| سیگنال | مقدار اختصاص یافته | | سیگنال | مقدار اختصاص یافته | | سیگنال | مقدار اختصاص یافته | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| بایت اول | بایت دوم | بایت اول | بایت دوم | بایت اول | بایت دوم |
| پارکینگ نهم | 0x10 | 0x0C | طبقه ششم | 0x02 | 0x7F | طبقه  سی و یکم | 0x79 | 0x30 |
| پارکینگ هشتم | 0x00 | 0x0C | طبقه هفتم | 0x78 | 0x7F | طبقه  سی و دوم | 0x24 | 0x30 |
| پارکینگ هفتم | 0x78 | 0x0C | طبقه هشتم | 0x00 | 0x7F | طبقه  سی و سوم | 0x30 | 0x30 |
| پارکینگ ششم | 0x02 | 0x0C | طبقه نهم | 0x10 | 0x7F | طبقه  سی و چهارم | 0x19 | 0x30 |
| پارکینگ پنجم | 0x12 | 0x0C | طبقه دهم | 0x20 | 0x79 | طبقه  سی و پنجم | 0x12 | 0x30 |
| پارکینگ چهارم | 0x19 | 0x0C | طبقه یازدهم | 0x79 | 0x79 | طبقه  سی و ششم | 0x02 | 0x30 |
| پارکینگ سوم | 0x30 | 0x0C | طبقه دوازدهم | 0x24 | 0x79 | طبقه  سی و هفتم | 0x78 | 0x30 |
| پارکینگ دوم | 0x24 | 0x0C | طبقه سیزدهم | 0x30 | 0x79 | طبقه  سی و هشتم | 0x00 | 0x30 |
| پارکینگ اول | 0x79 | 0x0C | طبقه چهاردهم | 0x19 | 0x79 | طبقه  سی و نهم | 0x10 | 0x30 |
| پارکینگ | 0x0C | 0x7F | طبقه پانزدهم | 0x12 | 0x79 | طبقه چهلم | 0x20 | 0x19 |
| زیرزمین نهم | 0x10 | 0x03 | طبقه شانزدهم | 0x02 | 0x79 | لابی  (L) | 0x47 | 0x7F |
| زیرزمین هشتم | 0x00 | 0x03 | طبقه هفدهم | 0x78 | 0x79 | همکف  (G ناکامل) | 0x42 | 0x7F |
| زیرزمین هفتم | 0x78 | 0x03 | طبقه هجدهم | 0x00 | 0x79 | مدیریت[[28]](#footnote-28)  (h برعکس) | 0x25 | 0x7F |
| زیرزمین ششم | 0x02 | 0x03 | طبقه نوزدهم | 0x10 | 0x79 | هایپراستار  (h) | 0x0B | 0x7F |
| زیرزمین پنجم | 0x12 | 0x03 | طبقه بیستم | 0x20 | 0x24 | هایپرمی (HE) | 0x06 | 0x09 |
| زیرزمین چهارم | 0x19 | 0x03 | طبقه  بیست و یکم | 0x79 | 0x24 | بایگانی (a) | 0x20 | 0x7F |
| زیرزمین سوم | 0x30 | 0x03 | طبقه  بیست و دوم | 0x24 | 0x24 | رستوران  (r) | 0x0F | 0x7F |
| زیرزمین دوم | 0x24 | 0x03 | طبقه  بیست و سوم | 0x30 | 0x24 | روف گاردن (rG) | 0x42 | 0x0F |
| زیرزمین اول | 0x79 | 0x03 | طبقه  بیست و چهارم | 0x19 | 0x24 | فود کورت  (FC) | 0x46 | 0x0E |
| زیرزمین | 0x03 | 0x7F | طبقه  بیست و پنجم | 0x12 | 0x24 | شهربازی  (PG) | 0x42 | 0x0C |
| طبقه اول | 0x79 | 0x7F | طبقه  بیست و ششم | 0x02 | 0x24 | پنت هاوس  (PH) | 0x09 | 0x0C |
| طبقه دوم | 0x24 | 0x7F | طبقه  بیست و هفتم | 0x78 | 0x24 | استخر  (PL) | 0x47 | 0x0C |
| طبقه سوم | 0x30 | 0x7F | طبقه  بیست و هشتم | 0x00 | 0x24 |  |  |  |
| طبقه چهارم | 0x19 | 0x7F | طبقه  بیست و نهم | 0x10 | 0x24 |  |  |  |
| طبقه پنجم | 0x12 | 0x7F | طبقه سی­ام | 0x20 | 0x30 |  |  |  |

### دریافت سیگنال­های کنترلی و شمارنده در بورد FSB310

شکل ‏2‑19 *چینش فریم­های متوالی دریافتی سیگنال کنترلی و شمارنده را در ارتباط* I2C نمایش می­دهد. همانگونه که مشاهده می­شود اولین بیت در فریم سیگنال کنترلی و همینطور فریم متناظر با شماره توقف برابر یک است. اما در فریم­های سیگنال شمارنده وضعیت توقف، اولین بیت برابر صفر است که باعث می­شود میان این دو دسته سیگنال تمایز برقرار شود. بیت­های معرف اطلاعات در این سه قاب پس از این بیت ارسال می­شوند. توجه داریم انتخاب بیت MSB متفاوت برای سیگنال­ها در کنار تخصیص دادن هفت بیت به سیگنال­های کنترلی و شماره توقف باعث می­شود که:

1. سیگنال­دهی با کمینه تعداد بیت انجام شود
2. به دلیل تفاوت میان بیت MSB سیگنال­ها، همپوشانی میان آنها کم باشد و در نتیجه بتوان در زمان تولید ACK و NACK در پروتکل I2C، اطمینان حاصل کرد که یک سیگنال وضعیت توقف اشتباها به جای یک سیگنال کنترلی و یا شماره توقف آشکار نمی­شود.

توجه داریم علاوه بر سازوکار تولید ACK و NACK در پروتکل، ماژول­های ورودی / خروجی نرم­افزار نیز وظیفه بررسی صحت دریافت سیگنال­های کنترلی و شمارنده را برعهده دارند (بخش ‏4-3-3-). به این ترتیب امکان اشتباه در آشکارسازی سیگنال با وجود دو لایه آشکارسازی سیگنال ورودی در بورد FSB310 بسیار کم می­باشد.



شکل ‏2‑19: قاب­های ارسالی داده از بورد LCB216 به بورد FSB310 در پروتکل I2C. قاب اول دوم برای ارسال سیگنال کنترلی و شماره توقف فعلی آسانسور استفاده می­شوند و بیت MSB آنها همیشه یک است. قاب­های سوم و چهارم برای ارسال وضعیت توقف استفاده   
می­شوند و بیت MSB آنها همیشه صفر است. بایت اول و دوم سیگنال شمارنده با استفاده از جدول ‏2‑7 تعیین می­شود.

## کنترل کیفیت (QC) بورد

یکی از مهمترین دلایل کند شدن چرخه تولید بورد FSB200a، پیچیده بودن فرآیند تست این بورد است. در واقع برای تست بورد FSB200a تمامی سیگنال­های ورودی به بورد داده شده و تمامی وضعیت­ها و پیام­های متناظر تا انتها مورد بررسی قرار می­گیرند. توجه داریم اگر بخواهیم این چرخه را در نسخه­های FSB300 و FSB310 نیز پیاده کنیم، آنگاه به عنوان مثال تنها برای بررسی صدای تمامی طبقات با فرض اینکه پس از اعلان وضعیت توقف10 ثانیه موسیقی پخش شود مجموعاً به حدود 320 ثانیه (5 دقیقه) زمان نیاز است. برای حل این مشکل، یک آزمون ساده کنترل کیفی تعریف می­شود که با استفاده از آن:

1. صحت کارکرد پورت­های ورودی و کلید­های فشاری
2. صحت پروگرام صحیح میکروکنترلر و حافظه فلش

تایید می­شود. این آزمون QC به ساده­ترین شکل ممکن صحت کارکرد تمامی قسمت­های مدار را با ضریب دقت بالایی تضمین می­کند و در عین حال به سرعت قابل اجرا می­باشد. آزمون QC بورد FSB300 در مد تست این بورد که در ادامه در مورد آن بحث می­شود صورت می­پذیرد، اما در مورد بورد FSB310 آزمون QC با ایجاد یک سیگنال کنترلی در بورد LCB صورت خواهد پذیرفت.

### مد تست بورد FSB300

کنترل کیفیت بورد FSB300 در مد تست بورد انجام می­پذیرد. مد تست با فشردن کلید TEST در POR قابل اجرا است و با استفاده از آن می­توان صحت اتصال تمامی ورودی­های بورد را بررسی نمود. پس از تایید صحت ورودی­ها، صحت کارکرد دیگر کلید فشاری بورد با کاهش سطح صدای بورد در خارج از مد تست مورد تایید قرار می­گیرد. در مد تست، با تحریک هر یک از ورودی­های بورد FSB شماره یک طبقه متناظر با شماره پایه (طبقه یک برای پایه یک که پایه DOB است، طبقه دو برای پایه دو که OVL است و ...) پخش می­شود. به این ترتیب اولاً   
می­توان نتیجه گرفت که ورودی­های بورد به صورت صحیح کار می­کنند و ثانیاً پروگرام تمامی قطعات به صورت صحیحی انجام پذیرفته است[[29]](#footnote-29).

قبل از توضیح چگونگی اعمال سیگنال­های ورودی مناسب در مد تست، لازم به یادآوری است که همانگونه که در بخش 2-7 اشاره شد، به غیر از سیگنال DOB که با سطح منطقی پیش فرض صفر در بورد   
پیاده­سازی می­شود و همینطور سیگنال EMR که AH است (و بنابراین مقدار پیش فرض صفر دارد)، سایر ورودی­ها با سطح منطقی پیش فرض یک در بورد پیاده­سازی شده­اند. در وضعیت تست برای بررسی صحت کارکرد هر ورودی، سیگنال مخالف مقدار منطقی پیش فرض به آن ورودی داده می­شود و سایر ورودی­ها بدون تغییر می­مانند. در این حالت یک صدای طبقه در بورد پخش خواهد شد که نشان­دهنده صحت کارکرد آن پین است. به بیان­ دیگر، برای آزمایش پین­های به غیر از DOB و EMR باید به آن پین مقدار صفر اعمال کرد (سایر پین­ها بدون ورودی   
می­مانند) و در مورد دو پین DOB و EMR باید منطق یک به این پین­ها اعمال شود (سایر پین­ها بدون ورودی   
می­مانند). با در نظر گرفتن موارد فوق، سطح منطقی پورت­های کنترلی و شمارنده که برای آنها موسیقی پخش خواهد شد به شرح جدول ‏2‑10 می­باشد. توجه داریم در مد تست نباید دو ورودی به صورت همزمان به دو پین مختلف بورد داده شود، به همین دلیل سطح منطقی تمام ورودی­ها به صورت همزمان در جدول ‏2‑10 آورده شده است تا نشان داده شود سایر ورودی­ها بدون مقدار باقی می­مانند. همچنین در مد تست، تمامی پایه­ها به صورت همزمان از بورد خوانده می­شوند، و در پیاده­سازی هشت بیت سیگنال شمارنده به عنوان بیت­های با اهمیت یک متغیر شانزده­بیتی در نظر گرفته می­شوند که شامل بیت­های سیگنال شمارنده و کنترلی است (مشابه جدول ‏2‑10).

یادآوری می­شود که مد تست بورد با پیش فرض عدم اتصال جامپرهای تغییر سطح منطقی سیگنال­های DOB و TRG در بورد پیاده­سازی می­شود و در صورت اتصال این جامپرها، تست بورد ممکن نمی­باشد. بنابراین در صورتی که قرار است از جامپرها استفاده شود باید ابتدا صحت کارکرد مدار قبل از اتصال جامپرها در مد تست بررسی شود، و سپس جامپرها در بورد متصل شوند.

جدول ‏2‑10- سطح منطقی پورت­ها برای آزمایش پین­های ورودی بورد FSB300

| پین مورد آزمایش | سیگنال خوانده شده از پورت C و D­[[30]](#footnote-30) | V+ | V- | EMR | DOB | OVL | TRG | B2 | G2 | A1 | B1 | C1 | E1 | F1 | G1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (PC0) | (PC1) | (PC2) | (PC3) | (PC4) | (PC5) | (PD0) | (PD1) | (PD2) | (PD3) | (PD4) | (PD5) | (PD6) | (PD7) |
| EMR | 0xFF34 |  |  | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| TRG | 0xFF10 |  |  | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| OVL | 0xFF20 |  |  | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| DOB | 0xFF38 |  |  | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| B2 | 0xFE30 |  |  | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| G2 | 0xFD30 |  |  | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| A1 | 0xFB30 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| B1 | 0xF730 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| C1 | 0xEF30 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| E1 | 0xDF30 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| F1 | 0xBF30 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| G1 | 0x7F30 |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

### آزمون QC در بورد FSB310

از آنجا که بورد FSB310 دارای تنها دو ورودی I2C می­باشد که سیگنال­های آن از طریق بورد LCB ارسال می­شوند، آزمون QC این بورد تنها پس از سوار کردن بورد FSB310 بر روی بورد LCB ممکن می­باشد. پس از سوار کردن این بورد بر روی بورد اصلی می­توان سیگنال TRG را به همراه یک شماره توقف و همینطور وضعیت توقف مشخص به بورد FSB310 ارسال کرد تا صحت سیگنال­دهی در کنار پخش موسیقی در بورد تعیین شود. در نهایت با کاهش و افزایش سطح صدای بورد می­توان صحت کارکرد کلیدهای فشاری آن را تعیین نمود.

## تطبیق بورد بورد FSB300 با ورودی­های AL و AH (مد یادگیری)

همانگونه که می­دانیم سیگنال DOB در بورد FSB300 می­تواند دارای منطق AH و AL باشد که هر کدام از منطق­ها می­توانند توسط کنتاکتورهای Normally Open (NO) و یا Normally Closed (NC) ایجاد شوند. از طرف دیگر سیگنال TRG نیز می­تواند منطق AH و یا AL داشته باشد. به این ترتیب تطبیق دادن بورد FSB300 با سطح منطقی هر یک از این ورودی­ها از اهمیت خاصی برخوردار است.

تطبیق بورد با سطوح مختلف منطقی سیگنال DOB و یا TRG دارای دو سطح نرم­افزاری و سخت­افزاری   
می­باشد. در سطح سخت­افزاری، نحوه بایاس پین­های ورودی باید به گونه­ای تنظیم شود که در حالت عدم اتصال ورودی، پین DOB با منطق سیگنال ورودی DOB یکسان باشد و پین TRG عکس منطق آن را داشته باشد. چگونگی انجام اینکار در فصل پنجم آمده است. در بخش نرم­افزاری نیز لازم است شکل سیگنال­های کنترلی اشاره شده در بخش 2-7-2 بر اساس منطق سیگنال­ها در کد اصلاح شود. این عمل را می­توان در مد یادگیری بورد FSB انجام داد که در ادامه در مورد آن صحبت خواهد شد.

### مد یادگیری

همانگونه که در بالا اشاره شد هدف از مد یادگیری اصلاح شکل سیگنال­ها با توجه به ورودی­های AL و AH برای DOB و TRG می­باشد. با توجه به ترتیبی که برای اولویت سیگنال­های کنترلی در بخش 2-7-2 تعریف شد، واضح است که تغییر سطح منطقی یکی از این دو سیگنال می­تواند باعث تغییر شکل سیگنال­های TRG، OVL و DOB شود، اما این تغییر بر سیگنال EMR تاثیری نمی­گذارد. بنابراین لازم است تمامی این سیگنال­ها با توجه به سطح منطقی ورودی­ها اصلاح شوند که این عمل در مد یادگیری انجام می­شود.

وضعیت یادگیری بگونه­ای تعریف شده است که با فشردن کلید کاهش صدا (کلید یادگیری) و تنها در POR فعال می­شود. در این وضعیت سیگنال­های DOB و TRG برای یک مدت زمانی مشخص از ورودی­ها خوانده می­شوند و پس از آن شکل سیگنال­های کنترلی تنظیم می­شود. برای انجام این عمل نیز لازم است منطقی که باعث فعال شدن یک پین مشخص می­شود به آن داده شود. این بدان معناست که (با فرض تنظیم صحیح جامپرهای بورد) هیچ سیگنالی به DOB داده نمی­شود و TRG نیز منطقی را که به عنوان فعال بودن این سیگنال تعبیر می­شود دریافت می­کند. جدول ‏2‑12 نشان دهنده منطق سیگنال­ها با توجه به سطح ورودی­­های خوانده شده از پین آنها در مد یادگیری می­باشد که در آن AL یا AH قرار گرفته شده بعد از نام هر سیگنال نشان دهنده سطح منطقی فعال­سازی آن می­باشد. با اجرای مد یادگیری، بورد FSB به صورت خودکار خود را با این سطوح ورودی تطبیق می­دهد.

جدول ‏2‑12- سطح منطقی پورت­ها در حالت عدم حضور ورودی و تنظیم مناسب جامپرها در مد یادگیری

|  | | سیگنال­های پورت کنترلی به ترتیب از کم ارزش­ترین بیت (PC0) به با ارزش­ترین بیت (PC5) | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| منطق سیگنال­ها | سیگنال پورت کنترلی متناظر پس از حذف بیت­های بدون اهمیت | V+ | V- | EMR | DOB | OVL | TRG |
| (PC0) | (PC1) | (PC2) | (PC3) | (PC4) | (PC5) |
| DOBAL\_TRGAL | 0x00 |  |  |  | 0 |  | 0 |
| DOBAH\_TRGAL | 0x08 |  |  |  | 1 |  | 0 |
| DOBAH\_TRGAH | 0x28 |  |  |  | 1 |  | 1 |
| DOBAL\_TRGAH | 0x20 |  |  |  | 0 |  | 1 |

### ترتیب اجرای مد تست و مد یادگیری

همانگونه که در بالا اشاره شد مد تست با فرض AL بودن سیگنال DOB و AH بودن سیگنال TRG   
پیاده­سازی شده است. از آنجا که مد یادگیری سطح منطقی این دو سیگنال را (به صورت بالقوه و با در نظر گرفتن تغییرات سخت­افزاری بورد) دستخوش تغییر می­کند، بنابراین لازم است مد تست قبل از مد یادگیری در بورد   
پیاده­سازی شود. عدم رعایت ترتیب فوق باعث می­شود که نتوان آزمون کنترل کیفی را به صورت صحیح در مد تست انجام داد.



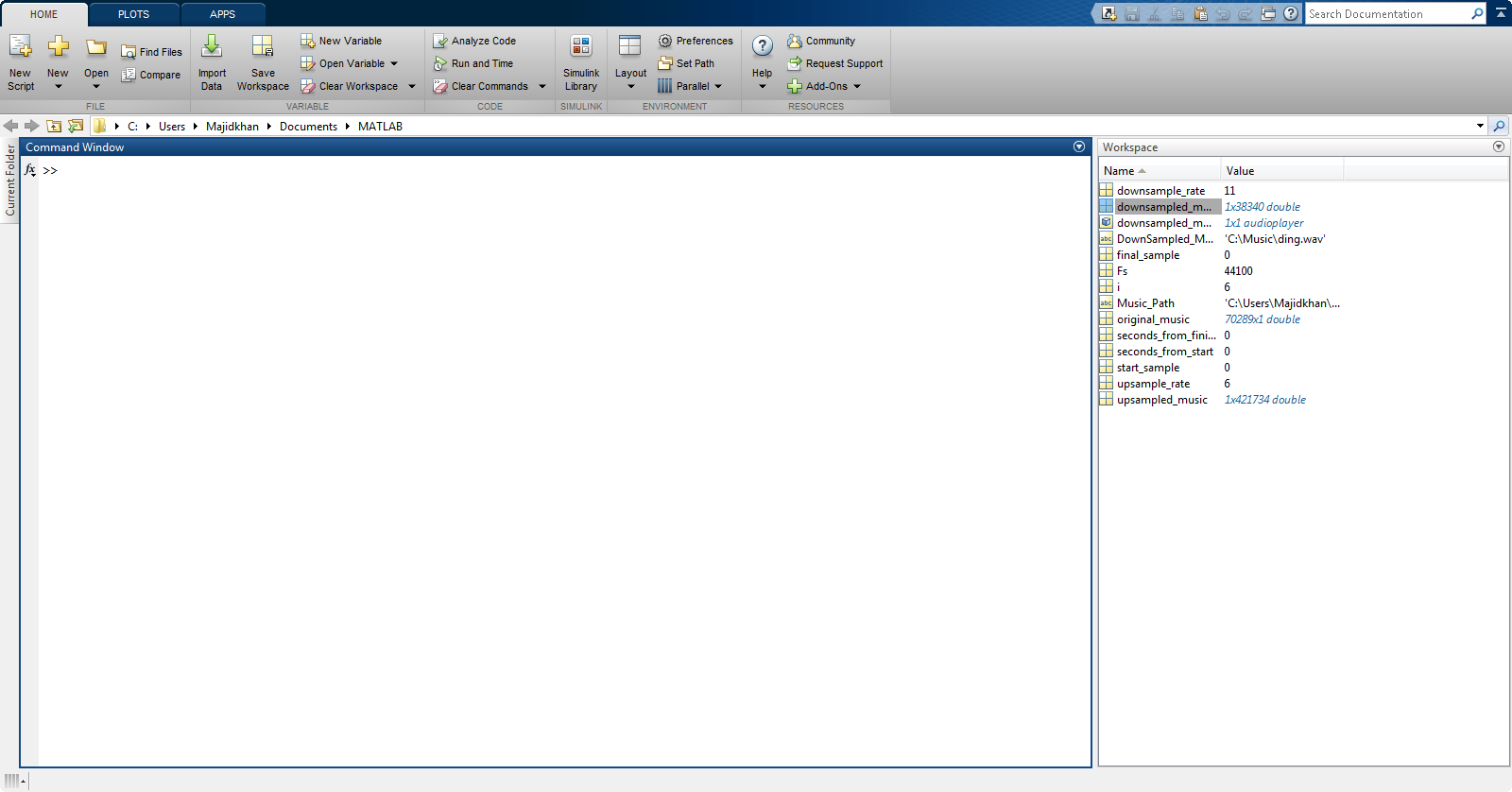
# تولید فایل­های موسیقی

## مقدمه

از آنجا که بورد FSB برای پخش فایل­های موسیقی wav با فرمت هشت بیتی و فرکانس نمونه­برداری 16kHz طراحی شده است، لازم است فایل­های موسیقی ذخیره شده در حافظه آن نیز با این فرمت تطبیق پیدا کنند. در این فصل چگونگی تولید فایل­های موسیقی منطبق با این فرمت با استفاده از نرم­افزار مَتلَب (Matlab) تشریح می­شود. برای سادگی کار، یک واسط گرافیکی در محیط متلب طراحی شده است که کار با آن بسیار ساده می­باشد و می­تواند علاوه بر تولید فایل­های موسیقی، آدرس متناظر با صفحات شروع و پایان هر پیام/ موسیقی را نیز معین کند. بنابراین در بخش اول این فصل، به معرفی این واسط پرداخته می­شود و سپس در بخش­های بعد در مورد جزئیات نوشتار کد متلب این واسط توضیح داده خواهد شد.

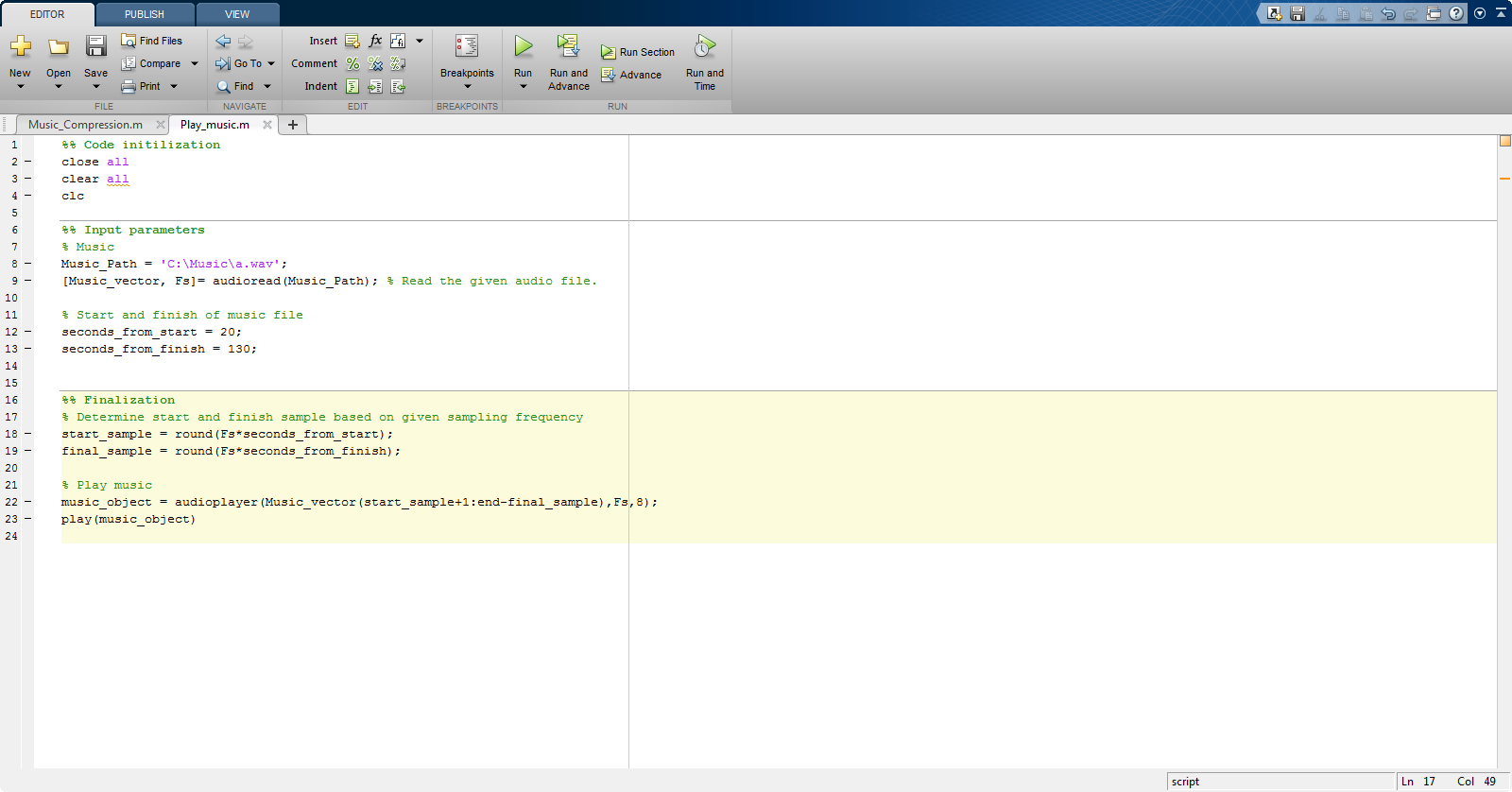
## واسط گرافیکی ویرایش فایل­های موسیقی برای بورد FSB

برای استفاده از این واسط گرافیکی لازم است ابتدا درباره چگونگی اجرای برنامه­ها در محیط نرم­افزار متلب توضیحاتی ارائه شود. برای استفاده از واسط گرافیکی، استفاده از نرم­افزار متلب 2014 (Matlab 2014b) توصیه   
می­شود. پس از اجرای نرم­افزار متلب، پنجره زیر بر روی صفحه نمایش ظاهر می­شود.



شکل ‏3‑1: محیط command widnow نرم­افزار متلب

این پنجره اصطلاحا command window نرم افزار است. با استفاده از گزینه open در بالا سمت چپ، می­توان فایل نرم افزار دلخواه را در محیط MATLAB باز کرد. در این حالت پنجره جدیدی باز خواهد شد که به صورت زیر است:



شکل ‏3‑2: محیط اجرای برنامه (کامپایلر) متلب

پنجره فوق محیط اجرای نرم­افزارها است. با استفاده از دکمه سبز رنگ Run که در بالا مشخص شده است، می­توان کد دلخواه را اجرا نمود.

برای اجرای واسط گرافیکی، لازم است کد Main\_Program.m در محیط کامپایلر اجرا شود. پس از اجرای این کد، پنجره زیر در نرم­افزار متلب مشاهده خواهد شد. این پنجره اصطلاحاً واسط گرافیکی نرم­افزار (GUI)[[31]](#footnote-31) نام دارد.



شکل ‏3‑3: محیط واسط گرافیکی ویرایش فایل­های موسیقی

همانگونه که مشاهده می­شود این واسط دارای سه بخش اصلی می­باشد. در بخش اول نوع بوردی که قرار است فایل­ها برای آن تنظیم شوند انتخاب می­شود. در قسمت دوم می­توان یک تک فایل موسیقی با فرمت دلخواه (مثلاً پیام اورژانسی، موسیقی طبقات و ...) را می­توان به فرمت مورد استفاده در بورد FSB تبدیل نمود، و بخش سوم برای تولید یک فایل موسیقی پیوسته (مورد استفاده در فلش) مورد استفاده قرار می­گیرد.

برای تشریح کارکرد واسط نرم­افزاری، فرض کنیم قرار است فایل­های موسیقی موجود در پوشه C:\Music را به فرمت بورد FSB300 تبدیل کنیم. برای اینکار با کلیک بر روی دکمه FSB300، نحوه انتخاب شماره فایل برای هر پیام / موسیقی مورد نظر معین می­شود. بنابراین پیام اورژانسی باید یا شماره 00 و مثلاً لابی با شماره 71 نام­گذاری شود. برای تطبیق دادن فرمت هر یک از این فایل­ها با بورد FSB300، ابتدا مسیر پوشه فایل­های موسیقی یعنی C:\Music را در فضای خالی متناظر تایپ می­کنیم. سپس شماره فایلی که قرار است تبدیل شود، مثلاً فایل 08، را در فضای خالی دوم تایپ می­کنیم. توجه داریم اگر یک فایل (مثلاً آلبوم اضافی یک) وجود ندارد، آنگاه لازم نیست شماره فایل متناظر آن (یعنی 79 در این مثال خاص) در این فضای خالی تایپ شود و این موسیقی/ پیام در بورد خالی در نظر گرفته خواهد شد. در قسمت سوم، فرمت فایل موسیقی را که می­تواند به عنوان مثال mp3، wav و یا هر فرمت دلخواهی باشد تایپ می­کنیم. فولدر چهارم، معرف نشانی پوشه­ای است که فایل­های موسیقی منطبق با بورد FSB در آن ذخیره می­شوند. به این منظور به عنوان مثال پوشه FSB Music در درایو C ساخته می­شود که درون آن خالی بوده و در قسمت فوق آدرس C:\FSB Music نوشته می­شود. حال فرض کنیم فایل موسیقی شماره 08 دارای یک قسمت خالی به مدت 0.5 ثانیه در ابتدای فایل و یک قسمت خالی به مدت 2 ثانیه در انتهای آن   
می­باشد که می­توان آن را از ابتدا و انتهای فایل برش داد. با استفاده از دو فیلد بعدی می­توان این چند ثانیه را از ابتدا و انتهای فایل حذف نمود. برای این منظور عدد 0.5 در فیلد بالایی (ثانیه­هایی که از شروع فایل حذف می­شوند)، و عدد دو در فیلد پایینی (ثانیه­هایی که از انتهای فایل حذف می­شوند) وارد می­شوند. توجه داریم برای اینکه مقدار دقیق دو عدد 0.5 و 2 معین شود، نیازی به پخش فایل موسیقی در نرم­افزارهای پخش موسیقی در محیط ویندوز (مثل media player و غیره) نمی­باشد. در واقع پس از فشردن کلید «*تولید فایل جدید*»، فایلی که ابتدای آن به اندازه 0.5 ثانیه و انتهای آن به اندازه 2 ثانیه قطع شده است و منطبق با بورد FSB می­باشد در محیط نرم­افزار متلب پخش خواهد شد. در صورتی که این مقادیر مناسب نباشد، می­توان این دو مقدار زمانی را به اندازه دلخواه تغییر داد و مجدداً فایل موسیقی را تولید نمود. توجه داریم عملیات تولید فایل موسیقی جدید باید برای تمامی فایل­های صوتی مورد استفاده در بورد FSB تکرار شود.

پس از تبدیل تک فایل­های موسیقی به فرمت wav مورد استفاده در بورد FSB، نوبت به تولید یک فایل پیوسته از موسیقی­ها می­رسد. برای انجام اینکار می­توان از قسمت پایینی واسط گرافیکی استفاده نمود و اگر ترتیب فایل­های موسیقی به درستی رعایت شده باشد، فایل پیوسته با کلیک بر روی دکمه تولید فایل پیوسته تولید خواهد شد. توجه داریم نام فایل موسیقی پیوسته Flash File.wav است و آدرس­­های متناظر با هر فایل برای استفاده در بورد FSB نیز در ابتدای این فایل پیوسته قرار داده خواهد شد (به بخش3-3-2 رجوع شود). اگر برای ذخیره فایل­ها یک فلش مورد نیاز باشد، تنها فایل Flash1\_File.wav در این پوشه ظاهر می­شود، اما در صورت نیاز به فلش دوم فایل Flash2\_File.wav نیز ظاهر خواهد شد. در یک حالت خاص ممکن است حجم بسیار کمی از فلش دوم توسط فایل آن پر شود، در این صورت نرم­افزار با ارسال یک پیام از کاربر درخواست می­کند تا در صورت امکان حجم فایل­های موجود را کاهش دهد تا بتوان آنها را در یک مموری جای داد.

توجه داریم برای اکثر خطاهای احتمالی که ممکن است در هنگام ساخت فایل ایجاد شود، نرم­افزار خود یک پیام خطا ایجاد می­کند. به عنوان مثال اگر مجموع حجم فایل­های پروگرام شده بر روی بورد از مقدار دو حافظه فلش تعبیه شده بر روی بورد بیشتر شود، نرم­افزار یک پیام خطا نشان خواهد داد و از کاربر درخواست خواهد کرد که حجم فایل­ها را کاهش دهد.

### شماره­گذاری فایل­های موسیقی مورد استفاده در بورد FSB

برای کارکرد صحیح نرم­افزار فوق، لازم است ابتدا یک پوشه در ویندوز ایجاد شود که حاوی فایل­های موسیقی است. نام فایل­ها در این پوشه تنها یک عدد است که از شماره 00 آغاز و به 80 ختم می­شود[[32]](#footnote-32)و [[33]](#footnote-33). جدول ‏3‑1 و جدول ‏3‑2 شماره فایل­ متناظر با پیام/موسیقی دلخواه را به ترتیب در بورد FSB300 و FSB310 نمایش می­دهد. توجه داریم در بورد FSB300 منظور از شماره توقف، شماره توقف نسبت داده شده به این وضعیت توقف خاص در بورد FSB300 است (به بخش ‏2-7-2- مراجعه شود).

جدول ‏3‑1: شماره فایل متناظر با پیام/موسیقی در بورد FSB300

| فایل صوتی | شماره فایل | شماره توقف متناظر | فایل صوتی | شماره فایل | شماره توقف متناظر |
| --- | --- | --- | --- | --- | --- |
| پیام اورژانسی | 00 | --- | **طبقه پانزدهم** | 21 | 27 |
| دینگ دانگ | 01 | --- | **طبقه شانزدهم** | 22 | 28 |
| پیام باز بودن درب | 02 | --- | **طبقه هفدهم** | 23 | 29 |
| پیام اضافه وزن | 03 | --- | **طبقه هجدهم** | 24 | 30 |
| دینگ | 04 | --- | **طبقه نوزدهم** | 25 | 31 |
| موسیقی (آلبوم اصلی) | 05 | --- | **پارکینگ نهم** | 47 | 1 |
| موسیقی پس از پیام اورژانسی | 06 | --- | **پارکینگ هشتم** | 48 | 2 |
| طبقه اول | 07 | 13 | **پارکینگ هفتم** | 49 | 3 |
| طبقه دوم | 08 | 14 | **پارکینگ ششم** | 50 | 4 |
| طبقه سوم | 09 | 15 | **پارکینگ پنجم** | 51 | 5 |
| طبقه چهارم | 10 | 16 | **پارکینگ چهارم** | 52 | 6 |
| طبقه پنجم | 11 | 17 | **پارکینگ سوم** | 53 | 7 |
| طبقه ششم | 12 | 18 | **پارکینگ دوم** | 54 | 8 |
| طبقه هفتم | 13 | 19 | **پارکینگ اول** | 55 | 9 |
| طبقه هشتم | 14 | 20 | **پارکینگ** | 56 | 10 |
| طبقه نهم | 15 | 21 | **زیرزمین** | 66 | 11 |
| طبقه دهم | 16 | 22 | **همکف** | 67 | 12 |
| طبقه یازدهم | 17 | 23 | **لابی** | 71 | 12 |
| طبقه دوازدهم | 18 | 24 | **آلبوم اضافی یک** | 79 | --- |
| طبقه سیزدهم | 19 | 25 | **آلبوم اضافی دو** | 80 | --- |
| طبقه چهاردهم | 20 | 26 |  |  |  |

جدول ‏3‑2: شماره فایل متناظر با پیام/موسیقی در بورد FSB310

| فایل صوتی | شماره فایل | فایل صوتی | شماره فایل | فایل صوتی | شماره فایل |
| --- | --- | --- | --- | --- | --- |
| پیام اورژانسی | 00 | **طبقه بیست و هفتم** | 33 | **زیرزمین** | 66 |
| دینگ دانگ | 01 | **طبقه بیست و هشتم** | 34 | **همکف** | 67 |
| پیام باز بودن درب | 02 | **طبقه بیست و نهم** | 35 | **هایپراستار** | 68 |
| پیام اضافه وزن | 03 | **طبقه سی­ام** | 36 | **هایپرمی** | 69 |
| دینگ | 04 | **طبقه سی و یکم** | 37 | **رستوران** | 70 |
| موسیقی (آلبوم اصلی) | 05 | **طبقه سی و دوم** | 38 | **لابی** | 71 |
| موسیقی پس از پیام اورژانسی | 06 | **طبقه سی و سوم** | 39 | **مدیریت** | 72 |
| طبقه اول | 07 | **طبقه سی و چهارم** | 40 | **بایگانی** | 73 |
| طبقه دوم | 08 | **طبقه سی و پنجم** | 41 | **روف گاردن** | 74 |
| طبقه سوم | 09 | **طبقه سی و ششم** | 42 | **فود کورت** | 75 |
| طبقه چهارم | 10 | **طبقه سی و هفتم** | 43 | **شهربازی** | 76 |
| طبقه پنجم | 11 | **طبقه سی و هشتم** | 44 | **پنت هاوس** | 77 |
| طبقه ششم | 12 | **طبقه سی و نهم** | 45 | **استخر** | 78 |
| طبقه هفتم | 13 | **طبقه چهلم** | 46 | **آلبوم اضافی یک** | 79 |
| طبقه هشتم | 14 | **پارکینگ نهم** | 47 | **آلبوم اضافی دو** | 80 |
| طبقه نهم | 15 | **پارکینگ هشتم** | 48 | **تعریف نشده** | 81 |
| طبقه دهم | 16 | **پارکینگ هفتم** | 49 | **تعریف نشده** | 82 |
| طبقه یازدهم | 17 | **پارکینگ ششم** | 50 |  |  |
| طبقه دوازدهم | 18 | **پارکینگ پنجم** | 51 | **تعریف نشده** | 149 |
| طبقه سیزدهم | 19 | **پارکینگ چهارم** | 52 |  |  |
| طبقه چهاردهم | 20 | **پارکینگ سوم** | 53 |  |  |
| طبقه پانزدهم | 21 | **پارکینگ دوم** | 54 |  |  |
| طبقه شانزدهم | 22 | **پارکینگ اول** | 55 |  |  |
| طبقه هفدهم | 23 | **پارکینگ** | 56 |  |  |
| طبقه هجدهم | 24 | **زیرزمین نهم** | 57 |  |  |
| طبقه نوزدهم | 25 | **زیرزمین هشتم** | 58 |  |  |
| طبقه بیستم | 26 | **زیرزمین هفتم** | 59 |  |  |
| طبقه بیست و یکم | 27 | **زیرزمین ششم** | 60 |  |  |
| طبقه بیست و دوم | 28 | **زیرزمین پنجم** | 61 |  |  |
| طبقه بیست و سوم | 29 | **زیرزمین چهارم** | 62 |  |  |
| طبقه بیست و چهارم | 30 | **زیرزمین سوم** | 63 |  |  |
| طبقه بیست و پنجم | 31 | **زیرزمین دوم** | 64 |  |  |
| طبقه بیست و ششم | 32 | **زیرزمین اول** | 65 |  |  |

## پردازش­ها برای تولید فایل­های موسیقی با استفاده از نرم­افزار متلب

در این قسمت به صورت تفصیلی در مورد کدهای متلب نوشته شده برای پردازش فایل­های موسیقی توضیح داده خواهد شد.

### کد Music\_Compression

از این کد برای تبدیل یک فایل موسیقی با فرمت دلخواه به فرمت مورد استفاده در بورد FSB استفاده   
می­شود. این کد شامل پنج بخش اصلی (به ترتیب) خواندن تک فایل موسیقی دلخواه و استخراج نرخ نمونه­برداری آن، حذف بخش­های اضافی از ابتدا و انتهای فایل و فیلتر کردن موسیقی، تبدیل نرخ نمونه­برداری سیگنال موسیقی به مقدار پیاده­سازی شده در بورد FSB به کمک زیرنمونه­برداری (downsampling) و نمونه­گذاری (upsampling)، و در نهایت تولید فایل موسیقی جدید است. در ادامه شرح کامل هر یک از این مراحل خواهد آمد.

* **خواندن فایل موسیقی**

برای خواندن یک فایل موسیقی در محیط متلب از تابع audioread استفاده می­شود. ورودی این تابع مسیر فایل مورد نظر است که این مسیر به صورت یک رشته کاراکتری داده می­شود (که این رشته در واسط گرافیکی توسط کاربر فراهم می­شود). این تابع شامل دو خروجی است که خروجی بردار نمونه­های صوت است که به فرمت double خوانده می­شوند. این نمونه­ها در این فایل در بردار original\_music ذخیره می­شوند. خروجی دوم نرخ نمونه­برداری متناظر با این فایل موسیقی است که در این کد در متغیر Fs ذخیره می­شود. بنابراین اگر Fs برابر 32kHz باشد و فایل موسیقی یک ثانیه باشد، آنگاه 32 نمونه در بردار original\_music ذخیره خواهند شد.

* **بررسی تعریف شدن فایل**

در این قسمت تعریف می­شود که آیا برای هر یک از بوردهای FSB300 و FSB310 شماره فایل ورودی توسط کاربرد به صورت صحیح تعریف شده است که اینکار توسط یک حلقه switch در توابع Check\_File\_Num\_FSB310() و Check\_File\_Num\_FSB300() صورت می­پذیرد. در ادامه نیز بررسی می­شود که در صورتی که یک فایل که شماره آن تعریف نشده اعلان شود، آنگاه یک خطا در نرم­افزار اعلان شود

* **حذف قسمت­های اضافی فایل موسیقی و فیلترینگ**

برای حذف n ثانیه از ابتدای فایل موسیقی، *نمونه از ابتدای فایل موسیقی حذف خواهد شد و عمل مشابهی نیز برای حذف چند ثانیه از انتهای فایل موسیقی انجام می­شود.*

*برای فیلترینگ فایل موسیقی ابتدا لازم است یک فیلتر دیجیتال مناسب در محیط نرم­افزار متلب تولید شود. برای اینکار می­توان از تابع* butter *استفاده نمود که یک فیلتر باتروث[[34]](#footnote-34) پایین­گذر با مرتبه دلخواه را تولید می­کند. پارامترهای ورودی این کد عبارت است از مرتبه فیلتر (که این مرتبه برابر 8 انتخاب می­شود تا فیلتر هرچه بیشتر به فیلتر ایده­آل نزدیک باشد) و همینطور پهنای باند نرمالیزه شده فیلتر پایین­گذر به فرکانس نمونه­بردای. اگر فرض کنیم قرار است تا فرکانس* در حوزه فرکانس فیلتر شود، آنگاه پهنای باند نرمالیزه شده به صورت تعریف خواهد شد. توجه داریم خروجی تابع فوق، ضرایب صورت و مخرج تابع انتقال فیلتر فوق در حوزه فرکانس می­باشد. برای انجام عمل فیلترینگ سیگنال موسیقی از تابع filter استفاده می­شود که ضرایب فوق و نمونه­های فایل موسیقی را به عنوان ورودی دریافت و سپس سیگنال فیلتر شده را به عنوان خروجی تحویل می­دهد.

* **تبدیل نرخ نمونه­بردای فایل موسیقی**

فرض کنیم نرخ نمونه­برداری فایل موسیقی اصلی فیلتر شده[[35]](#footnote-35) برابر Fs باشد و نرخ نمونه­برداری پیاده­سازی شده در بورد FSB برابر *است. برای تبدیل* به *لازم است نمونه­ها با نسبت تغییر کند. توجه داریم در بیشتر موارد ، به همین دلیل نسبت کوچکتر از یک است. فرض کنیم با ساده کردن نسبت فوق به کسر برسیم که و هیچ عامل ضربی مشترکی ندارند (دستیابی به این کسر در متلب از طریق تابع rat صورت   
می­پذیرد). این نسبت به معنای آن است که باید تعداد نمونه­های سیگنال برابر شوند و سپس با نسبت کاهش پیدا کنند. برای برابر کردن تعداد نمونه­ها، هر نمونه سیگنال اصلی به اندازه بار تکرار می­شود. این عمل در اصلاح نمونه­گذاری (*upsampling*) نام دارد. سپس در سیگنال جدید از هر نمونه پشت سرهم یکی نگه داشته می­شود و بقیه نمونه­ها حذف می­شوند که این عمل زیرنمونه­برداری (*downsampling*) نام دارد. خروجی عملیات فوق، یک سیگنال جدید خواهد بود که نرخ نمونه­برداری آن است.*

* ***تولید فایل موسیقی جدید***

*با استفاده از بردار نمونه­های موسیقی و در دست داشتن نرخ نمونه­برداری آن می­توان یک فایل موسیقی* wav *در محیط متلب تولید نمود. برای انجام اینکار در محیط متلب از تابع* audiowrite *استفاده می­شود. این تابع علاوه بر بردار نمونه­ها و همینطور نرخ نمونه­برداری، دو ورودی دیگر نیز نیاز دارد:*

* *تعداد بیت­ها به ازای هر نمونه: این پارامتر نشان دهنده تعداد بیت تخصیص یافته به هر نمونه فایل موسیقی است که در بورد FSB این مقدار برابر هشت تعریف شده است. به این ترتیب نرخ بیت (bitrate) فایل­های پخش شده در بورد برابر است که با فرض ، نرخ بیت برابر خواهد شد.*
* *مسیر ذخیره و نام فایل جدید: نشان دهنده­ای مسیری است که فایل موسیقی در آن ذخیره می­شود که به فرمت رشته کاراکتری به تابع داده می­شود. توجه داریم فرمت فایل موسیقی پس از نام فایل می­آید. به عنوان مثال «C:\FSB Music\22.wav» یک رشته صحیح ذخیره فایل است.*

### کد Attach\_Music

از این کد برای تولید یک فایل موسیقی پیوسته استفاده می­شود که این فایل بر روی حافظه فلش بورد ذخیره می­شود. ضمن اینکه آدرس صفحه ابتدایی و انتهایی هر پیام/ موسیقی نیز با استفاده از این کد در بایت­های 45 تا 644 از ابتدای فایل قرار داده می­شود. توجه داریم از آنجا که اندازه صفحات حافظه فلش 528 بایتی است، اگر یک فایل موسیقی دارای نمونه باشد و از آنجا که هر نمونه هشت بایتی است، آنگاه این فایل صفحه حافظه فلش را اشغال می­کند.

برای تولید فایل موسیقی پیوسته در کد متلب، ابتدا هر فایل موسیقی منطبق با فرمت بورد FSB با شروع از فایل 01.wav توسط تابع audioread خوانده می­شود. سپس نمونه­های متناظر این فایل در بردار Flash\_vector ذخیره می­شوند و پس از آن نمونه­های شماره فایل بعدی بلافاصله پس از نمونه­های این فایل قرار می­گیرد. به این ترتیب بردار Flash\_vector دارای اطلاعات تمامی فایل­های موسیقی به صورت پشت سر هم خواهد شد که ترتیب قرارگیری آن مشابه ترتیب ارائه شده در جدول ‏3‑1 و جدول ‏3‑2 است. ضمن اینکه شماره صفحات متناظر در حافظه فلش نیز با روش ارائه شده در بالا قابل استخراج است. برای تعیین آدرس­ صفحات ابتدایی و انتهایی هر پیام/ موسیقی کافیست تعداد صفحات اشغال شده توسط هر فایل را در زمان پشت سر هم قرار دادن آنها در نظر گرفت.

همانگونه که در فصل بعدی به صورت تفصیلی اشاره خواهد شد، شماره صفحه ابتدایی و انتهایی هر پیام/موسیقی توسط 4 بایت نمایش داده می­شود که دو هشت بیتی برای صفحه ابتدایی پیام و دو هشت بیتی برای صفحه انتهایی پیام است. علت استفاده از دو هشت بیت، عبور شماره صفحات از 256 است. ضمن اینکه این هشت بیتی­ها در ابتدای فایل موسیقی و بلافاصله پس از هدر استاندارد فایل wav ذخیره می­شوند. از آنجا که چهل فایل در بورد تعریف شده است، مجموعاً به 160 بایت داده برای ذخیره شماره صفحات احتیاج می­باشد. برای بیان شماره صفحه توسط دو بایت، خارج قسمت تقسیم شماره صفحه بر 528 و همینطور باقیمانده آن به ترتیب نماینده هشت بیت با اهمیت و هشت بیت کم اهمیت شماره صفحه هستند. حال بردار شماره صفحات یا Page\_index که یک متغیر تعریف شده در MATLAB است به این صورت تشکیل می­شود که با شروع از اولین پیام، هشت با اهمیت صفحه ابتدایی پیام/ موسیقی در اولین مکان، هشت بیت کم اهمیت صفحه ابتدایی در دومین مکان، هشت بیت با اهمیت صفحه انتهایی در سومین مکان و هشت بیت کم اهمیت صفحه انتهایی در چهارمین مکان قرار داده می­شوند. با کنار هم قرار دادن این مقادیر در کنار مقادیر فایل موسیقی و استفاده از تابع audiowrite، یک فایل پیوسته موسیقی ساخته می­شود که آدرس صفحات ابتدایی و انتهایی هر پیام/موسیقی آن نیز به صورت بایت به بایت در ابتدای فایل قرار دارند. توجه داریم از آنجا که هدر استاندارد فایل­های wave خود دارای چهل و چهار بایت پیش فرض می­باشد (به بخش 2-4-2-1 رجوع شود)، آدرس صفحات از بایت 45 تا 204 این فایل موسیقی پیوسته قرار می­گیرد که متناظر با بایت­های 45 تا 204 صفحه صفرم از حافظه فلش است.

## رمزدار کردن فایل­های موسیقی ذخیره شده بر روی بورد

به منظور محدود­سازی دسترسی غیرمجاز به فایل­های موسیقی، این فایل­ها به صورت رمزنگاری شده تولید شده و در حافظه فلش ذخیره می­شوند. نحوه رمز کردن موسیقی­­ها نیز به این صورت است که ابتدا جای چهار بیت LSB و MSB عوض می­شود، سپس تمامی بیت­ها NOT می­شوند. در بورد FSB دقیقاً عکس این فرآیند تکرار   
می­شود، ابتدا جای چهار بیت LSB و MSB عوض می­شود، سپس تمامی بیت­ها NOT می­شوند.

برای پیاده­سازی الگوریتم فوق در نرم­افزار متلب، ابتدا تمامی داده­ها را در قالب 8 بیتی ذخیره کرده و سپس باقی­مانده آنها را به پیمانه 16 محاسبه می­کنیم. به این ترتیب 4 بیت LSB به دست خواهد آمد. در ادامه با کم کردن این حاصل از مقدار اصلی و تقسیم آن بر 16، چهار بیت MSB به مکان چهار بیت LSB منتقل می­شوند. در ادامه نیز با ضرب کردن باقیمانده تقسیم به 16 در 16، بیت­های LSB به MSB منتقل خواهند شد. NOT کردن نیز با استفاده از تابع bitcmp در متلب صورت می­پذیرد. پیاده­سازی تابع رمزنگاری موسیقی در تابع Music\_Ciphering() صورت پذیرفته است. همچنین رمزگشایی نیز در بورد FSB در تابع Music\_Deciphering() انجام خواهد شد (بخش ‏4-4-2-6-).

استفاده از روش ساده فوق در کنار استفاده از روش نمونه­گذاری باعث می­شود که در صورت عدم دسترسی به واسط نرم­افزاری متلب، امکان کپی­برداری از بورد به شدت پایین بیاید.



# بخش نرم­افزاری

## مقدمه

در این فصل در مورد کد نرم­افزاری مورد استفاده در بوردهای FSB300 و FSB310 بحث می­شود که این کد برای میکروکنترلر ATMEGA8 نوشته شده است. نوشتار کدها به صورت استانداردسازی شده در شرکت صانیک است که برای درک ساختار آن توصیه می­شود به گزارش «*استانداردسازی نوشتار کد*» رجوع شود.

با در نظر گرفتن بلوک دیاگرام ارائه شده در شکل ‏2‑2 برای نرم­افزار کد، فلوچارت نرم­افزار بورد FSB در بالاترین سطح به صورت شکل ‏4‑1 می­باشد. همانگونه که مشاهده می­شود کد نرم­افزار دارای دو بلوک اصلی تنظمیات اولیه و حلقه اصلی (عبارت while(1)) می­باشد. وظیفه بلوک تنظیمات اولیه که توسط تابع Initial\_Setup() در ماژول Main پیاده­سازی شده است، تنظیم رجیسترهای میکرو، استخراج شماره صفحات موسیقی از صفحه شماره صفر حافظه فلش (توسط تابع Page\_Extraction())، تنظیم متغیرهای کد به یک دسته مقادیر اولیه (توسط تابع Parameter\_Initial\_Setup()) و پس از آن سایر تنظیمات مرتبط مانند شماره آلبوم، سطح صدا و غیره می­باشد.

پس اجرای از انجام تنظمیات اولیه، نوبت به اجرای حلقه اصلی برنامه می­رسد. همانگونه که در فصل دوم اشاره شد هر تیک زمانی برنامه تقریباً برابر 16.5ms می­باشد و بنابراین هر تکرار حلقه اصلی برنامه نیز تقریباً برابر همین زمان خواهد شد. تنظیم رجیستر وضعیت (Status\_Register) و همینطور شمارش­گر زمان وضعیت فعلی (state\_timer) اولین قدم­هایی است که در هر تکرار حلقه اصلی انجام می­شود که مورد دوم توسط تابع timer\_update() انجام خواهد پذیرفت. همانگونه که در ادامه اشاره خواهد شد تنظیم state\_timer برمبنای وضعیت فعلی آسانسور و در تابع Change\_State() انجام خواهد گرفت. استخراج سیگنال ورودی قدمی است که در هربار تکرار حلقه اصلی برنامه انجام می­شود تا به نرم­افزار اجازه دهد خود را با وضعیت فعلی (که در تابع Change\_State() و برمبنای سیگنال­های ورودی و سایر پارامترهای مرتبط تنظیم می­شود) منطبق سازد. در نهایت نوبت به خواندن نصف صفحه داده از حافظه فلش (در صورت نیاز و بر مبنای وضعیت فعلی) خواهد رسید که این عمل توسط تابع Flash\_Data\_Read() انجام می­شود. توجه داریم همانگونه که در بخش 2-3 نیز اشاره شد پخش صوت در نرم­افزار به صورت همروند با سایر قسمت­های نرم­افزار و توسط یک وقفه صورت می­پذیرد. این وقفه به صورت جداگانه در ادامه این فصل (بخش ‏4-3-2-4-) تبیین خواهد شد.



شکل ‏4‑1: فلوچارت نرم­افزار بورد FSB در بالاترین سطح

## دیاگرام وضعیت در بورد FSB

همانگونه که در فصل اول نیز اشاره شد منظور از وضعیت در بورد، حالت­های کاری مختلف آسانسور (DOB، EMR، OVL و ...) که این حالت­ها با توجه به سیگنال­های کنترلی تعیین می­شوند. وضعیت­های ارائه شده در کد عموماً متناظر با وضعیت­های ارائه شده در فصل اول می­باشند، اما همانگونه که در ادامه اشاره خواهد شد بعضی از وضعیت­ها (مثل موسیقی به علاوه یک، سکوت به علاوه یک و ...) جزو وضعیت­های استاندارد تعریف شده در فصل اول نمی­باشند و برای تسهیل در پیاده­سازی نرم­افزار و یا خوانایی بیشتر کد تعریف شده­اند. وضعیت­های کلی نرم­افزار در جدول زیر تعریف شده­اند:

جدول ‏4‑1: وضعیت­های نرم­افزار بورد FSB و نماد آنها در کد

| وضعیت | نماد در کد |
| --- | --- |
| (پخش پیام) اورژانسی | EMR |
| مد تست | TEST\_MODE |
| مد یادگیری | LEARN\_MODE |
| (پخش پیام) اضافه وزن | OVL |
| (پخش پیام) باز بودن درب | DOB |
| پخش موسیقی (قبل از رسیدن به توقف) | MUS |
| پخش موسیقی اورژانسی پس از رسیدن به توقف و قبل از آغاز دیبانس سیگنال شمارنده | MUS\_1 |
| پخش موسیقی آرام بخش پس از وضعیت اورژانسی | MUS\_2 |
| پخش تمامی موسیقی یک آلبوم در مد تغییر آلبوم | MUS\_3 |
| سکوت | SILENCE |
| سکوت پس از پخش پیام اورژانسی | SILENCE\_1 |
| سکوت و بررسی مجدد سیگنال­ها در مد تست | SILENCE\_2 |
| پخش دینگ | DING |
| پخش دینگ دانگ | DING\_DONG |
| اعلان وضعیت توقف | DECLARE\_STOP\_STATUS |
| دیبانس سیگنال شمارنده پس از قطع شدن سیگنال TRG در حین پخش موسیقی | DEBOUNCE\_NUM\_SIGNAL |
| وضعیت تغییر آلبوم | ALBUM\_CHANGE |

مفهوم هر یک از وضعیت­ها در ادامه و در زیربخش­های مربوط خواهد آمد؛ ضمن اینکه گذار وضعیت­ها و تنظیم وضعیت­ جدید طبق تعاریف ارائه شده در گزارش «*استانداردسازی نوشتار کد*» در تابع Change\_State() صورت می­پذیرد. بنابراین دیاگرام وضعیت­هایی که در ادامه خواهند آمد در واقع دیاگرام وضعیت تابع Change\_State() می­باشند.

### وضعیت اورژانسی (EMR)

وضعیت اورژانسی پس از POR و در صورت وجود سیگنال اورژانسی اتفاق می­افتد. بنابراین این وضعیت   
می­تواند تنها یکبار در ابتدای کار مدار رخ دهد[[36]](#footnote-36). دیاگرام وضعیت حالت اورژانسی در شکل ‏4‑2 نشان داده شده است. همانگونه که مشاهده می­شود پس از رخ دادن وضعیت اورژانسی، ابتدا پیام اورژانسی در بورد پخش می­شود و سپس وضعیت بدون توجه به سیگنال­های ورودی برابر موسیقی به علاوه دو قرار داده می­شود.



شکل ‏4‑2: دیاگرام وضعیت حالت اورژانسی

### وضعیت موسیقی به علاوه دو (MUS\_2)

دیاگرام وضعیت موسیقی به علاوه دو در شکل ‏4‑3 آمده است. همانگونه که اشاره شد این وضعیت بلافاصله پس از وضعیت اورژانسی و بدون توجه به سیگنال­های ورودی رخ می­دهد. هدف از این وضعیت آن است که پس از پخش پیام اورژانسی، برای مدت زمانی معین موسیقی پخش شود. این موسیقی تم ملایم­تری دارد و طول آن نیز نسبتاً کوتاه است، ضمن اینکه پس از اتمام پخش، این موسیقی تکرار نمی­شود. توجه داریم پس از خاتمه پخش موسیقی، بدون توجه به سیگنال­های ورودی به وضعیت SILENCE\_1 وارد می­شویم.



شکل ‏4‑3: دیاگرام وضعیت حالت موسیقی به علاوه دو (MUS\_2)

### وضعیت سکوت به علاوه یک (SILENCE\_1)

دیاگرام وضعیت سکوت به علاوه یک در شکل ‏4‑4 آمده است همانگونه که اشاره شد وضعیت سکوت به علاوه یک بلافاصله پس از وضعیت موسیقی به علاوه دو رخ می­دهد. در این وضعیت تا زمانی که سیگنال اورژانسی برقرار باشد، بورد بدون توجه به سایر سیگنال­های ورودی در حالت سکوت باقی می­ماند؛ اما اگر سیگنال اورژانسی به هر دلیلی در بورد قطع شود، به وضعیت سکوت باز می­گردیم که در آن سایر سیگنال­های ورودی (غیر از اورژانسی) مورد بررسی قرار می­گیرند[[37]](#footnote-37). در مورد وضعیت سکوت در ادامه بحث خواهد شد.



شکل ‏4‑4: دیاگرام وضعیت حالت سکوت به علاوه یک (SILENCE\_1)

### وضعیت سکوت (SILENCE)

وضعیت سکوت نمایان­گر وضعیتی است که در آن هیچ صوتی پخش نمی­شود و سیگنال­های ورودی (در هر تیک زمانی) بررسی می­شوند. دیاگرام وضعیت این حالت در شکل ‏4‑5 نمایش داده شده است. بر طبق اولویت سیگنال­ها، اولین وضعیتی که بعد از سکوت می­توان به آن گذار کرد وضعیت اورژانسی می­باشد که این گذار تنها در صورتی رخ می­دهد که سیگنال ورودی اورژانسی شده باشد و بورد در POR باشد. همچنین اگر یکی از سیگنال­های تست و یا یادگیری (LEARN) با فشردن کلیدهای فشاری متناظر در POR تولید شوند، بورد به ترتیب وارد وضعیت­های تست و یادگیری خواهد شد که در ادامه در مورد آنها توضیح داده می­شود. توجه داریم ورود به این وضعیت­ها تنها در بورد FSB300 ممکن می­باشد. وضعیت بعدی که ممکن است رخ بدهد دریافت سیگنال تغییر آلبوم در وضعیت سکوت است. در این حالت بورد وارد وضعیت تغییر آلبوم می­شود که در مورد آن در ادامه توضیح داده خواهد شد. در حالت بعدی اگر سیگنال­ OVL دریافت شود، آنگاه شمارنده تاخیر در پخش پیام OVL افزایش پیدا می­کند تا وجود سیگنال OVL در چند تکرار حلقه اصلی بررسی شود، ضمن اینکه میان پخش­های مکرر پیام OVL تاخیر وجود داشته باشد. در نهایت اگر سیگنال OVL تا سرریز کردن شمارنده آن باقی بماند، آنگاه این شمارنده صفر شده و به وضعیت OVL می­رویم (که در ادامه در مورد آن توضیح داده می­شود). پس از سیگنال OVL، سیگنال دارای بیشترین اولویت سیگنال DOB است که فرآیندی که با دریافت این سیگنال انجام می­شود کاملاً مشابه سیگنال OVL است. توجه داریم علت اینکه پس از دریافت سیگنال OVL، شمارنده تاخیر پیام DOB صفر می­شود این مطلب می­باشد که ممکن است برای لحظاتی قبل از دریافت سیگنال OVL، سیگنال DOB دریافت شده باشد. اگر این عمل انجام نشود، آنگاه در صورت دریافت سیگنال DOB، پیام آن با تاخیر کمتری نسبت به حالت عادی پخش خواهد شد. وضعیت مشابهی نیز در حالت دریافت سیگنال DOB وجود دارد. همچنین بر طبق این دیاگرام اگر در وضعیت سکوت سیگنال TRG دریافت شود، باید پخش موسیقی آغاز شود که برای آن لازم است سیستم به وضعیت MUS می­رود. در نهایت اگر در وضعیت سکوت هیچ سیگنالی دریافت نشود، آنگاه وضعیت بعدی مجدداً برابر سکوت خواهد بود.



شکل ‏4‑5: دیاگرام وضعیت حالت سکوت

### وضعیت اضافه وزن (OVL)

وضعیت OVL وضعیتی است که پس از دریافت سیگنال OVL در [[38]](#footnote-38)OVL\_DELAY بار متوالی آشکارسازی سیگنال اضافه وزن (پس از دیبانس) رخ می­دهد[[39]](#footnote-39). اگر وضعیت سیستم OVL شود، آنگاه در حلقه اصلی برنامه، پیام OVL به صورت کامل پخش می­شود و پس از آن بورد به وضعیت سکوت می­رود تا سیگنال­های ورودی مجدداً بررسی می­شوند. دیاگرام وضعیت حالت اضافه وزن در شکل ‏4‑6 آورده شده است. بر طبق این دیاگرام اگر پس از پخش پیام OVL مجدداً سیگنال OVL دریافت شود، آنگاه شمارنده آن مجدداً افزایش پیدا می­کند و وارد وضعیت سکوت می­­شویم (توجه داریم در وضعیت قبل از وضعیت OVL که برابر SILENCE بوده است، شمارنده تاخیر پخش OVL برابر صفر قرار داده شده است و بنابراین این شمارنده در وضعیت فعلی از صفر آغاز می­شود). به طور مشابه اگر سیگنال DOB پس از پخش پیام OVL دریافت شود، آنگاه شمارنده تاخیر DOB افزایش پیدا کرده و وارد وضعیت سکوت می­شویم. اگر سیگنال TRG نیز بلافاصله پس از پخش پیام OVL دریافت شود به معنای حرکت بلادرنگ آسانسور پس از پخش پیام است و در این حالت پخش موسیقی آغاز می­شود.



شکل ‏4‑6: دیاگرام وضعیت حالت اضافه وزن

### وضعیت باز بودن درب (DOB)

وضعیت DOB وضعیتی است که پس از دریافت سیگنال DOB در DOB\_DELAY­ بار متوالی آشکارسازی سیگنال بازبودن درب (پس از دیبانس) رخ می­دهد. اگر وضعیت سیستم DOB شود آنگاه در حلقه اصلی پیام DOB به صورت کامل پخش می­شود و پس از آن سیگنال­های ورودی و وضعیت سیستم مجدداً بررسی می­شوند. دیاگرام وضعیت حالت باز بودن درب در شکل ‏4‑7 آورده شده است. همانگونه که مشاهده می­شود دیاگرام وضعیت در این حالت کاملاً مشابه حالت OVL است و بنابراین از بررسی جزئیات آن صرف نظر می­شود.



شکل ‏4‑7: دیاگرام وضعیت حالت باز بودن درب

### وضعیت موسیقی (MUS)

وضعیت موسیقی وضعیتی است که در آن پخش موسیقی به اندازه چهار تیک زمانی برنامه انجام می­پذیرد[[40]](#footnote-40) و پس از آن مجدداً وضعیت سیگنال­های کنترلی بررسی می­شود. توجه داریم وضعیت موسیقی تنها در حالتی اتفاق می­افتد که سیگنال TRG دریافت شود. دیاگرام وضعیت حالت MUS در شکل ‏4‑8 آورده شده است. همانگونه که مشاهده می­شود در صورتی که سیگنال تغییر آلبوم در این وضعیت وارد بورد شود، بورد وارد وضعیت تغییر آلبوم می­شود که در مورد آن در ادامه صحبت خواهد شد. همچنین اگر سیگنال TRG پس از ورود به وضعیت موسیقی همچنان دریافت شود، آنگاه پخش موسیقی ادامه پیدا می­کند. اما اگر سیگنال TRG پس از ورود به این وضعیت قطع شود[[41]](#footnote-41)، ابتدا یک پرچم که نشان دهنده ورود به یک توقف جدید است (پرچم new\_stop\_arrival\_flag) در این وضعیت تنظیم می­شود و سپس بورد وارد وضعیت MUS\_1 می­شود. توجه داریم در حالت MUS ممکن است سیگنال DOB به دلایل مختلف (مثل قطع شدن سیگنال به دلیل قرار گرفتن افراد در برابر سنسور آسانسور) در کنار وجود سیگنال TRG برقرار شود. اما از آنجا که اولویت سیگنال TRG بالاتر از سیگنال DOB می­باشد، سیگنال کنترلی ورودی به عنوان سیگنال TRG تعبیر خواهد شد و در نتیجه امکان پخش پیام DOB وجود نخواهد داشت[[42]](#footnote-42). این امر در مورد سیگنال OVL نیز صادق است[[43]](#footnote-43).



شکل ‏4‑8: دیاگرام وضعیت حالت موسیقی (MUS)

### وضعیت موسیقی به علاوه یک (MUS\_1)

هدف از وضعیت موسیقی به علاوه یک:

1. ایجاد یک پنجره زمانی بعد از قطع شدن سیگنال TRG برای تولید سیگنال شمارنده جدید توسط آسانسور است (که ممکن است به دلیل مکانیکی بودن تولید آن زمان­بر باشد). در این وضعیت همچنان موسیقی در بورد پخش می­شود.
2. پخش موسیقی پس از پخش دینگ/ دینگ دانگ و اعلان وضعیت توقف است.

دیاگرام وضعیت MUS\_1 در شکل ‏4‑9 نمایش داده شده است. در این وضعیت اگر پرچم رسیدن به توقف جدید در وضعیت MUS (به یک) تنظیم شده باشد، آنگاه در بورد به اندازه MUSIC\_BEFORE\_STOP برابر تیک زمانی برنامه موسیقی پخش می­شود. در انتهای این زمان بورد اصلی آماده تولید سیگنال شمارنده متناظر با توقف فعلی شده است و بنابراین وضعیت بعدی برابر DEBOUNCE\_NUM\_SIGNAL خواهد شد. در صورتی که این پرچم تنظیم نشده باشد، به این مفهوم است که وضعیت توقف اعلان شده است و بنابراین پخش موسیقی به اندازه MUSIC\_AFTER\_STOP برابر تیک زمانی برنامه ادامه پیدا می­کند و پس از آن وارد وضعیت سکوت می­شویم و بورد مجدداً حضور سیگنال­های مختلف را بررسی می­کند. ثابت MUSIC\_AFTER\_STOP باید بگونه­ای تنظیم شود که تمامی زمان توقف و خروج مسافران از آسانسور را پوشش دهد.



شکل ‏4‑9: دیاگرام وضعیت حالت موسیقی به علاوه یک (MUS\_1)

### وضعیت دیبانس سیگنال­های شمارنده (DEBOUNCE\_NUM\_SIGNAL)

هدف از وضعیت فوق، ایجاد یک پنجره زمانی برای بررسی سیگنال­های شمارنده در بورد FSB300 است. دیاگرام وضعیت این حالت در شکل ‏4‑10 نمایش داده شده است. همانگونه که مشاهده می­شود در این حالت پخش موسیقی به اندازه دو برابر تعداد دفعاتی که مطلوب است سیگنال شمارنده دیبانس شود منهای یک ادامه پیدا   
می­کند[[44]](#footnote-44). به این ترتیب سیگنال شمارنده به اندازه کافی دیبانس خواهد شد و در نتیجه امکان تشخیص وضعیت توقف فعلی فراهم می­شود و در عین حال شماره توقف فعلی با قبلی مقایسه خواهد شد. در صورتی که شماره توقف افزایش پیدا کرده باشد، بورد وارد وضعیت دینگ خواهد شد و اگر کاهش پیدا کند نیز وارد وضعیت دینگ دانگ خواهد شد. در حالت استثنایی اگر سیگنال شمارنده نامفهوم باشد (طبق پیاده­سازی توابع آشکارسازی سیگنال شمارنده در بخش­های ‏4-3-3-4- و ‏4-3-3-2-) شماره توقف فعلی برابر قبلی در نظر گرفته می­شود و به همین دلیل کد بلافاصله به وضعیت MUS\_1 می­رود. بنابراین در این حالت نه صدای دینگ و دینگ دانگ و نه وضعیت توقف پخش خواهد شد و تنها پخش موسیقی ادامه خواهد یافت. توجه داریم در بدترین حالت ممکن است به دلیل اینکه افزایش و یا کاهش شماره توقف در بار بعدی نیز درست تشخیص داده نشود (به دلیل اینکه شماره توقف فعلی به اشتباه برابر شماره توقف قبلی قرار داده شده است)، یکبار هم به جای دینگ، دینگ دانگ پخش شود (یا برعکس). بنابراین بیشینه در دو حالت وضعیت دینگ و یا دینگ دانگ درست تشخیص داده نمی­شود، اما پس از آن بورد وضعیت عادی باز می­گردد.

توجه داریم در بورد FSB310 دیبانس سیگنال معنایی ندارد، زیرا سیگنال­ها از طریق پروتکل I2C به بورد داده می­شوند. اما ورود به وضعیت DEBOUNCE\_NUM\_SIGNAL در بورد FSB310 مشکلی ایجاد نمی­کند. بورد در این حالت همچنان سیگنال شمارنده را دریافت می­کند، اما بروزرسانی سیگنال شمارنده در بورد FSB310 تنها زمانی صورت می­پذیرد که شماره توقف تغییر کرده باشد (به بخش ‏4-3-3-4- رجوع شود). به همین دلیل بروزرسانی سیگنال شمارنده در بورد FSB310 نیز در وضعیت DEBOUNCE\_NUM\_SIGNAL صورت می­پذیرد.



شکل ‏4‑10: دیاگرام وضعیت حالت دیبانس سیگنال شمارنده (DEBOUNCE\_NUM\_SIGNAL)

### وضعیت­های DING و DING DONG

این دو وضعیت پس از قطع سیگنال­های TRG و دیبانس سیگنال­های شمارنده (پس از وضعیت DEBOUNCE\_NUM\_SIGNAL) رخ می­دهند. دیاگرام وضعیت این حالت در شکل ‏4‑11 نمایش داده شده است. تفاوت دو وضعیت DING و DING DONG در این مطلب است که اگر شماره توقف افزایش پیدا کرده باشد قبل از پخش وضعیت توقف، دینگ پخش می­شود، ولی اگر شماره توقف کاهش پیدا کرده باشد صدای دینگ دانگ پخش خواهد شد. در نهایت توجه داریم پس از اتمام وضعیت فوق مستقیماً به وضعیت اعلان وضعیت توقف (DECLARE\_STOP\_STATUS) می­رویم.



شکل ‏4‑11: دیاگرام وضعیت حالت­های دینگ و دینگ­دانگ (DING & DING DONG)

### وضعیت اعلان وضعیت توقف (DECLARE STOP STATUS)

وضعیت DECLARE STOP STATUS بلافاصله پس از وضعیت DING و یا DING DONG رخ می­دهد و در آن وضعیت توقف فعلی با استفاده از سیگنال­ شمارنده پخش می­شود. دیاگرام وضعیت این حالت در شکل ‏4‑12 نمایش داده شده است. همانگونه که مشاهده می­شود پس از این وضعیت بدون توجه به ورودی­ها وارد وضعیت موسیقی به علاوه یک می­شویم.



شکل ‏4‑12: دیاگرام وضعیت حالت اعلان وضعیت توقف (DECLARE STOP STATUS)

### وضعیت­های CHANGE ALBUM و موسیقی به علاوه سه (MUS\_3)

وضعیت تغییر آلبوم پس از دریافت سیگنال تغییر آلبوم رخ می­دهد و وضعیت موسیقی به علاوه سه نیز به صورت بلادرنگ پس از وضعیت تغییر آلبوم رخ خواهد داد. دیاگرام وضعیت این حالت­ها در شکل ‏4‑13 نمایش داده شده است. هدف از این دو وضعیت، اجازه دادن به کاربر برای گوش دادن به آلبوم­های موسیقی موجود در بورد و انتخاب آنها است. همانگونه که مشاهده می­شود در صورت دریافت سیگنال تغییر آلبوم، ابتدا شماره آلبوم افزایش پیدا می­کند و در صورتی که این شماره به سه رسیده باشد (تعداد آلبوم­های موجود)، مجدداً به یک باز می­گردد. سپس متغیر نشان­دهنده صفحه فعلی موسیقی به اولین صفحه از آلبوم مورد نظر تنظیم می­شود. پس از انجام این دو عمل، بورد وارد وضعیت موسیقی به علاوه سه می­شود. در وضعیت موسیقی به علاوه سه، پخش یک آلبوم از ابتدای آن آغاز می­شود و تا زمانی که سیگنال تغییر آلبوم دریافت نشود پخش ادامه پیدا می­کند. در صورتی که قبل از اتمام پخش آلبوم سیگنال تغییر آلبوم دریافت شود، پخش متوقف شده و آلبوم جدید از ابتدا پخش می­شود. اما در صورتی که سیگنال دریافت نشود، پخش آلبوم تا انتها ادامه پیدا می­کند و پس از اتمام آن (با توجه به اینکه شماره آلبوم تغییر نکرده است)، آلبوم فعلی به عنوان آلبوم اصلی برای پخش انتخاب خواهد شد و بورد وارد وضعیت سکوت می­شود. با توجه به تعاریف فوق واضح است که هرگونه تغییر در سایر سیگنال­های کنترلی و شمارنده ورودی تاثیری بر   
وضعیت­های فوق نخواهد داشت. به عبارت دیگر به دلیل اولویت سیگنال تغییر آلبوم، سایر عملیات­ها در بورد مسکوت خواهند ماند. ضمن اینکه اگر بورد در این حالت خاموش/روشن شود، شماره آلبوم در بورد ذخیره شده و از چرخه کاری بعدی این آلبوم پخش خواهد شد.

(الف) دیاگرام وضعیت حالت تغییر آلبوم (ALBUM CHANGE) (ب) دیاگرام وضعیت حالت موسیقی به علاوه سه (MUS\_3)

شکل ‏4‑13: دیاگرام وضعیت حالت تغییر آلبوم (CHANGE ALBUM) و موسیقی به علاوه سه (MUS\_3)

### وضعیت تست (TEST MODE) و سکوت به علاوه دو (SILENCE\_2)

همانگونه که در فصل دوم اشاره شد، برای بررسی صحت اتصال تمامی پین­های ورودی/ خروجی، کلید فشاری و در نهایت صحت پروگرام میکروکنترلر و حافظه فلش در بورد FSB300 از مد تست استفاده می­شود. دیاگرام وضعیت این مد در شکل ‏4‑14 نشان داده شده است. همانگونه که اشاره شد مد تست تنها در POR اتفاق می­افتد، بنابراین با فشردن کلید فشاری افزایش صدا (کلید تست) بورد وارد وضعیت سکوت به علاوه دو می­شود. در این وضعیت ابتدا بررسی می­شود که آیا هیچ­ یک از پین­های ورودی به صورت استاندارد اشاره شده در جدول ‏2‑10 تحریک شده است یا خیر. سپس سه حالت ممکن است اتفاق بیفتد: اگر کلید تست فشرده شده باشد و یک پین تحریک شده باشد و در عین حال سیگنال­ها به قدر کافی دیبانس شده­اند، بورد وارد وضعیت TEST MODE   
می­شود. در این وضعیت بسته به پین تحریک شده در بورد، صدای یک طبقه پخش می­شود. در غیر این صورت اگر تنها کلید تست فشرده شده باشد ولی هیچ پینی تحریک نشده است یا دیبانس تکمیل نشده است، بورد در وضعیت سکوت به علاوه دو می­ماند (به این مفهوم که تست به پایان نرسیده است)، اما اگر کلید تست رها شود بورد از وضعیت تست خارج می­شود.



شکل ‏4‑14: دیاگرام وضعیت مد تست (TEST MODE) و سکوت به علاوه دو (SILENCE\_2)

### وضعیت یادگیری (LEARN\_MODE)

همانگونه که در فصل دوم اشاره شد از مد یادگیری برای اصلاح منطق فعال­سازی سیگنال­های DOB و TRG در نرم­افزار بورد استفاده می­شود. دیاگرام وضعیت این حالت در شکل ‏4‑15 نشان داده شده است. طبق تعریف ارائه شده در وضعیت سکوت، مد یادگیری تنها در POR و با فشردن کلید فشاری کاهش صدا (کلید یادگیری) ایجاد   
می­شود. مدت زمان توقف در این وضعیت دو برابر تعداد دیبانس­های مورد نیاز منهای یک در این مد (ثابت LEARN\_DEBOUNCE\_NUMBER نشان دهنده تعداد دیبانس در این مد است) است. در طول این وضعیت و در هر تکرار حلقه اصلی، سیگنال کنترلی با استفاده از تابع Parallel\_CTRL\_Signal\_Detection() (بخش ‏4-3-3-2-) خوانده و دیبانس می­شود. در نهایت پس از سپری شدن این زمان، تابع Learn\_Mode\_Assignments() فراخوانی می­شود که در آن با توجه به سطح منطقی پورت کنترلی، منطق سیگنال­­های DOB، OVL و TRG تخصیص   
می­یابد. جزئیات بیشتر در مورد نحوه عملیات فوق در بخش ‏4-4-4- آمده است.



شکل ‏4‑15: دیاگرام وضعیت مد یادگیری (LEARN MODE)

### مد زمان توقف برنامه در وضعیت­های مختلف

با توجه به مطالب ارائه شده در مورد چگونگی گذار وضعیت­های مختلف بورد، جدول ‏4‑2 مدت زمان توقف بورد در هر یک از وضعیت­ها را مشخص می­کند. توجه داریم منظور از طول یک پیام، تعداد صفحات متناظر با آن پیام در حافظه فلش است. از طرف دیگر در هر تیک زمانی برنامه یک نیم صفحه از حافظه فلش (و بعد از   
نمونه­گذاری یک صفحه کامل از آن) خوانده می­شود، بنابراین مدت زمان توقف در این وضعیت­ها برابر طول پیام ضرب­ در تیک زمانی برنامه ضرب در عامل نمونه­گذاری است. از طرف دیگر مدت زمان توقف در وضعیت MUS\_1 نیز همانگونه که اشاره شد بستگی به وضعیت قبلی دارد. بنابراین اگر آسانسور به تازگی به یک توقف رسیده است (وضعیت قبلی MUS است)، مدت زمان پخش موسیقی برابر MUSIC\_BEFORE\_STOP و اگر پس از رسیدن به توقف و آشکارسازی وضعیت توقف است (وضعیت قبلی DECLARE\_STOP\_STATUS یا DEBOUNCE\_NUM\_SIGNAL است) مدت زمان برابر MUSIC\_AFTER\_STOP انتخاب می­شود.

جدول ‏4‑2: مدت زمان توقف نرم­افزار در هر یک از وضعیت­ها

|  |  |
| --- | --- |
| وضعیت | مدت زمان توقف |
| EMR | طول پیام اورژانسی |
| TEST\_MODE | طول پیام متناظر با پین تحریک شده |
| LEARN\_MODE | دو برابر تعداد دفعات دیبانس در این مد منهای یک |
| OVL | طول پیام اضافه وزن |
| DOB | طول پیام باز بودن درب |
| MUS | چهار تیک زمانی برنامه |
| MUS\_1 | قبل از رسیدن به توقف: MUSIC\_BEFORE\_STOP  بعد از رسیدن به توقف: MUSIC\_AFTER\_STOP |
| MUS\_2 | طول موسیقی آرامش­بخش بعد از پیام اورژانسی |
| MUS\_3 | چهار تیک زمانی برنامه |
| SILENCE | صفر (عدم توقف) |
| SILENCE\_1 | صفر (عدم توقف) |
| SILENCE\_2 | صفر (عدم توقف) |
| DING | طول دینگ |
| DING\_DONG | طول دینگ دانگ |
| DECLARE\_STOP\_STATUS | طول پیام وضعیت توقف فعلی |
| DEBOUNCE\_NUM\_SIGNAL | دو برابر تعداد دفعات دیبانس در این مد منهای یک |
| ALBUM\_CHANGE | صفر (عدم توقف) |

در ادامه این فصل، چگونگی پیاده­سازی ماژول­های مختلف کد مورد بررسی قرار خواهد گرفت.

## پیاده­سازی نرم­افزار بورد (ماژول­های کد)

نرم­افزار بورد FSB شامل تعدادی ماژول است که هر ماژول به علاوه هدر آن کارکرد مشخصی را در بورد بر عهده دارد. فهرست تمامی ماژول­های مورد استفاده در بورد در جدول ‏4‑3 آمده است. توجه داریم هدر General.h نیز هدری است که بعضی از عمومی­ترین پارامترهای نرم­افزار در آن تعریف شده است که در ادامه در مورد پارامترهای تعریف شده در این هدر و سپس با شروع از ماژول Main، در مورد تمامی ماژول­های پیاده­سازی شده در برنامه و توابع آنها بحث خواهد شد.

جدول ‏4‑3: ماژول­های پیاده­سازی شده در نرم­افزار بورد FSB

|  |  |
| --- | --- |
| ماژول | کارکرد |
| Main | در برگیرنده تابع Main بورد FSB و سایر توابع کمکی مرتبط به آن |
| IO\_Module | تشخیص سیگنال­های ورودی |
| AT45DB321E\_Setup | تنظمیات اولیه فلش AT45DB321E |
| EEPROM\_Secure\_Read | خواندن داده به صورت مطمئن از EEPROM |
| Flash\_Data\_Read | خواندن داده از فلش AT45DB321E |
| Learn\_Mode | مد یادگیری در بورد FSB300 |
| Test\_Mode | مد تست در بورد FSB300 |
| I2C\_Protocol | توابع مرتبط با پروتکل I2C |
| I2C\_ACK\_NACK\_Generation |

### هدر General.h

فهرست ثوابت و متغیرهای سراسری (external) که در هدر فوق تعریف می­شوند در جدول ‏4‑9 آمده است. متفیرهای سراسری که در تمامی ماژول­های نرم­افزار استفاده می­شوند و در این هدر تعریف شده­اند به شرح جدول ‏4‑5 می­باشند.

جدول ‏4‑4: فهرست ثوابت تعریف شده در هدر General.h

|  | نام ثابت | توضیح | مقدار |
| --- | --- | --- | --- |
|  | Enum Total\_State | مدل یک شمارنده که دارای تمامی وضعیت­های بورد می­باشد | SILENCE، SILENCE\_1 و غیره |
| ثوابت مربوط به زمان­بندی­های بورد | START\_UP\_DELAY | زمان توقف برنامه پس از POR را نشان  می­دهد | 625 (تیک زمانی) |
| OVL\_DELAY | تاخیر زمانی میان پخش دو پیام OVL | 310 (تیک زمانی) |
| DOB\_DELAY | تاخیر زمانی میان پخش دو پیام DOB | 310 (تیک زمانی) |
| MUSIC\_BEFORE\_STOP | مدت زمان پخش موسیقی پس از رسیدن به یک توقف و قبل از آغاز دیبانس سیگنال شمارنده | 32 (تیک زمانی) |
| MUSIC\_AFTER\_STOP | مدت زمان پخش موسیقی پس از اعلان شماره توقف | 625 (تیک زمانی) |
| DOBAL\_TRGAL تا DOBAH\_TRGAH | سطح سیگنال­های DOB و OVL در حالت­های هر دو AL، هر دو AH و غیره | به جدول ‏2‑12 رجوع شود |
| تعاریف پورت­های I/O | CTRL\_PORT\_READ تا SPI\_PORT\_WRITE | تعریف رجیسترهای جهت و داده GPIOها برمبنای نوع سیگنال دریافتی از آن پورت |  |
| ثوابت مربوط به دیبانس سیگنال­ها | CTRL\_DEBOUNCE\_NUMBER | تعداد دفعاتی که لازم است سیگنال­های کنترلی قبل از آشکارسازی دیبانس شوند (بورد FSB300) | 5 |
| NUM\_DEBOUNCE\_NUMBER | تعداد دفعاتی که لازم است سیگنال­های شمارنده قبل از آشکارسازی دیبانس شوند (بورد FSB300) | 5 |
| VOL\_DEBOUNCE\_NUMBER | تعداد دفعاتی که لازم است سیگنال­های تغییر سطح صوت قبل از آشکارسازی دیبانس شوند | 5 |
| LEARN\_DEBOUNCE\_NUMBER | تعداد دفعاتی که سیگنال­های DOB و TRG در مد یادگیری دیبانس می­شوند | 32 |
| BOARD\_TYPE\_DEBOUNCE\_NUMBER | تعداد دفعاتی که سیگنال پین PB6 (که مشخص کننده نوع بورد است) دیبانس می­شود | 9 |
| آدرس مقادیر ذخیره شده در EEPROM | E2PROM\_VOL\_ADD تا E2PROM\_DOB\_INPUT\_ADD | محل ذخیره سطح صدای بورد تا شکل سیگنال DOB و TRG در EEPROM | بایت صفر تا بایت پنج |
| نوع بورد | FSB300 | بورد از نوع FSB300 است | صفر |
| FSB310 | بورد از نوع FSB310 است | یک |
| ثابت­های مرتبط با ارتباط I2C | I2C\_DATA\_LENGTH | طول داده ارسالی در I2C | 4 |
| I2C\_SLAVE\_ADDRESS | آدرس Slave در ارتباط I2C  (آدرسی که پردازنده FSB با آن در ارتباط I2C شناخته می­شود) | 0x01 |
| I2C\_BITRATE | مقدار اختصاص یافته به رجیستر نرخ بیت ارتباط I2C | 0x00 |
| I2C\_FREQUENCY\_PRESCALAR | مقدار مقسم فرکانسی ارتباط I2C | 0x00 |
| شماره فایل آلبوم­ها | ALBUM1\_FILE\_NUMBER | شماره فایل آلبوم اصلی بورد در جدول ‏3‑2 | 5 |
| EXTRA\_ALBUM1\_FILE\_NUMBER | شماره فایل آلبوم اضافی یک بورد در جدول ‏3‑2 | 78 |
| EXTRA\_ALBUM2\_FILE\_NUMBER | شماره فایل آلبوم اضافی یک بورد در جدول ‏3‑2 | 79 |
| NUMBER\_OF\_ALBUMS | تعداد آلبوم­های تعریف شده در بورد | 3 |
| سایر ثوابت | MAX\_VOLUME\_LEVEL | بیشینه سطح صدای بورد (غیر قابل تغییر!) | 6 |
| UPSAMPLE\_FACTOR | نرخ نمونه­گذاری در هنگام پخش بورد | 2 |
| TOTAL\_MEMORY\_FILES | مجموع تعداد فایل­های تعریف شده در حافظه بورد | 150 |

جدول ‏4‑5: فهرست متغیرهای خارجی تعریف شده در هدر General.h

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| current\_state | وضعیت فعلی بورد (شامل SILENCE، SILENCE+1 و غیره) |
| buffer\_data | ماتریس داده­هایی که از حافظه فلش خوانده می­شود |

ذکر چند نکته در مورد ثوابت ضروری می­­باشد:

* **زمان­ها و تاخیرها:** این ثوابت نشان­گر مضربی از تیک زمانی برنامه می­باشند. بنابراین زمان میان پخش شدن دو پیام متوالی DOB برابر DOB\_DELAY ضرب در تیک زمانی برنامه می­باشد. ثابت MUSIC\_AFTER\_STOP نشان دهنده زمانی می­باشد که بورد پس از قطع سیگنال TRG منتظر می­ماند تا سیگنال­های شمارنده را بخواند (بخش ‏4-2-1-). ضمن اینکه MUSIC\_AFTER\_STOP نیز بگونه­ای انتخاب شده است تا پس از پخش وضعیت توقف مدت زمان خروج مسافران از آسانسور را پوشش دهد.
* **پورت­های ورودی/ خروجی:** برای پورت­های کنترلی، شمارنده و SPI، نوشتارهای READ، WRITE و DIRECTION به ترتیب متناظر با PIN، PORT و DDRB می­باشد. بنابراین خواندن از پورت کنترلی یعنی PINC با CTRL\_PORT\_READ جایگزین می­شود.
* **پارامترهای دیبانس:** نشان دهنده تعداد دفعات خواندن یک سیگنال برای دیبانس آن می­باشد. این پارامترها نیز زمان را به صورت مضربی از تیک زمانی معرفی می­کنند. به عنوان مثال CTRL\_DEBOUNCE\_NUMBER = 4 به این معناست که سیگنال کنترلی در چهار تیک زمانی برنامه باید برابر باشد تا در مورد مقدار آن تصمیم­گیری شود.
* **پارامترهای مرتبط با ارتباط I2C:** از آنجا که بورد FSB300 در حالت دریافت در مد Slave در ارتباط I2C فعالیت می­کند، رجیسترهای نرخ بیت­ و مقسم فرکانسی تاثیری در سرعت ارتباط ندارند و به همین دلیل مقدار آنها برابر صفر انتخاب می­شود. از طرف دیگر آدرس نسبت داده شده به بورد کاملاً اختیاری است، اما باید دقت شود که بورد CIB با همین آدرس بورد FSB را مورد خطاب واقع کند.
* **ثابت تعداد فایل­ها**: ثابت TOTAL\_MEMORY\_FILES نشان دهنده تعداد فایل­هایی است که بورد FSB می­تواند آدرس ابتدایی و انتهایی آنها را از ماتریس Page\_address (به توضیحات در مورد ماژول Main رجوع شود) خوانده و مدیریت کند. در نرم­افزار پیش فرض تنها 80 فایل از این 150 فایل مورد استفاده قرار گرفته و به صورت کامل در بورد تعریف شده­اند، اما در صورت افزایش این عدد بیش از 80 لازم است بسته به کارکرد فایل جدید، سایر ماژول­های بورد (مخصوصاً ماژول IO و Flash\_Data\_Read) دستخوش تغییر شوند.

ذکر موارد زیر در مورد متغیرهای خارجی ضروری می­باشد:

* **current\_state**: این متغیر از نوع Total\_State می­باشد که این نوع متغیر در نرم­افزار تعریف شده است. Total\_State یک enumeration است که شامل تمامی وضعیت­های بورد (یعنی EMR، MUS\_1، SILENCE و ...) می­باشد، بنابراین current\_state نیز یکی از مقادیر فوق را اختیار خواهد کرد.
* **buffer\_data:** این آرایه مقادیر خوانده شده از حافظه فلش AT45DB321E را (که در Flash\_Page\_Read() خوانده می­شود) در خود ذخیره می­کند تا در وقفه پخش موسیقی از آن استفاده شود. از آنجا که در هربار خواندن داده از حافظه فلش تنها یک نیم صفحه داده از آن خوانده می­شود، طول این آرایه برابر 264 و از نوع کاراکتر است.

### ماژول Main

این ماژول که با نام Main.cpp نام­گذاری شده است، شامل تابع Main() و کلیه توابع زیر مجموعه آن است. ماژول Main دارای یک هدر به همین نام می­باشد که تمامی متغیرها و تعاریفی که در ماژول Main مورد استفاده قرار می­گیرند در آن آمده است. برای بعضی از متغیرهای تعریف شده در هدر این ماژول، از هدر ماژول­های بعدی نیز استفاده شده است که توضیحات در مورد آنها در ادامه خواهد آمد. فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑6 آمده است. مهمترین ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑7 آمده­اند. در ادامه در مورد کارکرد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑6: فهرست توابع تعریف شده در ماژول Main(به غیر از تابع main())

|  |  |
| --- | --- |
| تابع | کارکرد |
| Initial\_Setup | تنظیمات اولیه رجیسترها و متغیرها |
| Parameter\_Initial\_Setup | تخصیص دادن یک مقدار اولیه به متغیرها درون تابع Initial\_Setup() |
| timer\_update | به روزرسانی زمان فعلی برنامه |
| Board\_Type\_Detection | آشکارسازی نوع بورد (FSB300 یا FSB310) |
| Change\_State | تابع تعیین وضعیت حلقه اصلی بر اساس وضعیت فعلی |
| Page\_Extraction | استخراج شماره صفحات هر پیام از اولین صفحه حافظه فلش |

جدول ‏4‑7: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول Main

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| متغیر | |
| enum Generic\_State | تعریف مدل وضعیت­های کلی حلقه اصلی (BUSY، READY و WAIT) |
| Status\_Register | وضعیت­های کلی حلقه اصلی تابع main (یعنی BUSY، READY و WAIT) |
| half\_page\_counter | شماره نیم صفحه­ای که خوانده شده است |
| OVL\_Delay\_Counter | شمارنده تاخیر میان پخش دو پیام OVL متوالی (به عبارت دیگر تعداد دفعات دریافت سیگنال OVL بعد از دیبانس آن) |
| DOB\_Delay\_Counter | شمارنده تاخیر میان پخش دو پیام DOB متوالی (به عبارت دیگر تعداد دفعات دریافت سیگنال DOB بعد از دیبانس آن) |
| buffer\_counter | بایت داده­ای از صفحه فعلی فلش که در وقفه پخش صوت در حال پخش شدن است |
| sub\_buffer\_counter | تعداد دفعاتی که بایت فعلی در هنگام پخش تکرار شده است (بیشینه برابر ضریب  زیرنمونه­برداری بورد خواهد شد) |
| start\_up\_flag | پرچم عبور از زمان بالا آمدن بورد یا POR (صفر به معنای عدم عبور) |
| new\_stop\_arrival\_flag | پرچم رسیدن به یک توقف جدید (یک به معنای رسیدن به یک توقف جدید است) |
| state\_timer | زمان فعلی در حلقه اصلی (مقادیر ان برای وضعیت­های گوناگون در جدول ‏4‑2 آمده است) |
| CTRL\_Signal\_Params | ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های کنترلی (به ماژول IO رجوع شود) |
| NUM\_Signal\_Params | ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های شمارنده (به ماژول IO رجوع شود) |
| VOL\_Signal\_Params | ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های سطح صدا (به ماژول IO رجوع شود) |
| I2C\_data | آرایه داده­های دریافتی در ارتباط I2C (اندازه 4 دارد) |
| I2C\_output\_flag | پرچم خروجی ارتباط I2C (اتمام I2C یا مشغول بودن آن) |
| page\_address | ماتریس شماره صفحه ابتدایی و انتهایی هر یک از پیام­ها/ موسیقی­های ذخیره شده در بورد |
| current\_album | آلبوم فعلی در حال پخش |
| stop\_track\_index | شماره سطری از ماتریس page\_address متناظر با وضعیت توقف فعلی |
| test\_input\_number | شماره پینی از پورد که در مد تست تحت آزمایش است |
| Board\_Type | نوع بورد (FSB300 یا FSB310) |

#### تابع main()

برای درک ساده­تر حلقه اصلی برنامه، فلوچارت تابع main() در ماژول Main.cpp مجدداً در شکل ‏4‑16 تکرار شده است. قبل از توضیح حلقه اصلی، لازم است چگونگی انتخاب پارامتر state\_timer شرح داده شود. به جهت اینکه که تمامی پیام­/ موسیقی­ها بدون نگرانی از تغییر یافتن وضعیت بورد یا سیگنال­های ورودی تا انتها پخش شوند و همینطور از آنجا که در هر تکرار حلقه اصلی نیمی از یک صفحه حافظه فلش خوانده می­شود، مقدار state\_timer برابر تعداد صفحاتی اشغال شده از حافظه برای آن پیام / موسیقی مشخص ضرب ­در UPSAMPLE\_FACTOR قرار داده می­شود (به جدول ‏4‑2 رجوع شود). با در نظر گرفتن انتخاب فوق، بسته به مقدار فعلی state\_timer، حلقه اصلی می­تواند دو عمل اجرای وضعیت فعلی یا تعیین وضعیت جدید را انجام دهد. در حالت اجرای وضعیت فعلی (state\_timer)، تابع شروع به خواندن نیم صفحه داده از فلش توسط تابع Flash\_Data\_Read() می­نماید (به بخش ‏4-4-2- رجوع شو د)؛ پس از این مرحله، خواندن سیگنال از پورت­های ورودی برای تعیین وضعیت سیگنال­های ورودی در بورد FSB300 انجام می­شود که این عملیات توسط توابع Parallel\_CTRL\_Signal\_Detection() و Parallel\_NUM\_Signal\_Detection() انجام می­شود. به این ترتیب فاصله بین هر دو خواندن متوالی سیگنال­های کنترلی و شمارنده در بورد FSB300 برابر یک تیک زمانی است. دو عمل فوق تا زمانی که state\_timer به مقدار صفر نرسیده باشند تکرار می­شوند. در صورتی که state\_timer برابر صفر باشد، وارد مرحله تعیین وضعیت بعدی بورد و همچنین تعیین state\_timer می­شویم. برای تعیین وضعیت بعدی بورد، مقدار current\_state توسط تابع Change\_State() به روز رسانی می­شود. سپس متغیر state\_timer نیز بر اساس وضعیت و در همین تابع به مقدار مطلوب تنظیم می­شود.

توجه داریم آشکارسازی سیگنال در حلقه سنکرون تابع تنها در صورتی انجام می­شود که بورد از نوع FSB300 باشد. همانگونه که در فلوچارت نیز مشاهده می­شود (و در گزارش «*پروتکل ارتباطی I2C*» نیز اشاره شده است) آشکارسازی سیگنال I2C در بورد FSB310 در حلقه آسنکرون تابع صورت خواهد پذیرفت. با توجه به نوع پیاده­سازی پروتکل I2C در حلقه آسنکرون مشاهده می­شود که بروزرسانی سیگنال­ها در بورد FSB310 تنها پس از خاتمه ارتباط I2C (یعنی ارسال هر چهار فریم ارتباط) صورت خواهد پذیرفت. با پایان این ارتباط، چهار بایت دریافتی به توابع Serial\_NUM\_Signal\_Detection() و Serial\_CTRL\_Signal\_Detection() (بخش ‏4-3-3-) داده می­شوند تا سیگنال کنترلی و شمارنده از آنها استخراج شود. به این ترتیب سرعت بروزرسانی سیگنال­ها در بورد FSB310 به سرعت ارسال داده در ارتباط I2C مرتبط است.



شکل ‏4‑16: فلوچارت تابع main()

قبل از توضیح زیر توابع مورد استفاده در حلقه اصلی، لازم است موارد زیر یادآوری شوند:

* تابع Flash\_Data\_Read() در هر تکرار حلقه اصلی فراخوانده می­شود. توجه داریم تابع بگونه­ای تعریف شده است که تنها در وضعیت­هایی که نیاز به پخش موسیقی و یا پیام است داده از فلش خوانده شود و در سایر وضعیت­ها (مثل وضعیت سکوت)، هیچ داده­ای از فلش خوانده نمی­شود. بنابراین فراخوانی این تابع در هر تکرار حلقه اصلی مشکلی ایجاد نخواهد کرد.
* مقداردهی اولیه به تقریباً تمامی متغیرهای خارجی (extern) در ابتدای ماژول Main صورت پذیرفته است تا هیچگونه تداخلی در مقداردهی اولیه به این متغیرها به علت همپوشانی با سایر ماژول­ها ایجاد نشود.

در ادامه در مورد زیر توابع مورد استفاده در تابع main بحث خواهد شد.

#### تابع تنظیمات اولیه (Initial\_Setup())

|  |  |
| --- | --- |
| **کارکرد:** | انجام تنظیمات پردازنده / استخراج پارامترهای مورد نیاز از EEPROM / مقداردهی اولیه به پارامترها |
| **ورودی­ها:** | فاقد ورودی |
| **خروجی­:** | فاقد خروجی |



شکل ‏4‑17: فلوچارت تابع Initial\_Setup()

**شرح تفصیلی:** تابع Initial\_Setup() وظیفه برقراری تنظیمات اولیه در تابع main() را برعهده دارد. فلوچارت این تابع در شکل ‏4‑17 آمده است که در ادامه در مورد هر یک بحث خواهد شد:

* **تشخیص نوع بورد:** در این عملیات که با استفاده از زیر تابع Board\_Type\_Detection() صورت می­پذیرد، سطح ولتاژ پایه PB6 نه بار متوالی خوانده شده و سپس بر اساس منطق اکثریت در مورد سطح آن تصمیم­گیری می­شود. در صورتی که این ولتاژ یک باشد، آنگاه بورد FSB310 و در غیر این صورت FSB300 است.
* **تنظیم رجیسترهای میکرو:**
* تایمر/کانتر یک: همانگونه که در فصل دوم اشاره شد از این تایمر/کانتر برای تولید تیک زمانی برنامه و همچنین تولید موج PWM در مد fast PWM استفاده می­شود. همانگونه که اشاره شد، حد بالای شمارش در این پیاده­سازی توسط رجیستر ICR1 به شمارنده داده می­شود که با توجه به بحثی که در مورد آن شد، مقدار این رجیستر برابر 0xFF انتخاب می­شود. سایر رجیسترهای این تایمر به صورت زیر تنظیم می­شوند:

|  |  |
| --- | --- |
| (‏4‑1) | TCCR1A = 0x82;  TCCR1B = 0x19;  ICR1 = 0xFF;  OCR1 = 0x00; |

توجه داریم رجیستر OCR1 مقدار مقایسه شمارنده را در خود نگه می­دارد که برای حصول اطمینان مقدار آن در اولین اجرا برابر صفر تنظیم می­شود (و در خود وقفه مقدار آن برابر عدد هشت بیتی متناظر با دامنه سیگنال قرار داده می­شود).

* **رجیسترهای (زمانی) ارتباط SPI:** دو رجیستر مرتبط با ارتباط SPI در این قسمت از کد تنظیم می­شوند که یکی SPCR و دیگری SPSR می­باشد. به جهت بیشینه بودن سرعت خواندن داده از حافظه AT45DB321E، فرکانس ساعت این ارتباط برابر بیشنه مقدار خود (یعنی نصف فرکانس ساعت میکروکنترلر) تنظیم می­شود، و همچنین با رعایت اینکه ارتباط با لبه بالارونده پالس آغاز شود، ارتباط با منطق یک پالس فعال شود، داده از MSB به LSB خوانده شود و اینکه میکرو در ارتباط Master باشد، داریم:

|  |  |
| --- | --- |
| (‏4‑2) | SPCR = 0x50;  SPSR = 0x01; |

* **WD Timer:** همانگونه که در گزارش «*استانداردسازی نوشتار کد*» اشاره شده است، هدف از استفاده از این تایمر در تمامی نرم­افزارها، ریست کردن نرم­افزار در صورت قرار گرفتن آن در یک وضعیت دارای خطا است. از آنجا که هر تیک زمانی برنامه برابر 5/16 میلی­ثانیه در نظر گرفته می­شود، زمان ریست شدن برنامه برابر 32 میلی­ثانیه در نظر گرفته می­شود که حاشیه کافی از 5/16 میلی­ثانیه دارد. اگر فرض کنیم Vcc = 5v، رجیستر این تایمر به صورت زیر تنظیم می­شود:

|  |  |
| --- | --- |
| (‏4‑3) | WDTCR = 0x19 |

که نتیجه مطلوب را می­دهد.

* **تنظیمات ورودی/ خروجی میکرو:**
  + **پورت­های C و D:** از این دو پورت­ به ترتیب برای خواندن سیگنال­های کنترلی و شمارنده در FSB300 استفاده می­شوند. به همین دلیل، این دو پورت با نام CTRL\_port و NUM\_port در   
    نرم­افزار تعریف شده­اند. در نتیجه رجیستر جهت پورت و همینطور خواندن از پورت به عنوان مثال برای پورت C به صورت CTRL\_port\_direction و CTRL\_port\_read تعریف می­شوند. از آنجا که هر دوی این پورت­ها در بورد FSB300 ورودی هستند داریم:

|  |  |
| --- | --- |
| (‏4‑4) | CTRL\_port\_direction = 0x00  NUM\_port\_direction = 0x00 |

در مورد بورد FSB310 اما به غیر از پین­های PC0 و PC1 که وظیفه دریافت سیگنال سطح صدا را بر عهده دارند، هیچکدام از این دو پورت وظیفه دریافت سیگنال را بر عهده ندارند و به همین دلیل تعریف می­کنیم:

|  |  |
| --- | --- |
| (‏4‑5) | CTRL\_port\_direction = 0xFC  NUM\_port\_direction = 0xFF |

این تخصیص پورت برای بوردهای FSB300 و FSB310 با استفاده از یک حلقه if صورت   
می­پذیرد.

* **پورت B:** همانگونه که می­دانیم چهارخط ارتباط SPI یعنی SS، MOSI، MISO و SCK به ترتیب برروی پایه­های PB2 تا PB5 قرار دارند. به همین دلیل، این پورت با نام SPI\_port در نرم­افزار تعریف شده است. توجه داریم از میان چهار پورت فوق، تنها پورت MISO در چهار پورت فوق ورودی می­باشد و سه پورت دیگر ورودی می­باشند. از طرف دیگر پورت PB1 نیز پورتی است که PWM تایمر/کانتر یک در آن تولید می­شود، بنابراین در مورد جهت پین­ها داریم:

|  |  |
| --- | --- |
| (‏4‑6) | SPI\_port\_direction = 0x2E |

* **استخراج صفحات ابتدایی و انتهایی پیام­ها/ موسیقی­ها از حافظه فلش:** این عمل توسط تابع Page\_Extraction() انجام می­شود که در زیر بخش ‏4-3-2-3- در مورد آن بحث خواهد شد.
* **تنظیم اولیه متغیرهای تابع Main:** این قسمت توسط زیر تابع Parameter\_Initial\_Setup()   
  پیاده­سازی می­شود و هدف از آن تعیین مقادیر اولیه برای پارامترهای مورد استفاده در ماژول Main.cpp می­باشد. مقادیر اولیه اختصاص یافته بگونه­ای انتخاب شده­اند که هیچگونه عملیات ناخواسته­ای در بورد (مانند پخش صدا، اعلان وضعیت توقف و غیره) صورت نپذیرد.
* **تعیین اندازه صفحات فلش و تنظیم آنها به 528:** همانگونه که در فصل دوم اشاره شد در صورتی که اندازه صفحات حافظه فلش برابر 528 بایت نباشد، آدرس­دهی بخش­های مختلف حافظه دچار مشکل خواهد شد و بنابراین لازم است اندازه صفحات به مقدار مطلوب تنظیم شود. برای تنظیم آدرس صفحات، ابتدا آدرسی که با ثابت EEPROM\_MEMORY\_PAGES\_ADD تعریف شده از حافظه EEPROM به صورت امن (بخش ‏4-4-) خوانده می­شود. این آدرس از حافظه دارای دو سطح یک و یا صفر است که سطح یک آن توسط ثابت PAGES\_MODIFIED\_528 تعریف می­شود. یک بودن این مقدار ذخیره شده نشان می­دهد که عملیات بررسی و تغییر دادن صفحات از قبل صورت پذیرفته است. اگر این مقدار صفر باشد، آنگاه در ابتدا تعداد حافظه­های موجود در بورد توسط تابع Count\_Memory\_Chips() (بخش ‏4-4-3-1-) صورت می­پذیرد. سپس برای تمامی چیپ­های حافظه، بررسی اندازه صفحات فلش و تغییر آنها (در صورت لزوم) با استفاده از توابع ماژول AT45DB321E\_Setup (بخش ‏4-4-3-) صورت می­پذیرد.
* **تنظیم سطح صدای دیجیتال و شماره آلبوم فعلی و سطح منطقی DOB و TRG:** قبل از اجرای حلقه اصلی برنامه لازم است سطح صدای دیجیتال و همینطور شماره آلبومی که قرار است در بورد پخش شود تعیین گردد. این مطلب به خصوص از آنجا اهمیت دارد که لازم است در صورت قطع و وصل شدن مدار، شماره آلبوم و همینطور سطح صدا به همان مقداری تنظیم شود که از قبل در بورد تنظیم شده است تا نیازی نباشد کاربر در هر بار قطع و وصل شدن بورد این دو پارامتر را تنظیم کند. به همین منظور دو آدرس با نام­های EEPROM\_VOL\_ADD و همینطور EEPROM\_ALBUM\_ADD تعیین شده­اند که به ترتیب نشان دهنده قسمتی از حافظه EEPROM هستند که سطح صدای فعلی و همینطور شماره آلبوم فعلی در آن ذخیره می­شود. توجه داریم مقادیر ذخیره شده در این دو متغیر در هر بار روشن شدن بورد از حافظه EEPROMخوانده   
  می­شوند و در صورتی که داده خوانده شده مطمئن نباشد و یا مقادیر آنها خارج از بازه مجاز مقادیر باشد، به مقادیر پیش فرض سطح صدای صفر و آلبوم یک تنظیم خواهند شد. یادآوری می­شود در هر نقطه­ای از برنامه که مقدار این دو پارامتر تغییر کند (یعنی VOL\_Signal\_Detection() برای سطح صدای دیجیتال و همینطور تابع Change\_State() برای شماره آلبوم)، مقدار پارامتر جدید دوباره در همین آدرس EEPROM ذخیره خواهند شد. در مورد شماره آلبوم نیز پس از خوانده شدن آن در بورد، متغیر شماره صفحه موسیقی به ابتدای آلبوم فعلی تنظیم می­شود تا در پخش موسیقی اشکالی ایجاد نشود. در آخرین مرحله نیز سطح منطقی سیگنال­های DOB و TRG که در EEPROM ذخیره شده است استخراج می­شود. مقداری که در EEPROM برای این منظور ذخیره شده است یکی از چهار ثابت DOBAL\_TRGAL، DOBAL\_TRGAH و ... است که در ماژول مد یادگیری (بخش ‏4-4-4-1-) تعریف شده­اند. با خواندن این داده­ها از EEPROM به صورت امن،   
  این مقدار به تابع DOBTRG\_Level\_Assignment()داده می­شود تا بر آن اساس سطح سیگنال تنظیم شود. حلقه if که در این نقطه تشکیل شده است بررسی می­کند که در صورتی که مقدار خوانده شده از EEPROM یکی از چهار سطح پیش فرض نباشد و یا داده خوانده شده قابل اعتماد نباشد، سطح سیگنال­ها به مقدار پیش فرض DOBAL\_TRGAL (که پرکاربرد ترین منطق مورد استفاده در بازار است) تنظیم شود.

#### تابع استخراج صفحات پیام­ها / موسیقی از فلش (Page\_Extraction())

|  |  |
| --- | --- |
| **کارکرد:** | استخراج صفحات ابتدایی و انتهایی متناظر با پیام­ها و موسیقی­ها و ذخیره آنها در ماتریس page\_address |
| **ورودی­ها:** | فاقد ورودی |
| **خروجی­:** | فاقد خروجی |

شرح تفصیلی: این تابع بایت­های 44 تا TOTAL\_MEMORY\_FILES صفحه صفرم فلش که شماره­های صفحه ابتدایی و انتهایی پیام­ها و موسیقی­ها در آن نگه­داری می­شوند را خوانده و سپس شماره صفحه ابتدایی را در ستون اول ماتریس page\_address و شماره صفحه انتهایی را در ستون دوم ذخیره می­کند. هر سطر این ماتریس متناظر با یکی از فایل­های تعریف شده در جدول ‏3‑2 است و فایل 00.wav در سطر اول، فایل 01.wav در سطر دوم و به همینگونه ذخیره می­شود. نحوه ارسال دستورات و خواندن این صفحه کاملا مشابه مجموعه توابع ماژول Flash\_Data\_Read() (بخش ‏4-4-2-1-) است. توجه داریم هر 4 بایت متناظر با یک پیام، دو بایت اول متناظر با صفحه ابتدایی و بایت­ اول این دو بایت MSB شماره صفحه است. به همین دلیل بایت MSB پس از خوانده شدن از حافظه در 256 ضرب می­شود و سپس با بایت LSB جمع می­شود تا شماره صفحه محاسبه شود (به بخش 4-4-5 رجوع شود).

#### تابع تغییر وضعیت (Change\_State())

|  |  |
| --- | --- |
| **کارکرد:** | استخراج وضعیت بعدی بورد بر اساس وضعیت فعلی و سایر ورودی­ها |
| **ورودی­ها:** | وضعیت فعلی (current\_state) |
| **خروجی­:** | وضعیت بعدی |

**شرح تفصیلی:** تابع Change\_State() مسئول تعیین وضعیت بعدی بورد بر اساس وضعیت فعلی و همینطور سیگنال­های ورودی است. گذارهای وضعیتی که در این تابع رخ می­دهد بر اساس نمودارهای وضعیت ارائه شده در بخش ‏4-2- است. پس از استخراج وضعیت بعدی، متغیر state\_timer یا زمانی که نرم­افزار در وضعیت بعدی سپری می­کند بر اساس مقادیر تعریف شده در جدول ‏4‑2 تنظیم می­شود. ضمن اینکه در مورد وضعیت­هایی که باید صوتی در آنها پخش بشود، اولین صفحه نیز در متغیر شمارنده صفحات ذخیره می­شود. ذکر دو نکته در ارتباط با تابع فوق ضروری است:

* در صورتی که یک وضعیت غیر مجاز رخ بدهد، وضعیت بعدی بورد برابر SILENCE تعریف می­شود و زمان توقف نیز برابر صفر انتخاب می­شود. این عمل به نرم­افزار اجازه می­دهد تا بدون ایجاد مشکل از یک وضعیت تعریف شده مجدداً آغاز به کار نماید.
* در انتهای بعضی وضعیت­ها (مشابه SILENCE+2 یا MUS+2)، از تابع Empty\_Flash\_Buffer() استفاده شده است که وظیفه آن خالی کردن آرایه داده­های خوانده شده از فلش می­باشد. در مورد علت استفاده از این تابع به بخش ‏4-3-2-2- رجوع شود.

#### تابع خالی کردن داده­های بافر (Empty\_Buffer\_Data())

|  |  |
| --- | --- |
| **کارکرد:** | خالی کردن بافر داده­های خوانده شده از فلش (آرایه buffer\_data) |
| **ورودی­ها:** | فاقد ورودی |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** قبل از توضیح در مورد کارکرد تابع فوق، لازم است علت پیاده­سازی آن در نرم­افزار شرح داده شود. توجه داریم هرگاه وضعیت برنامه از به عنوان مثال موسیقی به علاوه یک به سکوت تغییر می­کند، آخرین صفحه داده خوانده شده از حافظه فلش در داخل آرایه سراسری buffer\_data باقی می­ماند. از آنجا که پخش صوت نیز به صورت همروند با برنامه اصلی و بدون توجه به وضعیت آن ادامه پیدا می­کند، این آرایه مرتباً در خروجی بورد پخش خواهد شد که باعث ایجاد یک صدای نامناسب و نویز مانند می­شود. وظیفه تابع Empty\_Buffer\_Data()، خالی کردن این آرایه از داده­ها پس از اتمام خواندن داده می­باشد.

نکته­ای که در مورد فراخوانی تابع فوق در بورد جلب نظر می­کند، فراخوانی این تابع تنها پس از گذار از وضعیت­های موسیقی به علاوه یک، موسیقی به علاوه دو، موسیقی به علاوه سه و DEBOUNCE\_NUM\_SIGNAL است (یعنی وضعیت­هایی که در آنها موسیقی پخش می­شود)، و به عنوان مثال این تابع پس از گذار از وضعیت DOB به سکوت فراخوانی نشده است. در واقع به صورت تجربی مشاهده شد که اگر بافر داده پس از گذار از وضعیت­هایی مانند DOB و یا OVL خالی شود، آنگاه پیام با صدای ناهنجاری قطع خواهد شد که مطلوب نیست. البته توجه داریم به دلیل خالی نشدن بافر پس از پخش این پیام­ها اگر پیام­های فوق با صدای بسیار بلند پخش شوند، این صدای نابهنجار در انتهای پخش پیام شنیده خواهد شد، اما این صدا بسیار کم بوده و در نتیجه قابل تحمل است. به صورت کلی علت رخداد فوق در بورد و چگونگی حل آن با یک روش قابل قبول یک مسئله حل نشده در طراحی بورد است!

#### تابع بروزرسانی زمان timer\_update()

|  |  |
| --- | --- |
| **کارکرد:** | بروزرسانی زمان­سنج حلقه اصلی |
| **ورودی­ها:** | فاقد ورودی |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی**: هدف از تابع فوق، بروزرسانی مقدار متغیر state\_timer می­باشد. در صورتی که بلافاصله پس از POR باشیم، مقدار state\_timer در تابع Initial\_Setup() برابر START\_UP\_DELAY قرار داده می­شود تا به بورد اجازه دهد تنظیمات اولیه خود را انجام دهد و ضمناً وجود سیگنال اورژانسی را بررسی کند. در همین حال متغیر start\_up\_flag نیز برابر صفر قرار داده می­شود تا نشان دهد بورد در وضعیت POR می­باشد و هنوز زمان بالا آمدن بورد سپری نشده است. سپس متغیر state\_timer در هر مرحله فراخوانی تابع یک واحد کاهش پیدا می­کند. پس سپری شدن START\_UP\_DELAY، start\_up\_flag برابر یک قرار داده می­شود. از این نقطه به بعد، وظیفه تابع تنها کاهش متغیر state\_timer به اندازه یک واحد در هر بار فراخوانی است.

#### وقفه پخش صوت

همانگونه که در بخش 2-4-4 اشاره شد، پخش صوت در بورد به صورت همروند با سایر قسمت­ها و توسط تایمر/کانتر یک صورت می­پذیرد. همچنین اشاره شد که این تایمر/کانتر به مد fast PWM تنظیم می­شود و وقفه آن در زمان برابر شدن شمارنده تایمر/کانتر با مقدار مقایسه (OCR1) رخ می­دهد.

در پیاده­سازی این وقفه در نرم­افزار، متغیر buffer\_counter مشخص کننده شماره بیت پخش شده از آرایه buffer\_data و به عبارت دیگر تعداد رخ­دادهای وقفه این تایمر/کانتر می­باشد و متغیر sub\_buffer\_counter نیز نشان دهنده تکرار این بایت در خروجی (به دلیل نمونه­گذاری) است. در هر بار رخ­دادن وقفه، ابتدا شکل موج مربعی با زمان وظیفه متناظر با مقدار ذخیره شده در این شماره از آرایه buffer\_data تشکیل می­گردد و این عمل به اندازه UPSAMPLE\_FACTOR تکرار می­شود. در 510-امین فراخوانی این وقفه (که در آن buffer\_counter برابر 255 است)، تیک زمانی به پایان رسیده است که در آن لازم است مقدار Status\_Register با توجه به وضعیت فعلی حلقه اصلی تنظیم شود. ضمن اینکه هر 264 بار (معادل 528 رخداد تایمر) نیز لازم است مقدار این متغیر صفر شود تا خواندن آرایه buffer\_data از ابتدای آن آغاز شود.

همانگونه که در بخش ‏2-4-4 اشاره شد، یک روش ساده برای تغییر سطح صدای بورد نصف کردن مقدار زمان وظیفه (duty\_cycle) می­باشد که این عمل به معنای تقسیم تمامی مقادیر خوانده شده از buffer\_data بر دو می­باشد. ایجاد شیفت بیتی در مقدار رجیستر اجازه می­دهد که بتوان مقدار DC را را تا برابر مقدار واقعی کوچک نمود که برابر تعداد شیفت بیتی مجاز است. توجه داریم N به 6 محدود می­شود، زیرا افزایش آن بیش از این مقدار باعث به کلی از دست رفتن اطلاعات نمونه­ها می­شود (زیرا باعث می­شود هر نمونه سیگنال صوت تنها با یک بیت بیان شود). بنابراین همانگونه که در بخش ‏2-4-4- اشاره شد، علی­رغم سادگی در روش فوق تعداد سطوح صدای خروجی بسیار محدود است. محدودیت صدای خروجی به خصوص هنگامی اهمیت می­یابد که توجه داشته باشیم DC­های بسیار کوچک باعث ایجاد صدای خروجی بسیار کمی می­شوند، بنابراین مقدار DC/32 و DC/64 عملاً در بورد FSB بدون استفاده می­باشند.

### ماژول ورودی/ خروجی بورد

این ماژول که با نام IO\_Module.c نام­گذاری شده است، مجموعه توابع آشکارسازی سیگنال­های ورودی و خروجی در بورد FSB را بر عهده دارد. توابعی که با پیشوند Parallel در این ماژول نام­گذاری شده­اند، توابعی هستند که برای ارسال و دریافت سیگنال در بورد FSB300 استفاده می­شوند. همچنین توابعی که با پیشوند Serial نام­گذاری شده­اند، برای تشخیص سیگنال­ها در بورد FSB310 کاربرد دارند. تابع VOL\_Signal\_Detection اما در هر دو بورد کاربرد دارد و برای تشخیص سیگنال­های سطح صدا استفاده می­شود. فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑11 آمده است. مهمترین ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑9 آمده­اند. در ادامه در مورد کارکرد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑8: فهرست توابع تعریف شده در ماژول ورودی/ خروجی بورد FSB

|  |  |
| --- | --- |
| تابع | کارکرد |
| VOL\_Signal\_Detection | آشکارسازی سیگنال کلیدهای فشاری سطح صدا و تعیین سطح صدای بورد |
| Parallel\_CTRL\_Signal\_Detection | تشخیص سیگنال­های کنترلی (به غیر از سطح صوت) در بورد FSB300 |
| Parallel\_NUM\_Signal\_Detection | تشخیص سیگنال­های شمارنده در بورد FSB300 |
| Parallel\_StopStatus\_Finder | تشخیص وضعیت توقف بر اساس سیگنال شمارنده دیبانس شده |
| Series\_CTRL\_Signal\_Detection | تشخیص سیگنال کنترلی (به غیر از سطح صوت) در بورد FSB310 |
| Series\_NUM\_Signal\_Detection | تشخیص سیگنال شمارنده در بورد FSB310 |

جدول ‏4‑9: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول ورودی/ خروجی بورد FSB

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| EMR\_DontC\_bits تا VOL\_DontC\_bits | بیت­های بدون اهمیت در هنگام استخراج سیگنال­های کنترلی EMR تا VOL در FSB300 |
| VOL\_UP\_INPUT | شکل سیگنال افزایش سطح صدا |
| VOL\_DOWN\_INPUT | شکل سیگنال کاهش سطح صدا |
| ALBUM\_CHANGE\_INPUT | شکل سیگنال تغییر آلبوم |
| VOL\_UNPUSHED\_INPUT | شکل سیگنال در حالت فشرده نشدن کلیدهای فشاری |
| PARALLEL\_EMR\_INPUT | شکل سیگنال اورژانسی در بورد FSB300 |
| PARALLEL\_PARK\_09 تا PARALLEL\_FLOOR19 | شکل سیگنال­های شمارنده از پارکینگ نه تا طبقه نوزدهم با فرض منطق AL در بورد FSB300 |
| SERIAL\_NO\_CTRL\_INPUT تا SERIAL\_DOB\_INPUT | شکل سیگنال­های کنترلی در بورد FSB310 |
| SERIAL\_STOP001 تا SERIAL\_STOP122 | شماره توقف آسانسور در بورد FSB310 |
| SERIAL\_PARK\_09 تا SERIAL\_FLOOR19 | شکل سیگنال­های شمارنده از پارکینگ نه تا طبقه نوزدهم با فرض منطق AL در بورد FSB310 |
| متغیر | |
| enum CTRL\_Signal\_Type | شکل نسبت داده شده به سیگنال­های کنترلی (اورژانسی، TRG و ...) |
| enum VOLUME\_Signal\_Type | شکل نسبت داده شده به سیگنال­های سطح صدا (افزایش صدا و ...) |
| مدل ساختار سیگنال­های کنترلی (ControlSignal) | مدل ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های کنترلی (نوع سیگنال یا CTRL\_Signal، ورودی کنترلی فعلی و قبلی یا previous\_input و current\_input، شمارنده دیبانس یا debounce\_counter، سطح منطقی TRG، OVL و DOB در FSB300 یا PARALLEL\_TRG\_input، PARALLEL\_OVL\_input و PARALLEL\_DOB\_input) |
| مدل ساختار سیگنال­های شمارنده | مدل ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های شمارنده (نوع سیگنال یا NUM\_Signal، شمارنده دیبانس یا debounce\_counter، ورودی شمارنده فعلی و قبلی یا previous\_input و current\_input، توقف قبلی و فعلی آسانسور یا previous\_stop و current\_stop) |
| مدل ساختار سیگنال­های سطح صوت | مدل ساختاری شامل تمامی پارامترهای مرتبط با سیگنال­های سطح صدا (نوع سیگنال یا یا VOL\_Signal، شمارنده دیبانس یا debounce\_counter، ورودی فعلی و قبلی یا previous\_input و current\_input، سطح صدای فعلی بورد یا digit\_volume\_level، وضعیت قبلی و فعلی کلیدهای فشاری یا keys\_current\_state و keys\_previous\_state) |

#### تابع آشکارسازی و دیبانس سیگنال­های کنترلی موازی (به غیر از سطح صوت) (Parallel\_CTRL\_Signal\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | دیبانس و آشکارسازی سیگنال­های کنترلی در بورد FSB300 |
| **ورودی­ها:** | ساختار پارامترهای سیگنال کنترلی، پرچم اتمام زمان بالا آمدن بورد (start\_up\_flag) |
| **خروجی­:** | فاقد خروجی |

(الف) فلوچارت تابع (ب) منطق دیبانس برای یک سیگنال دلخواه

شکل ‏4‑18: فلوچارت دیبانس سیگنال­های کنترلی (به غیر از سطح صدا)

**شرح تفصیلی:** این تابع برای دیبانس و آشکارسازی سیگنال­های کنترلی استفاده می­شود. منظور از آشکارسازی، تشخیص نوع سیگنال با استفاده از یک سیگنال دیبانس شده است (که سیگنال دیبانس شده لزوماً نشان دهنده یک ورودی قابل قبول در بورد نمی­باشد).

**یادآوری**: از آنجا که تابع Parallel\_CTRL\_Signal\_Detection() در هر تکرار حلقه اصلی فراخوانده می­شود و تعداد دیبانس تعریف شده برای آن 5 بار می­باشد، ممکن است وجود یک سیگنال کنترلی مشخص در حین پخش یک پیام/موسیقی دیگر تعیین گردد. اما از آنجا که ممکن است state\_timer متناظر با وضعیت فعلی همچنان غیر صفر باشد، بورد لزوماً وارد وضعیت جدید متناظر با این سیگنال ورودی نخواهد شد (چون تابع Change\_State() فراخوانی نمی­شود).

فلوچارت چگونگی دیبانس این سیگنال­ها در شکل ‏4‑18 آمده است. از آنجا که قرار است وضعیت­ اورژانسی تنها پس از POR و در صورت وجود سیگنال اورژانسی رخ بدهد، دیبانس سیگنال اورژانسی و آشکارسازی آن در این زمان صورت می­پذیرد. به همین دلیل اولین حلقه if سپری شدن زمان بالا آمدن بورد را مورد بررسی قرار می­دهد. در مورد سایر سیگنال­های کنترلی اما عمل دیبانس پس از سپری شدن POR انجام می­شود؛ همانگونه که اشاره شد در هر بار فراخوانی تابع در POR، سیگنال کنترلی وارد حلقه دیبانس می­شود که در آن در صورتی که سیگنال قبلی و فعلی برابر باشند، شمارنده دیبانس افزایش پیدا می­کند. در نهایت در حلقه آشکارسازی در صورتی که دیبانس به تعداد کافی انجام شده باشد، آشکارسازی سیگنال کنترلی با استفاده از سیگنال دیبانس شده (که همان ورودی فعلی است) صورت می­پذیرد. توجه داریم آشکارسازی سیگنال اورژانسی تنها در POR و وضعیت SILENCE+1 مطلوب است و در سایر وضعیت­ها این سیگنا آشکارسازی نمی­شود (حتی اگر به عنوان ورودی پورت کنترلی دیبانس شده باشد). علت اینکه سیگنال اورژانسی در وضعیت SILENCE+1 همچنان دیبانس می­شود این است که در صورتی که سیگنال اورژانسی مجدداً قطع شود و آسانسور از وضعیت اضطراری خارج شود بتوان به کارکرد عادی بورد بازگشت.

یادآوری می­شود که آشکارسازی سیگنال کنترلی تنها در صورتی انجام می­شود که بورد در وضعیتی غیر از وضعیت یادگیری و یا تست داشته باشد. در وضعیت یادگیری لازم است شکل­ سیگنال­های ورودی بر مبنای مقادیر ورودی DOB و TRG تنظیم شوند که این کار در تابع Learn\_Mode\_Assignments() (توضیحات در ماژول یادگیری) انجام خواهد شد.

#### تابع آشکارسازی و دیبانس سیگنال­ شمارنده موازی (Parallel\_NUM\_Signal\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | دیبانس و آشکارسازی سیگنال­ شمارنده در بورد FSB300 |
| **ورودی­ها:** | ساختار پارامترهای سیگنال شمارنده |
| **خروجی­:** | شماره سطری از ماتریس page\_address متناظر با وضعیت توقف فعلی |

**شرح تفصیلی:** دیبانس و آشکارسازی سیگنال شمارنده در این تابع صورت می­پذیرد. منطق دیبانس سیگنال­ شمارنده کاملاً مشابه منطق سیگنال­های کنترلی می­باشد، با این تفاوت که دیبانس این سیگنال­ها تنها در وضعیت DEBOUNCE\_NUM\_SIGNAL انجام خواهد شد تا تغییر این سیگنال تنها پس از قطع شدن سیگنال TRG در بورد FSB300 بررسی شود؛ پس از استخراج سیگنال شمارنده دیبانس شده، این سیگنال به تابع Parallel\_StopStatus\_Finder() داده می­شود تا در آن توسط یک حلقه switch، وضعیت توقف (پارکینگ، لابی، و ...) و شماره توقف متناظر با آن استخراج شود. در ضمن شماره توقف توقف فعلی نیز در ساختار سیگنال شمارنده ذخیره می­شود تا از آن برای تعیین وضعیت DING یا DING DONG استفاده شود. در نهایت اینکه متغیر stop\_track\_index که نماینده شماره سطری از ماتریس page\_address است که نشان دهنده صوت متناظر با توقف فعلی است توسط این حلقه تخصیص می­یابد.

**یادآوری:** تابع در حلقه اصلی به مدت دو برابر منهای یک مقدار ثابت NUM\_DEBOUNCE\_NUMBER در وضعیت DEBOUNCE\_NUM\_SIGNAL باقی می­ماند. از طرف دیگر در پیاده­سازی این تابع، در صورتی که بورد در وضعیت DEBOUNCE\_NUM\_SIGNAL نباشد، پارامتر NUM\_debounce\_number که نشان دهنده تعداد دفعات دیبانس صحیح این سیگنال شمارنده است صفر خواهد شد. به این ترتیب تا زمانی که بورد وارد وضعیت DEBOUNCE\_NUM\_SIGNAL نشود سیگنال شمارنده دیبانس نخواهد شد. ضمن اینکه دیبانس سیگنال شمارنده تنها یکبار در این حالت اتفاق خواهد افتاد.

**توجه:** در صورتی که شماره توقف فعلی نامفهوم باشد، شماره توقف فعلی و قبلی برابر در نظر گرفته خواهد شد (حالت default حلقه switch فاقد تخصیص شماره توقف است) که در این وضعیت (همانگونه که در بخش ‏4-2-2- اشاره شد)، نه صدای دینگ و نه دینگ دانگ پخش می­شود و بورد بدون اعلان وضعیت توقف تنها موسیقی پخش خواهد کرد.

#### تابع آشکارسازی سیگنال کنترلی سریال (Serial\_CTRL\_Signal\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | آشکارسازی سیگنال­ کنترلی در بورد FSB310 |
| **ورودی­ها:** | ساختار پارامترهای سیگنال کنترلی، پرچم سپری شدن زمان بالا آمدن بورد (start\_up\_flag)، داده­های دریافتی در ارتباط I2C (I2C\_data\_frame[]) |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** سیگنال کنترلی که در اولین فریم ارتباط I2C به بورد FSB310 می­رسد در این تابع آشکارسازی می­شود. تابع به صورت کلی شامل یک حلقه switch است که در آن مقدار داده دریافتی در ارتباط I2C با مقادیر تعریف شده در جدول ‏2‑7 مقایسه می­شود و سیگنال کنترلی متناظر اعلان می­گردد. توجه داریم سیگنال اورژانسی مشابه بورد FSB300 تنها در POR و یا وضعیت SILENCE+1 آشکارسازی می­شود در سایر وضعیت­ها در صورت دریافت سیگنال اورژانسی، سیگنال کنترلی برابر NO\_SIGNAL در نظر گرفته می­شود. ضمن اینکه در صورتی که هیچ یک از مقادیر این جدول در این فریم ارسال نشده باشد، سیگنال کنترلی برابر حالت NO\_SIGNAL در نظر گرفته می­شود.

#### تابع آشکارسازی سیگنال شمارنده سریال (Serial\_NUM\_Signal\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | آشکارسازی سیگنال­ کنترلی در بورد FSB310 |
| **ورودی­ها:** | ساختار پارامترهای سیگنال شمارنده، داده­های دریافتی در ارتباط I2C (I2C\_data\_frame[]) |
| **خروجی­:** | شماره سطری از ماتریس page\_address متناظر با وضعیت توقف فعلی |

**شرح تفصیلی:** شماره توقف (stop) به همراه سیگنال­ شمارنده که در به ترتیب دومین، و سومین و چهارمین فریم ارتباط I2C دریافت می­شوند در این تابع آشکارسازی می­شوند. در اولین قدم در این تابع، آفست سیگنال شماره توقف (مقدار 0x80 که به تمامی شماره توقف­ها اضافه می­شود، به بخش ‏2-9-1- رجوع شود) از دومین فریم ارتباط I2C کم می­شود. در ادامه و در صورتی که شماره توقف فعلی بین 1 تا 122 باشد و با توقف قبلی برابر نباشد (که برابر بودن نشان دارای خطا بودن شماره توقف است) به عنوان شماره توقف فعلی ذخیره می­شود. حال متغیر I2C\_NUM\_bytes که یک متغیر 16 بیتی و نشان دهنده دو هفت بیتی متناظر با سون سگمنت­های سیگنال شمارنده است به این صورت تشکیل می­شود که هفت بیت سون سگمنت سمت چپ در بیت­های با اهمیت این متغیر قرار می­گیرند. حال با استفاده از یک حلقه switch مقدار I2C\_NUM\_bytes با مقادیر پیش فرض تعریف شده برای وضعیت توقف­ها (پارکینگ، لابی، همکف و ...) مقایسه می­شود و در صورت تشخیص یک مورد یکسان، شماره سطری از ماتریس page\_address متناظر با این وضعیت توقف در متغیر stop\_track\_index قرار داده می­شود. در آخرین مرحله نیز این متغیر به عنوان خروجی تابع بازگردانده می­شود. توجه داریم در حالت استثنایی اگر وضعیت توقف ارسالی معنادار نباشد، از آنجا که وضعیت توقف فعلی قابل پخش نیست، شماره توقف فعلی و قبلی با یکدیگر برابر در نظر گرفته می­شود تا بورد وارد وضعیت DING و یا DING\_DONG نشود.

#### تابع آشمارسازی سیگنال­ سطح صدا و تغییر آلبوم (VOL\_Signal\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | دیبانس و آشکارسازی سیگنال­ ورودی کلیدهای فشاری (سطح صوت و تغییر آلبوم) |
| **ورودی­ها:** | ساختار پارامترهای سیگنال سطح صدا |
| **خروجی­:** | فاقد خروجی |



شکل ‏4‑19: فلوچارت دیبانس سیگنال­ سطح صدا/ آلبوم در تابع InputSignalDetection()

**شرح تفصیلی**: از تابع فوق برای دیبانس و آشکارسازی سیگنال­های کنترلی ورودی از کلیدهای فشاری استفاده می­شود. فلوچارت چگونگی دیبانس این سیگنال­ها در شکل ‏4‑19 آمده است. همانگونه که در بخش   
‏2-8-2- اشاره شد، به دلیل اینکه این سیگنال­های کنترلی با استفاده از کلیدهای فشاری وارد بورد می­شوند، منطق دیبانس آنها متفاوت با سایر سیگنال­ها می­باشد. در واقع یک سیگنال کنترلی در این حالت تنها زمانی تشخیص داده می­شود که ابتدا کلیدها فشرده شده و سپس رها شوند. به این منظور پس از تشخیص نوع سیگنال ورودی از این کلیدها، اولین نسبت دادن یک وضعیت به کلیدها است. به منظور اینکه قدم­های توصیف شده در بخش ‏2-8-2- رعایت شود، نسبت دادن کلیدها با بررسی این مطلب آغاز می­شود که هر دو کلید در دیبانس قبلی فشرده شده­اند یا خیر. اگر پاسخ به این سوال منفی باشد، بدان معناست که تغییر آلبوم مد نظر کاربر نیست و در نتیجه وضعیت قبلی و فعلی کلید بروزرسانی می­شود. اما در صورتی که تشخیص داده شود که هر دو کلید فشرده شده­اند، تنها در صورتی اجازه بروزرسانی وضعیت کلیدها داده می­شود که هر دو کلید رها شوند. به این ترتیب پس از وارد شدن سیگنال تغییر آلبوم، تنها رها شدن کلیدها باعث بروزرسانی وضعیت می­شود.

پس از تعیین وضعیت کلیدها، نوبت به تعیین نوع سیگنال کنترلی (افزایش/ کاهش صدا و یا تغییر آلبوم)   
می­رسد. این حلقه تنها زمانی فعال می­شود که هر دو کلید رها شده باشند، به این ترتیب هیچ سیگنال کنترلی تا زمان رها شدن هر دو کلید در بورد اعمال نخواهد شد. توجه داریم تمامی عملیات تعیین وضعیت کلیدها و همینطور تعیین نوع سیگنال کنترلی تنها در صورتی انجام می­شود که یک سیگنال ورودی از کلید به قدر کافی دیبانس شده باشد، در غیر این صورت ممکن است قبل از تغییر وضعیت­ کلیدها به اشتباه تغییری در سیگنال کنترلی دریافتی از آنها اعلان شود.

## ماژول خواندن داده به صورت امن از EEPROM

این ماژول که با نام EEPROM\_Secure\_Read.c نام­گذاری شده است، مجموعه توابع خواندن داده به صورت امن از حافظه EEPROM را در خود جای می­دهد. خواندن داده داده از EEPROM در این ماژول به این صورت تعریف می­شود که اگر در دو بار پشت سرهم خواندن داده از EEPROM، محتوا برابر باشد، آنگاه داده خوانده شده به صورت صحیح در نظر گرفته می­شود. . فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑12 آمده است. مهمترین ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑13 آمده­اند. در ادامه در مورد کارکرد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑10: فهرست توابع تعریف شده در ماژول خواندن داده به صورت امن از EEPROM

|  |  |
| --- | --- |
| تابع | کارکرد |
| EEPROM\_Secure\_Read | تابع خواندن داده از EEPROM و بررسی امن بودن خواندن داده |
| charEEPROM\_read | خواندن یک کاراکتر از EEPROM |

جدول ‏4‑11: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول خواندن داده به صورت امن از EEPROM

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| CORRUPTED\_DATA | داده خوانده شده از EEPROM سالم نیست |
| TRUSTED\_DATA | داده خوانده شده از EEPROM سالم است |
| EEWE | بیت EEWE رجیستر کنترلی EEPROM |
| EERE | بیت EERE رجیستر کنترلی EEPROM |
| EEPROM\_REPEAT\_READ | تعداد دفعاتی که برای خواندن از EEPROM تلاش می­شود |
| متغیر | |
| مدل ساختار خواندن از EEPROM | مدل یک ساختار که داده خوانده شده از EEPROM و پرچم متناظر با آن (داده سالم یا دارای مشکل) را در خود دارد |

#### تابع خواندن داده از EEPROM (charEEPROM\_read())

|  |  |
| --- | --- |
| **کارکرد:** | خواندن یک بایت از EEPPROM از آدرس مورد نظر |
| **ورودی­ها:** | آدرس بایت مورد نظر در EEPROM (EEPROM\_Byte\_Address) |
| **خروجی­:** | بایت خوانده شده |

**شرح تفصیلی: از** این تابع برای خواندن داده از EEPROM از آدرس مورد نظر استفاده می­شود. نحوه خواندن داده به این صورت است که ابتدا با استفاده ازیک حلقه while بررسی می­شود که هیچ عملیات خواندن و نوشتن دیگری بر روی EEPROM در جریان نباشد. پس از آن آدرس بایت مورد نظر از EEPROM در متغیر EEAR قرار داده می­شود. سپس فرمان خواندن داده از EEPROM با تنظیم رجیستر کنترلی EEPROM (یعنی EECR) داده می­شود. در نهایت نیز مقدار داده خوانده شده از EEPROM که در رجیستر داده آن (EEDR) قرار دارد به عنوان خروجی این تابع بازگردانده می­شود.

#### تابع بررسی سالم بودن داده خوانده شده از EEPROM (EEPROM\_Secure\_Read())

|  |  |
| --- | --- |
| **کارکرد:** | بررسی سالم بودن داده خوانده شده از EEPROM |
| **ورودی­ها:** | آدرس بایت مورد نظر در EEPROM (EEPROM\_Byte\_Address) |
| **خروجی­:** | ساختار داده خوانده شده از EEPROM (شامل داده و پرچم سالم بودن آن) |

**شرح تفصیلی:** این تابع، تابع اصلی خواندن داده در این ماژول است. دو متغیر EEPROM\_first\_read و EEPROM\_second\_read، دو خواندن متوالی داده با استفاده از تابع charEEPROM\_read() را در خود نگه­داری می­کنند. در حلقه while این تابع، بررسی می­شود که آیا این دو خواندن متوالی با یکدیگر برابر هستند یا خیر و در صورت برابر نبودن، عمل خواندن داده مجدداً و تا تعدادی که با ثابت EEPROM\_REPEAT\_READ مشخص می­شود تکرار می­گردد. در نهایت در یک حلقه if بررسی می­شود که آیا تعداد تکرار از مقدار ثابت فوق بیشتر شده است یا خیر و در صورتی که این تعداد بیشتر باشد، پرچم خواندن از EEPROM که در خروجی تابع (و در غالب یک ساختار) بازگردانده می­شود برابر CURRPOTED\_DATA قرار داده می­شود؛ در غیر این صورت این پرچم برابر TRUSTED\_DATA خواهد شد. توجه داریم این تابع همیشه آخرین داده خوانده شده از EEPROM را به عنوان خروجی باز می­گرداند، اما تنها زمانی باید از این داده استفاده شود که پرچم تعیین کند داده امن بوده است.

### ماژول خواندن داده از فلش AT45DB321E

این ماژول که با نام Flash\_Data\_Read.c در کد نام­گذاری شده است، وظیفه ارسال فرمان­ به حافظه(های) AT45DB321E نصب شده بر روی بورد و خواندن داده از آنها از طریق ارتباط SPI را دارد. همانگونه که در ادامه توضیح داده خواهد شد، این ماژول با کمک پورت PB0، قابلیت برقراری ارتباط با دو حافظه فلش را دارد. فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑12 آمده است. مهمترین ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑13 آمده­اند. در ادامه در مورد کارکرد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑12: فهرست توابع تعریف شده در ماژول خواندن داده از فلش AT45DB321E

|  |  |
| --- | --- |
| تابع | کارکرد |
| Flash\_Data\_Read | آماده­سازی خواندن موسیقی / پیام از بخش­های متناظر در حافظه فلش |
| MUS\_Page\_Reset | بازگرداندن متغیر شماره صفحه فعلی موسیقی به ابتدای آلبوم فعلی |
| Flash\_Page\_Read | انتخاب حافظه فلش. ارسال فرمان خواندن داده به فلش.  خواندن داده­ها از فلش و ذخیره­ آنها در آرایه buffer\_data |
| SPI\_Interface | دسترسی و خواندن / نوشتن داده­ها از طریق ارتباط SPI |
| Memory\_Selection | ارسال سیگنال انتخاب چیپ در ارتباط SPI به یکی از حافظه­های موجود در بورد |

جدول ‏4‑13: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول خواندن داده از فلش AT45DB321E

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| ENABLE\_CHIP\_SELECT | ارسال سیگنال انتخاب چیپ بر روی پورت SS ارتباط SPI |
| ENABLE\_SECOND\_FLASH | فراخوانی فلش دوم نصب شده بر روی بورد (قطع ارتباط با فلش اول) با ارسال یک بر روی پایه PB0 |
| Main\_Mem\_Read | فرمان خواندن یک صفحه از فلش AT45DB321E به صورت پیوسته |
| FINAL\_PAGE | آخرین صفحه­ای که در فلش بر روی آن داده وجود دارد |
| متغیرهای خارجی | |
| EMR\_current\_page | شماره صفحه­ای از پیام اورژانسی که از فلش خوانده می­شود |
| MUS\_current\_page | شماره صفحه­ای از موسیقی که از فلش خوانده می­شود |
| OVL\_current\_page | شماره صفحه­ای از پیام اضافه­ وزن که از فلش خوانده می­شود |
| DING\_current\_page | شماره صفحه­ای از دینگ که از فلش خوانده می­شود |
| DING\_DONG\_current\_page | شماره صفحه­ای از دینگ دانگ که از فلش خوانده می­شود |
| STOP\_STATUS\_current\_page | شماره صفحه­ای از پیام وضعیت توقف که از فلش خوانده می­شود |

#### تابع خواندن داده از فلش (Flash\_Data\_Read())

|  |  |
| --- | --- |
| **کارکرد:** | فراخوانی تابع Flash\_Page\_Read براساس وضعیت فعلی بورد برای خواندن نیمی از صفحه حافظه فلش. |
| **ورودی­ها:** | شمارنده نیم صفحه فعلی (half\_page\_counter)، اشاره­گر به متغیر زمان فعلی (\*state\_timer)، شماره آلبوم فعلی (current\_album)،  اشاره­گر به محل آلبوم اصلی در ماتریس page\_address، اشاره­گر به محل آلبوم اضافی یک در ماتریس page\_address، اشاره­گر به محل آلبوم اضافی دو در ماتریس page\_address |
| **خروجی­:** | half\_page\_counter به روز شده |

**شرح تفصیلی**: هدف از این تابع، آماده­سازی پخش داده از حافظه فلش با توجه به وضعیت تعیین شده در حلقه اصلی ماژول Main است. در صورتی که وضعیت فعلی نیازمند پخش موسیقی/ پیام باشد، تابع Flash\_Page\_Read() فراخوانی می­شود که هدف از آن خواندن داده­های نیمی از صفحه فعلی حافظه می­باشد (به بخش بعدی رجوع شود). توجه داریم علاوه بر موارد بالا، تابع half\_page\_counter را به عنوان ورودی دریافت و مقدار به روز شده آن را به عنوان خروجی باز می­گرداند. به این ترتیب در تابع main مشخص می­شود که کدام نیم صفحه (نیم صفحه اول یا دوم) از صفحه فعلی داده خوانده شده است. در نهایت اینکه شماره صفحه فعلی که از حافظه پخش شده است (که با توجه به وضعیت فعلی در متغیرهای MUS\_current\_page، DOB\_current\_page و ... ذخیره خواهد شد) پس از اتمام خواندن هر دو نیم صفحه در متغیرهای سراسری فوق ذخیره خواهد شد.

توجه داریم در صورتی که متغیر current\_state در این تابع برابر یکی از وضعیت­های استاندارد تعریف شده نباشد، آنگاه بورد وارد یک وضعیت غیر مجاز شده است. در این حالت وضعیت فعلی بورد برابر سکوت قرار داده می­شود و زمان بورد که اشاره­گر به آن به عنوان ورودی به تابع داده شده است برابر صفر قرار می­گیرد. این انتخاب به نرم­افزار اجازه می­دهد تا پس از خروج از تابع Flash\_Data\_Read()، بلافاسله تابع Change\_State() را فراخوانی کرده و وضعیت بعدی بورد را تعیین کند.

توجه داریم چهار متغیر ورودی دیگر تابع فوق یعنی سه اشاره­گر به آلبوم­ها و همینطور شماره آلبوم در تابع MUS\_Page\_Reset() کاربرد دارند که در ادامه در مورد آن بحث خواهد شد.

#### تابع بازگردانی صفحه موسیقی به ابتدای آلبوم (MUS\_Page\_Reset())

|  |  |
| --- | --- |
| **کارکرد:** | بازگرداندن صفحه فعلی موسیقی (متغیر MUS\_current\_page) به ابتدای آلبوم. |
| **ورودی­ها:** | شماره آلبوم فعلی (current\_album)، اشاره­گر به محل آلبوم اصلی در ماتریس page\_address، اشاره­گر به محل آلبوم اضافی یک در ماتریس page\_address، اشاره­گر به محل آلبوم اضافی دو در ماتریس page\_address |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** همانگونه که در توضیحات بورد اشاره شد، پس از به اتمام رسیدن یک آلبوم، لازم است پخس مجدد آن با شروع از ابتدای آلبوم در بورد انجام شود. هدف از تابع فوق، انجام همین عملیات با مقایسه MUS\_current\_page با شماره آخرین صفحه آلبوم فعلی در حال پخش است. همانگونه که اشاره شد شماره صفحات ابتدایی و انتهایی تمامی صوت­ها در ماتریس page\_address نگه­داری می­شود. بنابراین در این تابع مقدار MUS\_current\_page با دومین ستون متناظر از ماتریس فوق و با سطر متناظر با شماره آلبوم فعلی مقایسه می­شود. توجه داریم تنها در وضعیت­های MUS، MUS\_1 و FLOOOR\_DEBOUNCE نیاز به بازگرداندن آلبوم به ابتدای آن است و به همین دلیل این تابع در حلقه Switch تابع Flash\_Data\_Read و تنها در سه وضعیت فوق فراخوانی می­شود. ضمن اینکه عدم فراخوانی این تابع در وضعیت MUS\_3 به این دلیل است که با رسیدن آلبوم فعلی به انتهای آن در مد انتخاب آلبوم، پخش موسیقی متوقف شود.

#### تابع خواندن یک صفحه داده از فلش (Flash\_Page\_Read())

|  |  |
| --- | --- |
| **کارکرد:** | تنظیم آدرس و خواندن داده از آدرس مورد نظر در فلش AT45DB321E (با انتخاب از میان دو فلش موجود بر روی بورد). |
| **ورودی­ها:** | شماره صفحه (page\_number) فعلی، شماره نیم صفحه فعلی (half\_page\_number) |
| **خروجی­:** | ‘1’ در صورت اتمام خواندن یک صفحه از حافظه، ‘0’ در غیر این صورت. |



شکل ‏4‑20: فلوچارت تابع Flash\_Page\_Read()

**شرح تفصیلی**: تابع Flash\_Page\_Read وظیفه فعال­سازی دسترسی به حافظه فلش توسط ارتباط SPI براساس شماره صفحه مورد نظر و همینطور شماره نیم صفحه را بر عهده دارد.

همانگونه که در فصل سوم اشاره شد، برای تمایز دادن میان داده­هایی که در اولین حافظه فلش یا دومین حافظه فلش نوشته شده است، شماره صفحه تعریف شده در بورد برای داده­های موجود در دومین فلش یک آفست 8100 صفحه­ای (متناظر با تعداد صفحات فلش) دارند. بنابراین اگر شماره صفحه دریافتی در ورودی تابع بیش از 8100 باشد، متغیر memory\_number که نشان دهنده شماره حافظه­ای است که قرار است داده از آن خوانده شود برابر دو قرار داده می­شود همچنین اگر خواندن از فلش دوم مد نظر باشد، آفست شماره صفحات به گونه­ای که باز گردانده می­شود که شماره صفحه 8101 متناظر با صفحه صفر حافظه دوم گردد.

پس از انتخاب حافظه فلش، نوبت به تنظیم بیت­های آدرس می­رسد. به این منظور ابتدا بیست و چهار بیت آدرس برای دسترسی به یک صفحه و بایت مشخص در حافظه فلش (به بخش 2-6-2 رجوع شود) به ترتیب از با اهمیت­ترین به کم اهمیت­ترین بایت در کاراکترهای Add\_First\_Byte، Add\_Second\_Byte و Add\_Third\_Byte ذخیره می­شوند. با توجه به اینکه هر نیم صفحه برابر 264 بیت می­باشد، لازم است نه بیت کم اهمیت آدرس صفحه به مقدار صفر یا 264 تنظیم شوند. . برای اینکار و با توجه به اینکه متغیر ورودی page\_number یک عدد صحیح 16 بیتی است، با شش­بار شیفت دادن این متغیر به سمت راست می­توان MSB آدرس صفحات به علاوه تعداد صفر مورد نیاز جهت کامل کردن رشته بیت را استخراج کرد. هشت بیت دوم به این صورت به این صورت تشکیل می­شود که از آنجا که قرار است داده یا از اولین بایت صفحه یا بایت 264 خوانده شود، متغیر page\_number دو بیت به چپ شیفت داده می­شود تا هشت بیت میانی رشته آدرس تشکیل شود. در نهایت تک بایت MSB شماره نیم صفحه در یک حلقه if به این مقدار افزوده می­شود و سپس بایت سوم آدرس بر اساس شماره نیم صفحه در همین حلقه تشکیل خواهد شد.

پس از تنظیم آدرس نوبت به خواندن داده از فلش می­رسد که اینکار با فراخوانی تابع SPI\_Interface() و ارسال دستورهای عملیاتی مناسب به حافظه انجام می­شود. برای اینکار ابتدا تابع Memory\_Selection() فراخوانی می­شود تا براساس شماره حافظه، سیگنال انتخاب چیپ به یکی از حافظه­ها فرستاده شود. به این ترتیب فعالیت فلش در این چرخه آغاز می­شود. سپس کد عملیاتی D2H به حافظه ارسال و در پی آن بایت­های آدرس و چهار بیت بدون اهمیت ارسال می­شوند. حال خواندن صفحه فعلی از حافظه به صورت بایت به بایت توسط حلقه for (که به اندازه 264 معادل نیم صفحه داده تکرار می­شود) انجام می­شود. توجه داریم در این حلقه نیازی به فراهم­آوری هیچ بایت آدرسی نیست (چون خواندن پیوسته است) و تنها بایت بی­اهمیت 0x00 به حافظه ارسال می­شود. در آخرین مرحله نیز سیگنال انتخاب چیپ توسط تابع Memory\_Selection غیر فعال می­شود.

آخرین عملیاتی که در این تابع انجام می­پذیرد، تنظیم خروجی آن است. در صورتی که خواندن یک صفحه در این تابع به اتمام رسیده باشد، خروجی این تابع برابر یک و در غیر این صورت برابر صفر است. پایان یافتن خواندن یک صفحه توسط متغیر شماره نیم صفحه تعیین می­شود. برگرداندن یک توسط این تابع به تابع Flash\_Data\_Read اجازه می­دهد شماره صفحات خوانده شده متناظر با پیام فعلی را یک واحد افزایش دهد.

#### تابع ارتباط SPI (SPI\_Interface())

|  |  |
| --- | --- |
| **کارکرد:** | ارسال داده از طریق ارتباط SPI. |
| **ورودی­ها:** | یک بایت داده ارسال. |
| **خروجی­:** | داده دریافت شده از سمت Slave ارتباط (رجیستر SPDR). |

**شرح تفصیلی:** از این تابع برای ارسال و دریافت داده از/ به فلش استفاده می­شود. همانگونه که در بخش 2-6-1 توضیح داده شد، برای ارسال داده در ارتباط SPI بر روی خط MOSI به سمت Slave باید آن را داخل رجیستر SPDR قرار داد. پس از اینکار تعداد هشت پالس ساعت لازم است که داده از این رجیستر به فلش منتقل شود و برعکس داده موجود در رجیستر فلش داخل SPDR قرار گیرد. پایان این پروسه با تغییر بیت چهارم رجیستر SPSR اعلان می­شود و پس از اینکه این رجیستر صفر شد، داده خوانده شده از فلش در رجیستر SPDR قرار گرفته است. کارکرد حلقه while موجود در این کد این است که به صورت مستمر یک شدن این رجیستر را بررسی می­کند. پس از تایید پایان انتقال یافتن داده توسط این حلقه while، داده خوانده شده از فلش بر روی رجیستر SPDR قرار گرفته است و به این ترتیب با تعریف آن به عنوان خروجی، کارکرد تابع پایان می­پذیرد.

#### تابع انتخاب چیپ حافظه (Memory\_Selection())

|  |  |
| --- | --- |
| **کارکرد:** | انتخاب یکی از حافظه­های موجود بر روی بورد با ارسال سیگنال انتخاب چیپ / قطع ارتباط با تمامی چیپ­ها |
| **ورودی­ها:** | شماره حافظه (memory\_number) |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** این تابع وظیفه ارسال سیگنال انتخاب چیپ (یک سیگنال با منطق صفر) به چیپی را دارد که توسط متغیر memory\_number معین شده است. منطق انتخاب چیپ حافظه در فصل دوم و همینطور فصل پنجم شرح داده شده است. بنابراین اگر شماره یک چیپ این تابع داده شود، تابع پین PB0 را به صفر و یا یک تنظیم می­کند و سپس پین CS را نیز به صفر تنظیم می­کند. در این حالت سیگنال CS به چیپ مورد نظر ارسال خواهد شد. اگر memory\_number برابر صفر قرار داده شود، آنگاه سیگنال انتخاب چیپ قطع خواهد شد که این کار با یک کردن پین CS و صفر کردن پین PB0 انجام می­شود. توجه داریم تنظیم تمامی پین­ها به گونه­ای در کد صورت می­پذیرد که منطق موجود در سایر پایه­های میکرو تغییری نکند.

#### تابع رمزگشایی فایل موسیقی (Music\_Deciphering())

|  |  |
| --- | --- |
| **کارکرد:** | رمزگشایی فایل­های موسیقی ذخیره شده در حافظه فلش |
| **ورودی­ها:** | شماره بایت |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** این تابع وظیفه رمزگشایی فایل­های موسیقی را برعهده دارد. برای رمزگشایی لازم است 4 بیت­ LSB یک بایت موسیقی با 4 بیت MSB آن جابجا شوند و در ضمن همه بیت­ها NOT بشوند (بخش ‏3-4-).

توجه داریم فراخوانی این تابع بلافاصله پس از خواندن یک بایت موسیقی از حافظه فلش در تابع Flash\_Page\_Read() است. به منظور رمزگشایی از این بایت خوانده شده، پس از ذخیره یک کپی از آن در یک متغیر کمکی، ابتدا 4 بیت بالا با استفاده یک شیفت بیتی به پایین منتقل می­شوند، سپس چهار بیت پایین از متغیر کمکی خوانده شده و به بالا شیفت داده می­شوند و حاصل در ماتریس buffer\_data ذخیره می­شود. در نهایت نیز این داده NOT منطقی می­شود.

### ماژول تنظیمات اولیه فلش AT45DB321E

این ماژول که با نام AT45DB321E\_Setup.c در کد نام­گذاری شده است، برای تعیین تعداد حافظه­های فلش نصب شده بر روی بورد و تنظیم اندازه صفحات آنها به 528 بایت استفاده می­شود. کلیه توابع این ماژول در تابع Initial\_Setup() در ماژول Main فراخوانی می­شوند تا تنظیمات اولیه مربوط به فلش­ها قبل از آغاز عملیات اصلی بورد صورت بپذیرد. جدول ‏4‑14 نشان دهنده توابع تعریف شده در این ماژول و جدول ‏4‑15 نشان دهنده ثوابت تعریف شده برای آن هستند. شرح کامل هر یک از توابع در ادامه خواهد آمد. یادآوری می­شود که علت نیاز به هر یک از توابع نیز در بخش ‏2-6-3- آمده است.

جدول ‏4‑14: فهرست توابع تعریف شده در ماژول تنظیمات اولیه فلش AT45DB321E

|  |  |
| --- | --- |
| تابع | کارکرد |
| Flash\_Page\_Size | تعیین اندازه صفحات حافظه فلش |
| Set\_Flash\_Register | تنظیم رجیستر وضعیت فلش برای اندازه صفحات 528 بایتی |
| Flash\_Erase | پاک کردن داده­های موجود در فلش |
| Count\_Memory\_Chips | بررسی تعداد حافظه­های فلش موجود در بورد |

جدول ‏4‑15: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول تنظیمات اولیه فلش AT45DB321E

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| Status\_reg\_Read | ارسال سیگنال انتخاب چیپ بر روی پروت SS ارتباط SPI |
| PAGE\_SIZE\_BIT | بیت اندازه صفحات حافظه AT45DB321E |
| FLASH\_BUSY\_BIT | بیت مشغول به کار بودن حافظه فلش |
| CHANGE\_SIZE\_OPCODE1 تا CHANGE\_SIZE\_OPCODE4 | چهار کد عملیاتی برای تغییر اندازه صفحات حافظه به 528 بایت |
| ERASE\_OPCODE1 تا ERASE\_OPCODE4 | چهار کد عملیاتی برای پاک کردن تمامی داده­های موجود در حافظه فلش |

#### تابع بررسی وجود دومین حافظه فلش (Count\_Memory\_Chips())

|  |  |
| --- | --- |
| **کارکرد:** | بررسی وجود دو حافظه بر روی بورد |
| **ورودی­ها:** | ماتریس شماره صفحات پیام­ها / موسیقی­ها (page\_address[][2]) |
| **خروجی­:** | تعداد چیپ­های حافظه موجود بر روی بورد |

**شرح تفضلی:** این تابع با شروع از اولین سطر ماتریس page\_address، دومین درایه هر سطر (که متناظر با آخرین صفحه این پیام / موسیقی است) را با 8100 یا تعداد صفحات حافظه فلش مقایسه می­کند. در صورتی که یکی از این اعداد از 8100 بزرگتر باشد، متغیر number\_of\_memories به دو تنظیم می­شود که نشان دهنده وجود دو چیپ حافظه در بورد است. در ادامه همین عدد به عنوان خروجی تابع بازگردانده می­شود.

#### تابع بررسی اندازه صفحات فلش (Flash\_Page\_Size())

|  |  |
| --- | --- |
| **کارکرد:** | تعیین اندازه صفحات حافظه فلش با استفاده از رجیستر وضعیت آن |
| **ورودی­ها:** | شماره چیپ حافظه (memory number) |
| **خروجی­:** | اندازه صفحات |

**شرح تفصیلی:** با استفاده از این تابع می­توان اندازه صفحات حافظه فلشی که با شماره memory number معین می­شود را تعیین کرد. به این منظور و پس از تنظیم سیگنال انتخاب چیپ با استفاده از تابع Memory\_Selection()، فرمان دریافت رجیستر وضعیت که با ثابت Status\_reg\_Read نمایش داده شده است به حافظه ارسال می­شود. سپس حافظه این رجیستر را باز می­گرداند که بیتی که با ثابت PAGE\_SIZE\_BIT نشان داده شده حاوی اطلاعات اندازه صفحات است. توجه داریم برای حصول اطمینان از اینکه این بیت در هنگام خواندن دچار نویز نشده است، عمل خواندن این بیت یازده­بار تکرار می­شود و حاصل در متغیری به نام vote ذخیره می­شود. اگر این متغیر بیش از پنج بار برابر یک باشد آنگاه اندازه صفحات برابر 512 و در غیر این صورت برابر 528 است.

#### تابع پاک کردن داده­ها از حافظه فلش (Flash\_Erase())

|  |  |
| --- | --- |
| **کارکرد:** | پاک کردن تمامی داده­های موجود بر روی حافظه فلش |
| **ورودی­ها:** | شماره چیپ حافظه (memory number) |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** از این تابع برای پاک کردن داده­های موجود بر روی حافظه فلش استفاده می­شود. به این منظور ابتدا چیپ شماره memory\_number فعال شده و سپس چهار کد عملیاتی که با ثابت­های ERASE\_OPCODE1 تا ERASE\_OPCODE4 نمایش داده می­شوند به حافظه ارسال می­شوند و ارسال سیگنال انتخاب چیپ پس از این عملیات متوقف می­شود. پس از انجام عملیات فوق، مجدداً چیپ انتخاب شده و سپس بیت مشغول بودن حافظه که با ثابت FLASH\_BUSY\_BIT نمایش داده می­شود از حافظه خوانده می­شود. تا زمانی که این بیت صفر باشد، حافظه مشغول پاک کردن اطلاعات است و به همین دلیل به برنامه اجازه داده نمی­شود که به قسمت­های بعدی مراجعت کند. پایان این عملیات به معنای خالی شدن حافظه است.

#### تغییر اندازه صفحات حافظه فلش (Change\_Page\_Size())

|  |  |
| --- | --- |
| **کارکرد:** | تغییر اندازه صفحات فلش به 528 بایت با فرستادن کدهای عملیاتی مناسب |
| **ورودی­ها:** | شماره چیپ حافظه (memory number) |
| **خروجی­:** | فاقد خروجی |

**شرح تفصیلی:** از این تابع برای تغییر اندازه صفحات حافظه فلش استفاده می­شود. کارکرد این تابع مشابه تابع Flash\_Erase() است و پس از ارسال کدهای عملیاتی که با ثوابت CHANGE\_SIZE\_OPCODE1 تا CHANGE\_SIZE\_OPCODE4 تعریف می­شوند و پس از آن منتظر ماندن برای تغییر بیت مشغول بودن حافظه، اندازه صفحات به 528 بایت تغییر می­کند.

### ماژول مد یادگیری در بورد FSB300

این ماژول که با نام Learn\_Mode.c تعریف شده است، توابع مرتبط با مد یادگیری در بورد FSB300 را در خود دارد. به این ترتیب محتوای این ماژول تنها در این بورد و تنها در صورتی که مد تست فعال شود فراخوانی   
می­شوند. جدول ‏4‑16 نشان دهنده توابع تعریف شده در این ماژول و جدول ‏4‑17 نشان دهنده ثوابت تعریف شده در هدر این ماژول هستند. یادآوری می­شود که مدت زمان توقف بورد در مد یادگیری دو برابر مقدار LEARN\_DEBOUNCE\_NUMBER منهای یک است.

جدول ‏4‑16: فهرست توابع تعریف شده در ماژول مد یادگیری

|  |  |
| --- | --- |
| تابع | کارکرد |
| Learn\_Mode\_Assignments | تنظیم سطح سیگنال­های کنترلی بر اساس سیگنال کنترلی دیبانس شده |
| DOBTRG\_Level\_Assignment | بازگرداندن سطح سیگنال متناظر با سیگنال­های DOB، OVL و TRG بر اساس مقدار سیگنال کنترلی |

جدول ‏4‑17: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول مد یادگیری

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| LEARN\_DontC\_bits | بیت­های بدون اهمیت پورت کنترلی در مد یادگیری |
| DOBAL\_TRGAL | شکل سیگنال کنترلی اگر DOB به صورت AL و TRG به صورت AL باشد |
| DOBAH\_TRGAL | شکل سیگنال کنترلی اگر DOB به صورت AH و TRG به صورت AL باشد |
| DOBAH\_TRGAH | شکل سیگنال کنترلی اگر DOB به صورت AH و TRG به صورت AH باشد |
| DOBAL\_TRGAH | شکل سیگنال کنترلی اگر DOB به صورت AL و TRG به صورت AH باشد |

#### توابع تنظیم سیگنال­های کنترلی (Learn\_Mode\_Assignments())

|  |  |
| --- | --- |
| **کارکرد:** | تنظیم شکل سیگنال­های TRG، OVL و DOB برمبنای سیگنال­ کنترلی دیبانس شده |
| **ورودی­ها:** | اشاره­گر به پارامترهای سیگنال کنترلی |
| **خروجی­:** | فاقد خروجی |

در مد یادگیری از این تابع برای تنظیم شکل سیگنال­های TRG، OVL و DOB با توجه به سطح منطقی پورت­های متناظر با این سیگنالها استفاده می­شود. همانگونه در مورد تابع Parallel\_CTRL\_Signal\_Detection()ا اشاره شد، آشکارسازی سیگنال کنترلی در مد یادگیری در این تابع انجام نمی­شود و تنها وظیفه این تابع افزایش شمارنده دیبانس است. در تابع Learn\_Mode\_Assignments() در صورتی که تعداد دیبانس­های سیگنال کنترلی از ثابت LEARN\_DEBOUNCE\_NUMBER بیشتر شود، سطح فعلی سه سیگنال DOB، OVL توسط متغیر DOB\_TRG\_active\_level به تابع DOBTRG\_LEVEL\_Assignment() داده می­شود. این تابع بر اساس سطح مقادیر، سه متغیر TRG\_input، OVL\_input و DOB\_input را که در ساختار پارامترهای سیگنال کنترلی تعریف شده­اند تنظیم می­کند. در نهایت نیز متغیر DOB\_TRG\_active\_level در EEPROM ذخیره می­شود تا در چرخه­کاری بعدی بورد و در تابع Initial\_Setup، مقدار آن از EEPROM خوانده شده و با فرخوانی مجدد تابع DOBTRG\_LEVEL\_Assignment()، سطح سیگنال­های کنترلی تنظیم شود.

### ماژول مد تست در بورد FSB300

این ماژول که با نام Test\_Mode.c نام­گذاری شده است، حاوی تابع مد تست بورد FSB300 می­باشد. کلیه ثوابت مورد نیاز در مد تست به غیر از ثابت تعداد دفعات دیبانس مطلوب در این نیز در هدر این ماژول آمده­اند. جدول ‏4‑18 نشان دهنده توابع تعریف شده در این ماژول و جدول ‏4‑19جدول ‏4‑15 نشان دهنده ثوابت تعریف شده در هدر این ماژول هستند. شرح کامل هر یک از توابع در ادامه خواهد آمد.

جدول ‏4‑18: فهرست توابع تعریف شده در ماژول تنظیمات اولیه فلش AT45DB321E

|  |  |
| --- | --- |
| تابع | کارکرد |
| Test\_Input\_Detection | تشخیص پینی که در مد تست تحریک شده است. |

جدول ‏4‑19: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول تنظیمات اولیه فلش AT45DB321E

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| CTRL\_TEST\_DontC\_bits | بیت­های بدون اهمیت پورت کنترلی در مد تست |
| EMR\_TEST\_INPUT تا  TRG\_TEST\_INPUT | شکل سیگنال قرار گرفته در پورت­های ورودی در مد تست برای پین­های پورت کنترلی |
| NUM\_B2\_TEST\_INPUT | شکل سیگنال قرار گرفته در پورت­های ورودی در مد تست برای پین­های پورت شمارنده |
| TOTAL\_NO\_PINS | مجموع تعداد پین­های ورودی بورد FSB300 |
| NO\_PIN\_EXCITED | مجموع تعداد پین­های ورودی بورد FSB300 به علاوه سه |

#### تابع تعیین پین تحریک شده در مد تست (Test\_Input\_Detection())

|  |  |
| --- | --- |
| **کارکرد:** | تعیین پین ورودی که در مد تست تحریک شده است. |
| **ورودی­ها:** | اشاره­گر به پارامترهای سیگنال کنترلی، اشاره­گر به پارامترهای سیگنال شمارنده |
| **خروجی­:** | شماره پین تحریک شده |

**شرح تفصیلی:** هدف از این تابع، تعیین پینی است که بر طبق جدول ‏2‑10 در مد تست تحریک شده است. کارکرد تابع به این صورت است که مقدار خوانده شده از پورت کنترلی و شمارنده را به نحوی داخل یک integer قرار می­دهد که رجیستر داده پورت شمارنده بیت­های MSB این integer را پر کنند. برای اینکار رجیستر داده پورت شمارنده داخل یک integer قرار داده می­شود و سپس هشت بیت به چپ شیفت داده می­شود. پس از اینکار، مقدار عدد بدست آمده توسط یک حلقه switch با مقادیر ارائه شده در جدول ‏2‑10 مقایسه می­شود و به این ترتیب پین تحریک شده تعیین می­شود. توجه داریم اگر هیچ پینی تحریک نشده باشد، حلقه switch مقدار NO\_PIN\_EXCITED را باز می­گرداند که به این مفهوم است که هیچ پین (دارای معنی) تحریک نشده است. این تخصیص باعث می­شود که بتوان در وضعیت SILENCE+2 عملیات مناسب (یعنی خروج از وضعیت و یا ادامه آن بسته به فشرده بودن کلید TEST) را انجام داد.

### ماژول­های ارتباط I2C

این دو ماژول که با نام­های I2C\_Protocol.c و I2C\_ACK\_NACK\_Generation.c نام­گذاری شده­اند، ماژول­های ارتباط I2C می­باشند که توصیف آنها در گزارش «*پروتکل ارتباطی I2C*» آمده است. تنها نکاتی که لازم است در مورد استفاده این مازول­ها در بورد FSB تذکر داده شود این است که:

* پروتکل I2C در قسمت آسنکرون Main تعریف شده است و تنها در بورد FSB310 فعال می­شود.
* ثوابت I2C\_DATA\_LENGTH تا I2C\_FREQUENCY\_PRESCALAR که در هدر General.h تحت عنوان I2C related parameters ذخیره شده­اند، حاوی تعاریف شامل مد کاری، فرکانس کاری، تعداد بایت ارسالی و ... می­باشند که به عنوان ورودی به تابع I2C\_Potocol() داده می­شوند.
* هدر تابع I2C\_ACK\_NACK\_Generation() از ثوابتی که به عنوان سیگنال شمارنده، شماره توقف و همینطور سیگنال کنترلی در هدر IO\_Module.h در ارتباط سریال تعریف شده برای تولید (N)ACK استفاده می­کند.



# بخش سخت­افزاری

## مقدمه

در این فصل در مورد سخت­افزار بورد FSB300 و بورد FSB310 و مسائل مرتبط با پیاده­سازی بحث می­شود. به دلیل شباهت ساختاری سخت­افزار این دو بورد، ابتدا در مورد سخت­افزار بورد FSB300 بحث می­شود، سپس تنها در مورد تفاوت­های سخت­افزار بورد FSB310 با بورد FSB300 بحث خواهد شد.

## سخت­افزار بورد FSB300

مهمترین تفاوت­های میان سخت­افزار FSB300 و FSB200a، تغییر در IC رگولاتور ولتاژ، تغییر در نحوه چینش ورودی­ها و حذف جامپر NC و NO در سیگنال DOB، کاهش تعداد هدرهای مورد استفاده برای پروگرام حافظه فلش و میکرو، تغییر IC تقویت توان مورد استفاده در خروجی PWM، تغییر فیلتر مورد استفاده برای تولید سیگنال آنالوگ و در نهایت امکان سیگنال­دهی به دو چیپ AT45DB321E است. علاوه بر موارد فوق، دو تغییر عمده دیگر در بورد در مقایسه با FSB200a به شرح زیر می­باشند:

* **حذف کریستال 16 مگاهرتزی**: در نسخه قبلی بورد از یک کریستال خارجی 16 مگاهرتزی برای تولید فرکانس پالس مورد استفاده در بورد استفاده شده است. در بورد جدید (و با تنظیم مناسب زمان­بندی­ها)، از کریستال داخلی میکروکنترلر با فرکانس 8 مگاهرتز استفاده شده است و   
  آزمایش­های مکرر نشان داد که استفاده از این اسیلاتور باعث افت کیفیت صدای پخش شده از بورد نمی­شود.
* **حذف حافظه میکرو SD خارجی و سوکت­های آن و LED آلبوم:** همانگونه که در فصل دوم اشاره شد، استفاده از روش­های فشرده­سازی صوت باعث ایجاد امکان ذخیره سه آلبوم بر روی حافظه فلش بورد شده است. به همین دلیل حافظه میکرو SD خارجی و همچنین سوکت آن در نسخه جدید حذف شده است و LED آلبوم نیز که کارکرد اصلی آن نشان دادن ورود بورد FSB200a به مد تغییر آلبوم است در بورد FSB300 وجود ندارد.

در ادامه این فصل به بررسی تغییرات صورت گرفته در سخت­افزار این بورد پرداخته می­شود

### استفاده از رگولاتور ولتاژ LM1117-3.3 به جای LF33 در تغذیه مدار

استفاده از IC رگولاتور LM1117-3,3v به جای IC رگولاتور LF33 در بورد FSB300 مهمترین تغییر صورت گرفته در تفذیه مدار می­باشد. IC رگولاتور جدید تفاوت قیمتی قابل توجهی با IC قدیمی با حفظ کیفیت مطلوب دارد که همین مطلب استفاده از آن را در بورد توجیه می­کند. پسوند 3.3 در کنار نام IC نشان دهنده سطح ولتاژ خروجی آن (3.3v) می­باشد که برای تغذیه میکروکنترلر و حافظه فلش مطلوب است.

### تغییر در چینش ورودی­ها و حذف جامپر NC و NO

در نسخه قبلی بورد FSB ورودی اورژانسی از پورت PB0 می­شود و LED آلبوم نیز از پورت PC2 تغذیه شده است. با حذف LED آلبوم، سیگنال اورژانسی در نسخه جدید از پورت PC2 دریافت می­شود که باعث می­شود تمامی سیگنال­های کنترلی از پورت C دریافت شوند. علاوه بر این مطلب، به دلیل اضافه کردن مد یادگیری در بورد، دیگر نیازی به تنظیم جامپرهای NO و NC برای تطبیق دادن سطح سیگنال­ DOB نیست و بورد به صورت خودکار (و با اعمال ورودی­های مناسب) خود را با این ورودی­ها تطبیق می­دهد. حذف این جامپر باعث حذف ترانزیستور T1 و متعلقات آن نسبت به بورد FSB200a شده است. اما توجه داریم جامپرهای AH برای هر دو ورودی TRG و DOB وجود دارند که باعث می­شود بتوان سطح منطقی آنها را به دلخواه تنظیم نمود.

### کاهش تعداد هدرهای پروگرام بورد و حذف جامپرهای خارجی آن

در بورد FSB200a، تعداد 72 هدر در بورد تعبیه شده است و از هفت­تای آنها برای پروگرام حافظه فلش و از هفت­ عدد دیگر برای پروگرام میکروکنترلر استفاده می­شود. علت استفاده از دو مجموعه هفت­تایی از هدر­ها، ایزوله کردن میکروکنترلر و همینطور حافظه فلش از یکدیگر در هنگام پروگرام هر یک از این دو عنصر است. بنابراین پس از پروگرام دو قطعه، این مجموعه هدرها به توسط هفت جامپر به یکدیگر متصل می­شوند.

توجه داریم اگر تغذیه و زمین را در نظر نگیریم، برای پروگرام هر یک از این دو IC تنها 5 سیگنال احتیاج است (شامل SCK، RESET، MISO، MOSI و CS). از طرف دیگر قرار دادن هفت جامپر در بورد فضای قابل توجهی را اشغال کرده است، ضمن اینکه استفاده از جامپر و قرار دادن آن وقت­گیر بوده و ممکن است جامپرها در محل کار بورد دچار مشکل شوند. برای حذف جامپرها و همچنین کاهش تعداد هدرها، با در نظر گرفتن اینکه:

1. برای پروگرام میکروکنترلر از پایه CS استفاده نمی­شود، در حالی که برای پروگرام فلش لازم است این پایه دارای منطق صفر باشد. بنابراین اگر حافظه فلش قبل از میکروکنترلر پروگرام شده باشد، می­توان میکروکنترلر را بدون نگرانی از وجود حافظه با استفاده از یک مجموعه خط یکسان پروگرام کرد.
2. الگوریتم پروگرام میکروکنترلر ATMega8 که در بخش Serial Programming Algorithm دیتاشیت آن آمده است نشان می­دهد که چند قدم جداگانه و پیچیده برای هماهنگ­سازی میان میکروکنترلر و پروگرامر لازم است. به عنوان مثال در ابتدای کار، لازم است همزمان­سازی (synchronization) میان IC و پروگرامر صورت بپذیرد. از آنجا که برای پروگرام حافظه فلش تنها لازم است پایه CS زمین نگاه داشته شود و تنها چند opcode به قطعه ارسال شود، پروگرام این قطعه در صورتی که به صورت همزمان به میکروکنترلر متصل باشد مشکلی ایجاد نمی­کند.

با توجه به مطالب فوق، می­توان از یک خط هفت­تایی هدرها که به یکدیگر متصل شده­اند برای پروگرام میکروکنترلر و حافظه فلش استفاده کرد. برای اینکار:

1. ابتدا میکروکنترلر از هرگونه داده­ای خالی می­شود (به خصوص لازم است هیچگونه تعاریفی مرتبط با ارتباط SPI در آن وجود نداشته باشد)، سپس حافظه فلش پروگرام می­شود.
2. پس از پروگرام حافظه فلش، میکروکنترلر نیز پروگرام می­شود.

به این ترتیب می­توان تنها با یک مجموعه هدر هر دو عنصر را پروگرام کرد.

### تقویت کننده توان LM386

از این IC برای تقویت توان سیگنال صوت آنالوگ پس از فیلتر آنالوگ استفاده می­شود. IC فوق یک تقویت کننده آپ-امپی است که توان خروجی آن در ولتاژ ورودی 9V برای بسته به نوع IC از *تا تغییر می­کند. برای کارکرد مناسب* IC *ولتاژ پیک تا پیک ورودی نباید از تجاوز کند. توجه داریم در بورد*FSB200a *از* LM380 استفاده شده است که اولاً این IC توان خروجی تا 2.5W را تامین می­کند و در ضمن دارای مدار محافظ در برابر اتصال کوتاه است. اما از آنجا که یک بلندگوی 0.6W برای استفاده در مدار فوق کفایت می­کند، ترجیح بر استفاده از LM386 است، زیرا قیمت مدل LM386 تقریباً یک هفتم قیمت LM380 است و بنابراین استفاده از این IC قیمت تمام شده مدار را نیز کاهش می­دهد. به دلیل سهولت تهیه، در این پروژه از IC نوع LM386s استفاده شده است که SOC می­باشد. مهمترین مشکل استفاده از IC فوق، عدم وجود مدار محافظت در برابر اتصال کوتاه می­باشد که در ادامه چگونگی حفاظت این IC در برابر اتصال کوتاه نیز ارائه خواهد شد.

شماتیک IC فوق در شکل ‏5‑1-الف آمده است. برای راه­اندازی IC اولاً ولتاژ تغذیه برابر 12 ولت انتخاب   
می­شود، ثانیاً منبع ولتاژ با یک خازن زمین می­شود تا نویز منبع تغذیه کاهش یابد. برای کاهش هرچه بیشتر نویز همچنین یک خازن 56nF از پایه 7 (پایه bypass) به زمین متصل می­شود. ضمن اینکه پایه­های 2، 3 و 4 و همینطور پایه 5 IC زمین می­شوند. پایه­های 1 و 8 این IC برای تنظیم بهره IC مورد استفاده قرار می­گیرند. اگر این دو پایه اتصال باز باشند، مقدار بهره برابر 20dB خواهد بود. ضمن اینکه اگر یک خازن میان این دو پایه قرار بگیرد بهره به بیشینه به مقدار خود خواهد رسید، اما سیگنال خروجی دارای اعوجاج زیادی خواهد شد. استفاده از یک مقاومت در کنار خازن باعث می­شود که بهره سطح مناسبی داشته باشد و در ضمن سیگنال خروجی دارای اعوجاج نباشد. در نهایت برای کاهش نویز در خروجی مدار از یک خازن 470 *استفاده می­شود. بدین ترتیب توپولوژی مدار به صورت* شکل ‏5‑1-ب خواهد شد. رعایت پلاریته خازن­ها در پیاده­سازی مدار از اهمیت خاصی برخوردار است.

(الف) شماتیک مدار (ب) توپولوژی پیاده­سازی شده

شکل ‏5‑1: شماتیک IC تقویت کننده توان LM386

#### محافظت IC در برابر اتصال کوتاه

برای محافظت مدار در برابر اتصال کوتاه می­توان در خروجی مدار و پس از خازن یک مقاومت با وات بالا قرار داد. این مقاومت باعث افزودن اعوجاج به سیگنال خروجی نمی­شود و سطح صدای کلی مدار را نیز کاهش چندانی نخواهد داد. مقدار این مقاومت و یک وات انتخاب می­شود. انتخاب مقاومت تنظیم می­کند که جریان خروجی بیش از حد مجاز نشود (چون در صورت اتصال کوتاه شدن ورودی مشابه یک بلندگو عمل می­کند)، و یک وات بودن آن نیز به این دلیل است که توان خروجی IC در مرتبه یک وات است. در نهایت اینکه استفاده از این مقاومت باعث نصف شدن توان خروجی و کاهش آن تا سطح 0.5W می­شود، اما همین توان نیز برای ایجاد یک صدای به قدر کافی بلند در بورد کفایت می­کند.

### فیلتر (میان­گذر) مورد استفاده برای بازتولید سیگنال آنالوگ از PWM

در بخش ‏2-4-3- اشاره شد که پس از تولید شکل موج PWM، لازم است این شکل موج توسط فیلتر   
پایین­گذر مناسب فیلتر شود تا صوت خروجی تولید شود. اما برای بهبود کیفیت صدا و به خصوص از آنجا که صوت انسان گستره 300 هرتز تا 3 کیلوهرتز را پوشش می­دهد و همینطور برای فیلتر کردن نویزهای با فرکانس پایین، در این پروژه ترجیح بر استفاده از یک فیلتر میان­گذر است. پاسخ فرکانسی یک فیلتر میان­گذر استاندارد به صورت زیر تعریف می­شود:

|  |  |
| --- | --- |
| (‏5‑1) |  |

که در آن معرف فرکانس مرکزی فیلتر است و برای *،* نماینده پهنای باند فیلتر است ( به عنوان ضریب کیفیت فیلتر نیز شناخته می­شود). به صورت کلی­تر، باند گذار فیلتر به طور تقریبی از قطب اول فیلتر () تا قطب دوم آن () خواهد بود. یادآوری می­شود با حل معادله مخرج بر حسب s، اندازه ریشه­ها برحسب رادیان بر ثانیه به دست خواهند آمد و برای تبدیل آن به هرتز لازم است مقدار حاصل شده تقسیم بر نیز بشود.

همانگونه که در بخش 5-5 اشاره شد، IC تقویت کننده توان LM386 دارای مقاومت ورودی است. از آنجا که فیلتر فوق قبل از پایه ورودی این IC (یعنی ) قرار می­گیرد، با تعریف به عنوان مقاومت ورودی این IC، توپولوژی فیلتر میان­گذر پیشنهادی به صورت زیر خواهد بود:



شکل ‏5‑2: فیلتر میان­گذر پیشنهادی برای فیلتر کردن موج PWM جهت تولید صوت

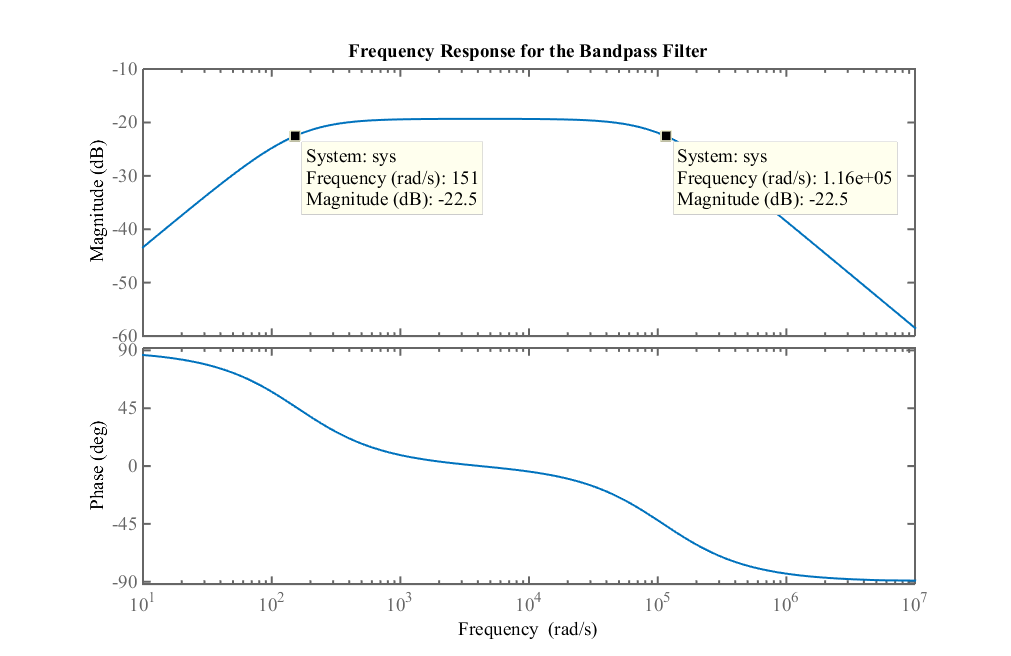
همانگونه که مشاهده می­شود مقاومت به صورت موازی با قرار گرفته است و مقدار آن به گونه­ای انتخاب می­شود که . به این ترتیب اولاً اثر بارگذاری مقاومت فوق حذف می­شود و ثانیاً ولتاژ پیک تو پیک ورودی IC تقویت کننده کمتر از 0.4V خواهد شد که برای آن اعوجاج در خروجی IC به کمترین مقدار ممکن   
می­رسد (طبق مقادیر تعریف شده در دیتاشیت IC). با تفاسیر فوق تابع انتقال فیلتر به صورت زیر می­باشد:

|  |  |
| --- | --- |
| (‏5‑2) |  |

و بنابراین:

|  |  |
| --- | --- |
| (‏5‑3) |  |

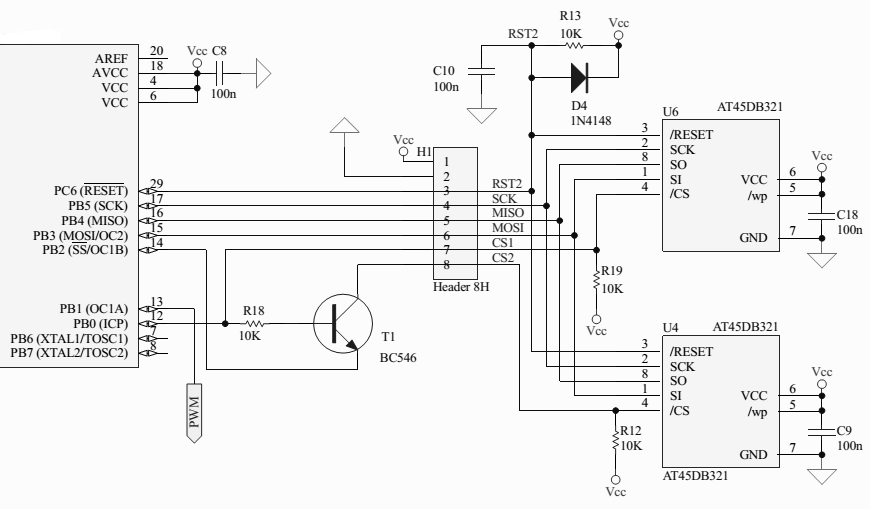
از آنجا که فرکانس نمونه­برداری موسیقی برابر 32kHz انتخاب شده است، لازم است فرکانس قطع بالای فیلتر در حدود 16kHz و یا کمتر باشد. ضمن اینکه انتخاب فرکانس قطع پایین در محدوده 30Hz تا 0.5kHz برای   
دست­یابی به کیفیت مطلوب صوت کفایت می­کند. برای طراحی فیلتر فوق می­توان از کد متلب BandPass\_Filter\_Design استفاده کرد که در پوشه کدهای متلب پروژه آمده است. این کد مقادیر عناصر ، ، و را به عنوان ورودی دریافت می­کند و فرکانس­های قطع، معادله تابع انتقال و در نهایت نمودار اندازه فیلتر بر حسب رادیان بر ثانیه و همینطور فاز آن را نمایش می­دهد[[45]](#footnote-45). به این ترتیب و با انتخاب خازن­های استاندارد *و*  می­توان مقاومت­ها را به صورت و در نظر گرفت که فرکانس­های قطع و را نتیجه خواهد داد، ضمن اینکه بهره فیلتر در باند گذار برابر   
dB است. پاسخ فرکانسی فیلتر پیشنهادی در شکل ‏5‑3 نمایش داده شده است.

**

شکل ‏5‑3: پاسخ فرکانسی فیلتر پیشنهادی

### سیگنال­دهی دو چیپ AT45DB321E توسط ارتباط SPI

همانگونه که در بخش ‏2-6-6- اشاره شد، امکان سیگنال­دهی به دو چیپ حافظه توسط ارتباط SPI وجود ندارد و لازم است یک مدار کمکی نیز در کنار این ارتباط مورد استفاده قرار بگیرد. پیکربندی مدار پیشنهادی و همینطور نحوه پیاده­سازی چیپ­های حافظه مدار در شکل ‏5‑4 آمده است.



شکل ‏5‑4: پیکربندی مدار ارسال سیگنال CS در ارتباط SPI به یکی از دو چیپ حافظه دلخواه

همانگونه که مشاهده می­شود سیگنال CS میکروکنترلر (پایه PB2 و یا SS) در این بورد به عنوان یک ورودی به امیتر ترانزیستور داده می­­شود و پایه PB0 نیز به بیس این ترانزیستور وصل می­شود. همچنین یک خروجی نیز به صورت مستقیم از پایه PB0 گرفته می­شود. به این ترتیب واضح است که هنگامی که سیگنال CS منطق یک داشته باشد، اولاً ترانزیستور غیر فعال است و در نتیجه در کلکتور آن جریانی وجود ندارد، ثانیاً هر دو چیپ برای حالت  
PB0 = 1 غیر فعال هستند. در صورتی که PB0 در این حالت منطق یک داشته باشد، چیپی که سیگنال CS را از پایه PB0 دریافت می­کند فعال می­شود، اما از آنجایی که ارتباط SPI در این حالت فعال نیست (چون سیگنال CS برابر یک است)، هیچ داده­ای از این فلش خوانده نمی­شود / به آن ارسال نمی­شود. حال اگرسیگنال CS برقرار شود (صفر شود)، اگر پایه PB0 منطق صفر داشته باشد، ارتباط SPI با چیپ یک برقرار می­شود و اگر منطق آن یک باشد، ترانزیستور فعال شده و با کشیدن جریان از مقاومت Pull-up وصل شده به چیپ دوم باعث برقراری ارتباط با این چیپ می­شود. جدول درستی این مدار در جدول ‏5‑1 آمده است. به این ترتیب می­توان با استفاده از یک سیگنال انتخاب چیپ، هر دو چیپ موجود در مدار را فعال کرد. جدول توجه داریم مقاومت قرار گرفته در بیس مدار باعث می­شود که جریان کلکتور محدود شده و در نتیجه جریان بیش از حدی از میکروکنترلر عبور نکند.

جدول ‏5‑1: فعال­ شدن CSها با توجه به سطح منطقی PB0 و سیگنال انتخاب چیپ

|  |  |  |  |
| --- | --- | --- | --- |
| CS2 | CS1 | پایه SS میکروکنترلر | PB0 |
| غیر فعال | فعال | 1 | 0 |
| غیر فعال | غیر فعال | 1 | 1 |
| غیر فعال | فعال | 0 | 0 |
| فعال | غیر فعال | 0 | 1 |

## سخت­افزار بورد FSB310

به صورت کلی سخت­افزار بورد FSB310 بسیار شبیه بورد FSB300 می­باشد، اما چند تفاوت عمده در آنها وجود دارد:

1. تمامی پورت­های دریافت سیگنال در مقایسه با بورد FSB300 حذف شده­اند.
2. از آنجا که سازوکار بررسی 24 ولت بودن ورودی در بورد LCB216 پیاده­سازی شده است، فیوزها و دیود­های رگولاتور ولتاژ (فیوز F1، دیود DZ1 و غیره) حذف شده­اند.
3. در مورد خطوط ارتباط I2C، از آنجا که بورد FSB300 در مد Slave فعالیت می­کند، ترجیح داده شده است مقاومت­های Pull-up ارتباط در سمت بور CIB216 پیاده­سازی شود.

به غیر از مواردی که در بالا اشاره شد، سایر قسمت­های سخت افزار بورد FSB310 مشابه بورد FSB300   
می­باشد.

# مراجع

1. Floor Speech Board [↑](#footnote-ref-1)
2. نسخه­های قبلی بورد عبارتند از FSB100 و FSB200a. [↑](#footnote-ref-2)
3. Control Signal [↑](#footnote-ref-3)
4. Numerator Signal [↑](#footnote-ref-4)
5. به بخش­های ‏2-9- و ‏5-3- این گزارش و همینطور گزارش «*پروتکل ارتباطی I2C*» رجوع شود. [↑](#footnote-ref-5)
6. Power On Reset: روشن شدن بورد پس از ریست [↑](#footnote-ref-6)
7. این تغییرات باید در ماژول ورودی / خروجی نرم­افزار (بخش ‏4-3-3-) انجام شود. [↑](#footnote-ref-7)
8. به بخش ‏4-3-1- رجوع شود. [↑](#footnote-ref-8)
9. به بخش 2-8-1 رجوع شود. [↑](#footnote-ref-9)
10. به بخش 2-6-2 رجوع شود. [↑](#footnote-ref-10)
11. لازم به ذکر است در نسخه جدید امکان وقوع رخداد فوق به صورت کلی از بین می­رود. زیرا اولاً بورد تنها در POR وارد وضعیت اورژانسی می­شود و ثانیاً در این وضعیت اجازه پخش هیچ پیامی غیر از پیام اورژانسی و موسیقی آن داده نخواهد شد. [↑](#footnote-ref-11)
12. Concurrent [↑](#footnote-ref-12)
13. اطلاعات بیشتر در www.soundfile.sapp.org/doc/WaveFormat/ موجود است. [↑](#footnote-ref-13)
14. Duty Cycle [↑](#footnote-ref-14)
15. Pulse Width Modulation [↑](#footnote-ref-15)
16. Dynamic Range [↑](#footnote-ref-16)
17. Lossless [↑](#footnote-ref-17)
18. Lossy [↑](#footnote-ref-18)
19. چگونگی تغییر اندازه صفحات در بخش 10 دیتاشیت IC آمده است. [↑](#footnote-ref-19)
20. Control signals [↑](#footnote-ref-20)
21. مقدار 0x04 با عنوان EMR\_DontCare\_bits در کد تعریف شده است. به فصل چهارم رجوع شود. [↑](#footnote-ref-21)
22. توجه داریم در زمانی که آسانسور در حال حرکت است، ممکن است مسافران به صورت ناخواسته در مسیر سنسور سیگنال DOB قرار بگیرند که این عمل منجر به تولید سیگنال DOB ناخواسته خواهد شد. [↑](#footnote-ref-22)
23. Numerator signals [↑](#footnote-ref-23)
24. علت این امر، پیاده­سازی بوردهای فرمان موجود در بازار می­باشد. مفهوم شماره توقف یکی از مفاهیمی است که در تولید بوردهای فرمان شرکت صانیک از آن استفاده می­شود، و لزوماً در پیاده­سازی بوردهای فرمان دیگر مورد استفاده قرار نمی­گیرد! [↑](#footnote-ref-24)
25. به صورت قراردادی در پیاده­سازی فعلی فرض شده است که یک بودن هر دو این سیگنال­ها به این معناست که در طبقات بیش از نوزده و کمتر از بیست نه قرار داریم. توجه داریم سیگنال­­دهی متناظر با تعریف فوق در بورد فرمان اصلی تعریف نشده است. [↑](#footnote-ref-25)
26. توجه داریم حذف پورت D1 میکرو ابهامی در شماره طبقات ایجاد نمی­کند و به این دلیل مجاز به حذف آن هستیم. [↑](#footnote-ref-26)
27. debounce [↑](#footnote-ref-27)
28. از آنجا که نمایش M توسط سون سگمنت ممکن نمی­باشد، مدیریت با یک صندلی نشان داده شده است! [↑](#footnote-ref-28)
29. البته لازم به ذکر است که در صورتی که موسیقی­های بورد بنا به درخواست مشتری تغییر کنند لازم است تمامی پیام­ها به صورت کامل در بورد بررسی شوند و این مورد باید بعد از آزمایش صحت کارکرد بورد (آزمایشی که در ادامه خواهد آمد) صورت بپذیرد. [↑](#footnote-ref-29)
30. پورت D هشت بیت سمت راست هستند و پورت C هشت بیت سمت چپ. [↑](#footnote-ref-30)
31. Graphical Unit Interface [↑](#footnote-ref-31)
32. البته اگر تعداد فایل­های بیشتری در آینده در بورد تعریف شود، این عدد می­تواند تا 150 افزایش پیدا کند. [↑](#footnote-ref-32)
33. اگر قرار باشد که در آینده شماره طبقات و فایل­های بیشتری به این مجموعه اضافه بشوند، بهتر است فایل­های متناظر با آنها به انتهای این مجموعه افزوده شوند. اضافه کردن تعاریف مناسب در هدرهای IO\_Module.h، تابع ACK\_NACK\_Generation() و سایر عملیات متناظر نیز ضروری می­باشد که توضیح در این موارد در فصل پنج آمده است. در ضمن اگر فایلهایی به مجموعه فوق اضافه شود و تعداد فایل­ها از TOTAL\_MEMORY\_FILES که در سرآیند General.h تعریف شده فراتر برود، لازم است این ثابت اصلاح شود. در شرایط فعلی می­توان تا 150 فایل در بورد تعریف کرد. [↑](#footnote-ref-33)
34. Butterworth [↑](#footnote-ref-34)
35. همانگونه که می­دانیم اعمالی مثل فیلتر کردن و یا حذف تعدادی نمونه پست سر هم از ابتدا و انتهای یک سیگنال باعث تغییر نرخ نمونه­برداری آن نخواهد شد. [↑](#footnote-ref-35)
36. برای درک بهتر چگونگی رخداد این وضعیت به توضیحات در مورد تابع InputSignalDetection() و همینطور توضیحات در مورد وضعیت سکوت مراجعه شود. [↑](#footnote-ref-36)
37. توصیه می­شود برای بررسی علت اینکه چرا سیگنال اورژانسی مجدداً در وضعیت SILENCE+1 بررسی می­شود به توضیحات در مورد تابع InputSignalDetection() رجوع شود. [↑](#footnote-ref-37)
38. یادآوری می­شود تمامی پارامترهایی که با حروف بزرگ نوشته شده­اند ثوابت کد نرم­افزار می­باشند. این ثوابت در ماژول­های مربوطه فهرست شده­اند. [↑](#footnote-ref-38)
39. تعداد این تکرارها در متغیر OVL\_Delay\_Counter ذخیره می­شوند. [↑](#footnote-ref-39)
40. علت اینکه پخش موسیقی به اندازه دو تیک ادامه پیدا می­کند آن است که قطع سیگنال TRG برای لحظاتی کوتاه باعث قطع شدن پخش سیگنال نشود. [↑](#footnote-ref-40)
41. اگر در کد تابع Change\_State() به وضعیت MUS دقت شود مشاهده می­شود که قطع شدن سیگنال ورودی با بررسی شرط  
    CTRL\_Signal = NO\_SIGNAL بررسی نشده است. علت این است که ممکن است به عنوان مثال یک مسافر از زمان حرکت تا هنگام توقف آسانسور در برابر چشمی DOB قرار بگیرد و در نتیجه عملاً بورد هیچگاه وضعیت NO\_SIGNAL را به عنوان وضعیتی که سیگنال TRG قطع شده است تشخیص ندهد. به همین دلیل نبودن سیگنال ورودی تنها با یک else if فاقد شرط بررسی سیگنال کنترلی از حالت وجود سیگنال تغییر آلبوم یا TRG جدا شده است. البته این مورد به صورت کلی در پیاده­سازی و در تمامی وضعیت­ها رعایت شده است. [↑](#footnote-ref-41)
42. همانگونه که در فصل اول اشاره شد، مورد فوق یکی از مهمترین مشکلات بورد FSB200a بوده است که در آن در صورت به وجود آمدن وضعیت فوق، پیام DOB پخش می­شود. [↑](#footnote-ref-42)
43. اگر چه توجه داریم امکان دریافت سیگنال OVL در حین حرکت آسانسور تقریباً ناممکن است، چون عدم وجود این سیگنال قبل از حرکت آسانسور بررسی می­شود. [↑](#footnote-ref-43)
44. با انتخاب فوق، سیگنال شمارنده در کل دوره وضعیت فوق فقط و فقط یکبار خوانده خواهد شد. [↑](#footnote-ref-44)
45. نمودار فوق به عنوان نمودار Bode نیز شناخته می­شود. [↑](#footnote-ref-45)