mPlayer Detailed Design

CSE, HCMUT

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name** | *mPlayer – MP3 Player* | | |
| **Date** | *10/05/2013* | **Version** | *Version – 1.2* |
| **Author** | *Ho Nhut Minh*  *Nguyen Van Hung*  *Huynh Khanh Phong* | **Approver** | *PhD. Vu Tuan Thanh* |
| **Organization** | *Embedded System Class - Faculty of Computer Science and Engineering – HCMC University of Technology* | | |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date:** | **Change Contents** | **Author** | **Approver** |
| *0.1* | *29/04/2013* | *▪Document created.* | *Hung* |  |
| *1.0* | *05/05/2013* | *▪Add state machine and interface* | *Phong* |  |
| *1.1* | *05/05/2013* | *▪Modified Libraries, Add main state machine, add playing flowchart* | *Phong*  *Minh* |  |
| *1.2* | *10/05/2013* | *▪Modified third-party functions, file\_system module, correct grammar and format* | *Hung, Minh, Phong* |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1 Software Architecture 4](#_Toc356691451)

[2 State-Machine model 5](#_Toc356691452)

[Main state machine 5](#_Toc356691453)

[Buttons state machine 7](#_Toc356691454)

[E-book state machine 8](#_Toc356691455)

[Mp3 player state machine 9](#_Toc356691456)

[3 Software Interface 11](#_Toc356691457)

[3.1 Application functions 11](#_Toc356691458)

[a. mplayer.c: 11](#_Toc356691459)

[b. ebook\_lyrics.c: 14](#_Toc356691464)

[c. radio.c: 15](#_Toc356691465)

[d. FileSystem.c 16](#_Toc356691466)

[e. MP3 player function call flowchart 18](#_Toc356691467)

[3.2 Middle ware functions 19](#_Toc356691468)

[a. mp3driver.c: 19](#_Toc356691469)

[b. buttons.c: 20](#_Toc356691470)

[c. httpclient.c: 20](#_Toc356691471)

[d. window.c: 21](#_Toc356691472)

[e. Peripheral.c: 25](#_Toc356691495)

[3.3 Driver and library function 25](#_Toc356691496)

[a. Lwip-1.4.1 25](#_Toc356691497)

[b. FatFs - FAT 29](#_Toc356691512)

[c. Rit128x96x4 driver 32](#_Toc356691519)

[d. Uartstdio - Prototypes for the UART console functions. 34](#_Toc356691520)

[e. GPIO 35](#_Toc356691521)

[f. SSI 37](#_Toc356691522)

[g. System Control 38](#_Toc356691523)

[h. SysTick 39](#_Toc356691524)

[i. Interrupt 39](#_Toc356691525)

[j. Pin Connection: 40](#_Toc356691526)

# Software Architecture

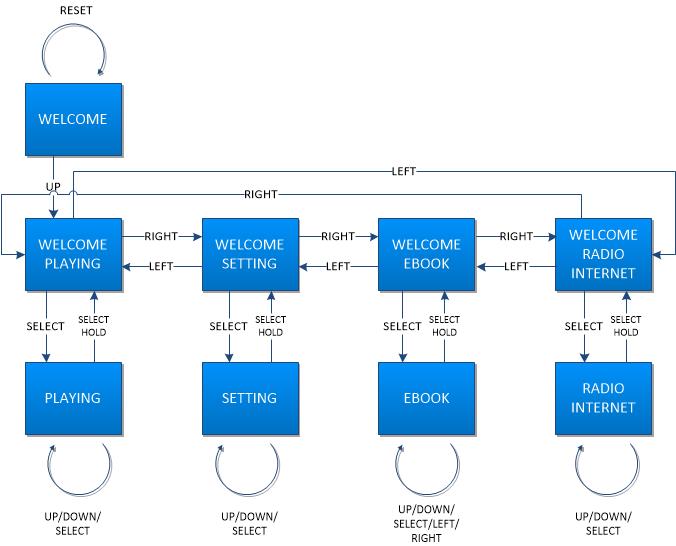


The diagram above is the detailed software architecture of Mplayer project. Some modules are inherited from open source community or Stellaris library; the others need to be developed.

|  |  |
| --- | --- |
| Reused Module | Developed Module |
| SSI | MP3\_Player |
| DiskIO | Internet\_Radio |
| Ethernet | File\_Browser |
| SystemTick | Ebook\_Reader |
| LCD\_Graphic | MP3\_driver |
| UART | HTTP\_Client |
| FAT\_File | Buttons |
| LWIP | Window\_display |

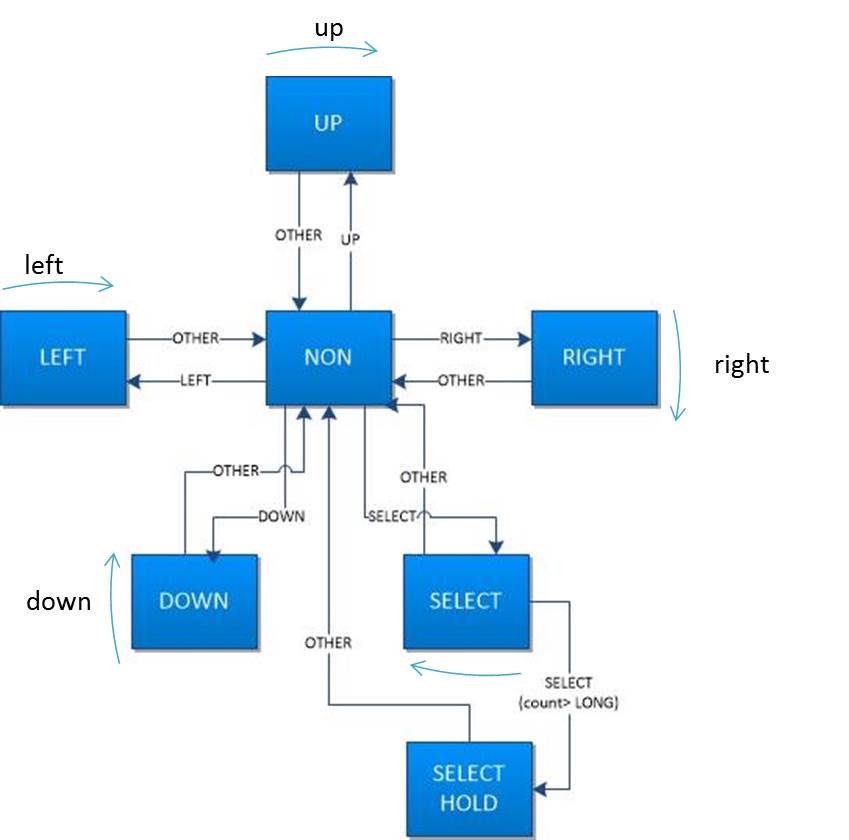
# State-Machine model

## MAIN STATE MACHINE



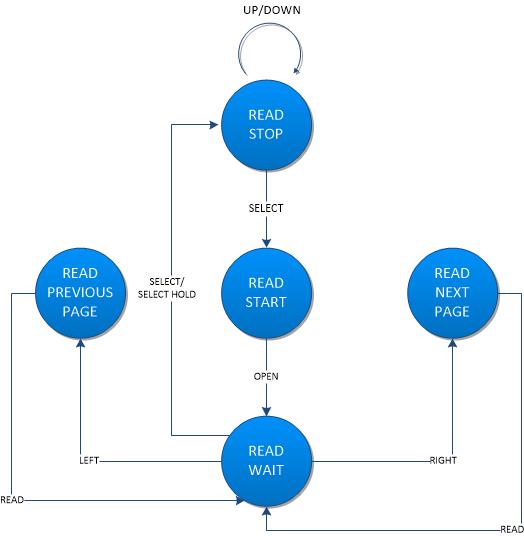
|  |  |  |  |
| --- | --- | --- | --- |
| State | Description | Event | Next state |
| WELCOME | Welcome screen | Reset button’s pressed | WELCOME |
| Up button’s pressed | WELCOME PLAYING |
| WELCOME  PLAYING | Welcome playing screen | Right button’s pressed | WELCOME SETTING |
| Left button’s pressed | WELCOM RADIO |
| Select button’s pressed | PLAYING |
| Select button’s pressed | SONGLIST |
| PLAYING | Playing screen | Select hold state | WELCOME PLAYING |
| SONGLIST | Songlist screen | Select button’s pressed | PLAYING |
| Up/Down button’s pressed | SONGLIST |
| WELCOME  SETTING | Welcome setting screen | Right button’s pressed | WELCOME EBOOK |
| Left button’s pressed | WELCOME PLAYING |
| Select button’s pressed | SETTING |
| SETTING | Setting screen | Select hold state | WELCOME SETTING |
| Up/ Down/ Select | SETTING |
| WELCOME EBOOK | Welcome ebook screen | Right button’s pressed | WELCOME RADIO |
| Left button’s pressed | WELCOME SETTING |
| Select button’s pressed | EBOOK |
| EBOOK | Ebook screen | Select hold state | WELCOME EBOOK |
| Up/ Down/ Select/ Left/ Right | EBOOK |
| WELCOME RADIO | Welcome radio ebook | Right button’s pressed | WELCOME PLAYING |
| Left button’s pressed | WELCOME EBOOK |
| Select button’s pressed | RADIO |
| RADIO | Radio screen | Select button’s pressed | RADIO |
| Select hold state | WELCOME RADIO |

## BUTTON STATE MACHINE



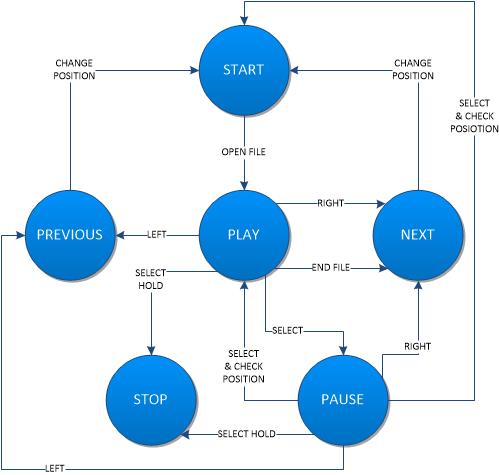
|  |  |  |  |
| --- | --- | --- | --- |
| State | Description | Event | Next state |
| Non | No button is detected | Left button’s pressed | Left |
| Right button’s pressed | Right |
| Up button’s pressed | Up |
| Down button’s pressed | Down |
| Select button’s pressed | Select |
| Left | Button Left is detected | Left button’s pressed  If pressed counts > limit, repeat left button | Left |
| Other button’s pressed | Non |
| Right | Button Right is detected | Right button’s pressed  If pressed counts > limit, repeat right button | Right |
| Other button’s pressed | Non |
| Up | Button Up is detected | Up button’s pressed  If pressed counts > limit, repeat up button | Up |
| Other button’s pressed | Non |
| Down | Button Down is detected | Down button’s pressed  If pressed counts > limit, repeat down button | Down |
| Other button’s pressed | Non |
| Select | Button Select is detected | Select button’s pressed and count >= normal count and count <= hold count | Select |
| Select button’s pressed and count >= hold count | Select Hold |
| Other button’s pressed | Non |
| Select Hold | Button Select ‘s holding | Select button’s pressed  If pressed counts > limit, repeat select and Hold button | Select Hold |
| Other button’s pressed | Non |

## READ EBOOK STATE MACHINE



|  |  |  |  |
| --- | --- | --- | --- |
| State | Description | Event | Next state |
| READ\_WAIT | Wait for next action of user | Left button’s pressed | READ\_PREVIOUS\_PAGE |
| Right button’s pressed | READ\_NEXT\_PAGE |
| Select button’s pressd or Select hold state | READ\_STOP |
| READ\_START | Open file and read fisrt page of txt file, display it | Return success | READ\_WAIT |
| READ\_PREVIOUS\_PAGE | Read previous page of txt file | Return success | READ\_WAIT |
| READ\_NEXT\_PAGE | Read next page of txt file | Return success | READ\_WAIT |
| READ\_STOP | Stop reading and choose another txt file to read | Down or up button’s pressed | READ\_STOP |
| Select button’s pressed | READ\_START |

## MP3 PLAYER STATE MACHINE



This picture is the lite version of the mp3 player state machine. This state machine implements the main function of mplayer. So it’s very complicated. For more details, please refer to page 18.

|  |  |  |  |
| --- | --- | --- | --- |
| State | Description | Event | Next state |
| PLAY | Play music file | Right button’s pressed or  End of music file | NEXT |
| Left button’s pressed | PREVIOUS |
| Select button’s pressed | PAUSE |
| Select hold state | STOP |
| PAUSE | Pause playing music | Up/down button’s pressed | PAUSE |
| Left button’s pressed | PREVIOUS |
| Right button’s pressed | NEXT |
| Select button’s pressed and check position is right | PLAY |
| Select button’s pressed and check position is wrong | START |
| Select hold state | STOP |
| NEXT | Update position in list song | Change position succeed | START |
| PREVIOUS | Update position in list song | Change position succeed | START |
| START | Open music file | Open file succeed | PLAY |

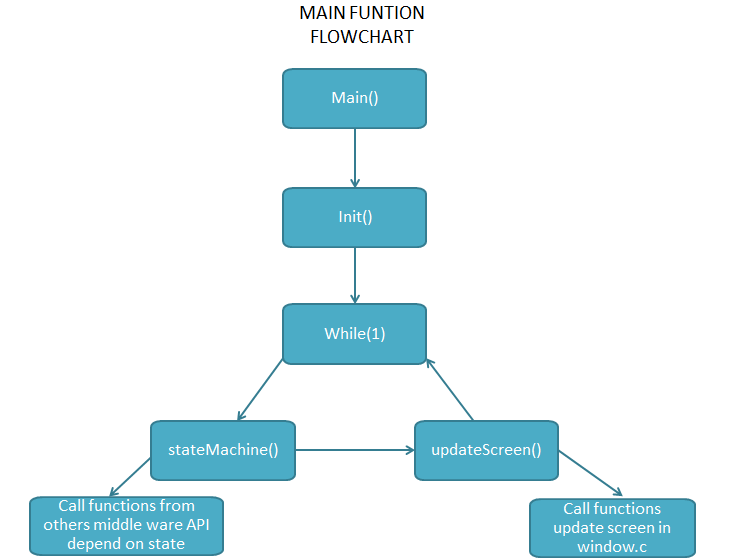
# Software Interface

All related functions will be described in a top-down approach of this project.

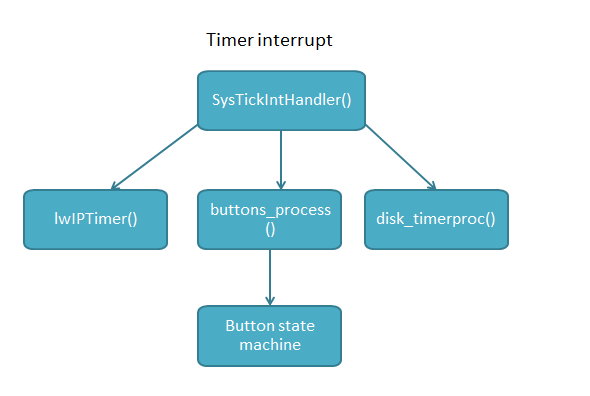
## 3.1 Application functions

### a. mplayer.c:

Main module of project (initial functions, timer interrupt handle, state machine…)

****

This is a general flow chart of main function. stateMachine() and updateScreen() will be described more detail below.

****

Timer interrupt is useful for handle buttons and other frequently actions such as time, Ethernet and disk process. SystemTick requires developer to modify startup\_rvmdk.S, please refer to Stellaris sample project that use SystemTick: SDcard, Ethernet for more information.

**initUART**

**Prototype:** void initUART(void);

**Parameters:** None

**Description**: Initialize UART for debug

**Returns**: None.

**SysTickIntHandler**

**Prototype:** void SysTickIntHandler (void);

**Parameters:** None

**Description**: Method control timer interrupt for button reading, Ethernet timer, DiskIO…

**Returns**: None.

**Init**

**Prototype:** void Init(void);

**Parameters:** None

**Description**: Initialize all needed functions, peripherals: enable and initialize GPIO, SDcard, Ethernet, LCD, VS1011E board,

**Returns**: None.

**stateMachine**

**Prototype:** void stateMachine (void);

**Parameters:** None

**Description**: Main state machine for the application.

**Returns**: None.

**updateScreen**

**Prototype:** void updateScreen (void);

**Parameters:** None

**Description**: Main process to update screen after any change need to display.

**Returns**: None.

**Update Screen Flowchart**:



**main**

**Prototype:** void main (void);

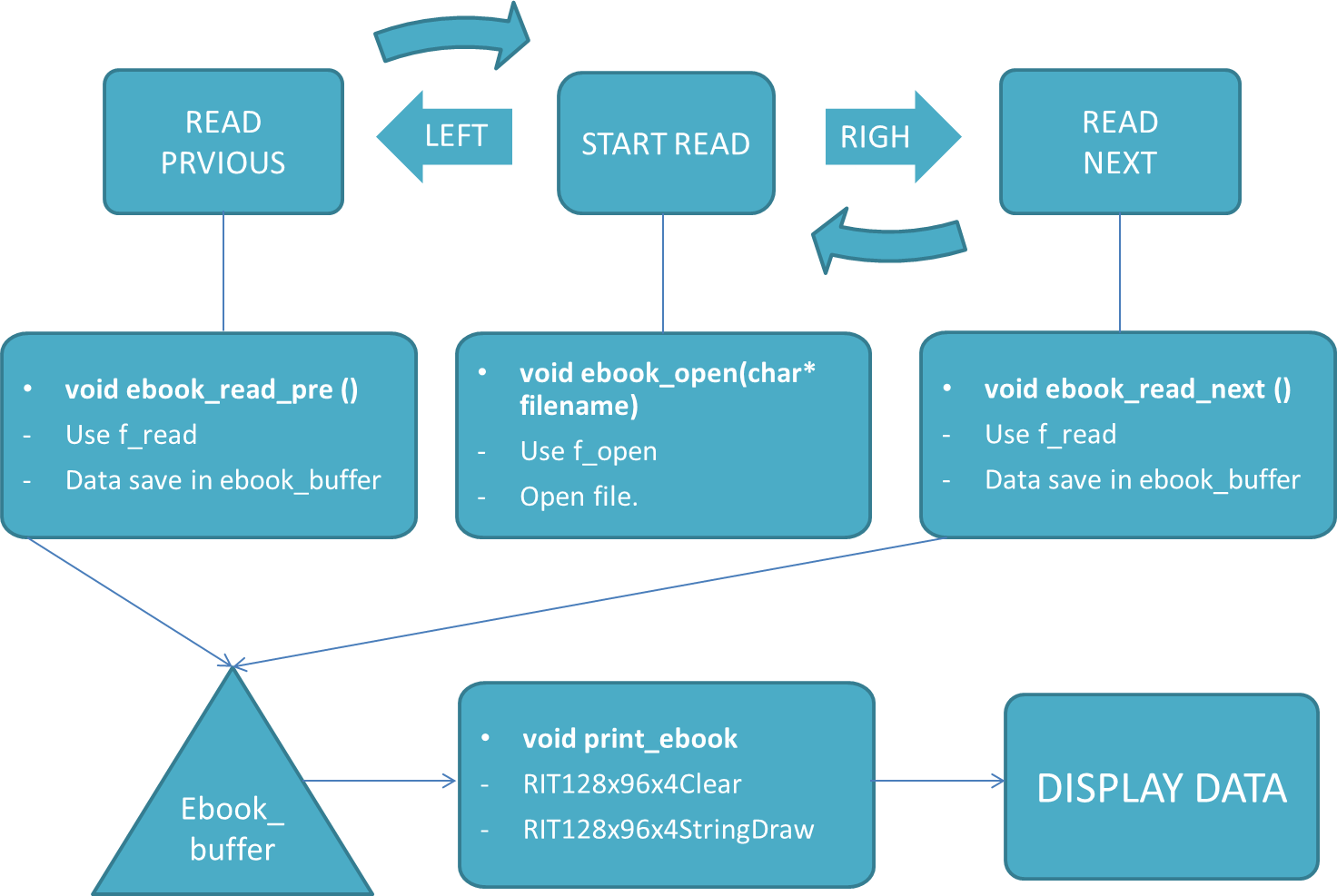
**Parameters:** None

**Description**: Main function of this project.

**Returns**: None.

### b. ebook\_lyrics.c:

**-** Provide function to read txt file and lyric file to display on LCD.

****

**ebook\_open**

**Prototype:** void ebook\_open (char\* filename).

**Parameters:**

- filename: name of ebook file to open.

**Description**:

- use f\_open function to open file with filename and save information of that file in FIL ebook\_file in order to read data after that.

- This function must be called first, before use ebook\_read\_next or ebook\_read\_pre to read the txt file.

**Returns**: None.

**ebook\_read\_next**

**Prototype**: void ebook\_read\_next();

**Description**: this function use to read next page of txt file and save data in ebook\_buffer to print it in OLED.

**Returns**: None.

**ebook\_read\_pre**

**Prototype**: void ebook\_read\_pre();

**Description**: this function use to read previous page of txt file and save data in ebook\_buffer to print it in OLED.

**Returns**: None.

**lyric\_open**

**Prototype:** void lyric\_open (char\* filename).

**Parameters:**

- filename: name of lyric file to open.

**Description**:

- use f\_open function to open file with filename and save information of that file in FIL lyric\_file in order to read data after that.

- This function must be called first, before use lyric\_read\_next to read the lrc file.

**Returns**: None.

**lyric\_read\_next**

**Prototype**: void lyric\_read\_next();

**Description**: this function use to read next line of lrc file and save data in lyric\_buffer to print it in OLED.

**Returns**: None.

### c. radio.c:

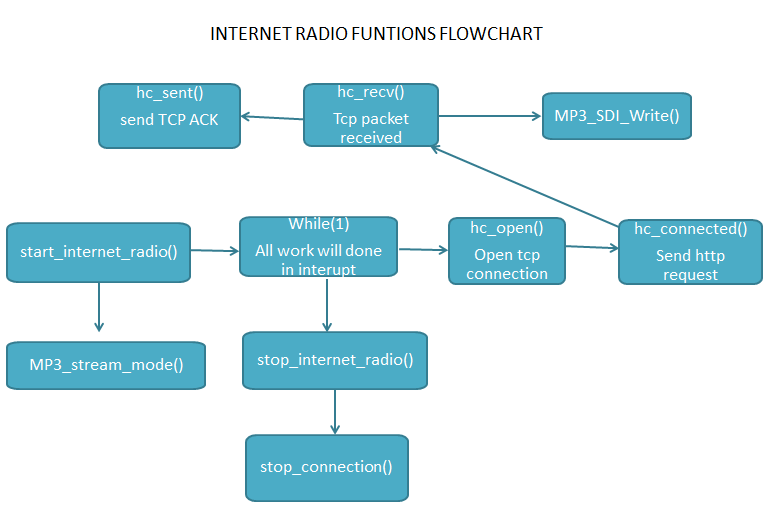
**-** This module streaming music from http server

- The flowchart below describes functions call for this purpose. It contains some function in httpclient.c and mp3driver.c

- httpclient.c: hc\_recv(), hc\_send(), hc\_open(), hc\_connected(), stop\_connection()

- mp3driver.c: MP3\_stream\_mode(), MP3\_SDI\_Write()

- All those functions will be detailed later in this document

****

**init\_Ethernet**

**Prototype:** void init\_Ethernet(void)

**Description**: initial Ethernet and lwip module

**Returns**: None.

**start\_internet\_radio**

**Prototype:** void start\_internet\_radio(void)

**Description**: start radio application

**Returns**: None.

**stop\_internet\_radio**

**Prototype:** void stop\_internet\_radio(void)

**Description**: stop radio application

**Returns**: None.

### d. FileSystem.c

Manager FAT file system in SD CARD, provide some methods to initialize SDCard and File system for use.

**init\_CARD**

**Prototype:** void init\_CARD()

**Description**: init SD card module, mount it to system.

**Returns**: None.

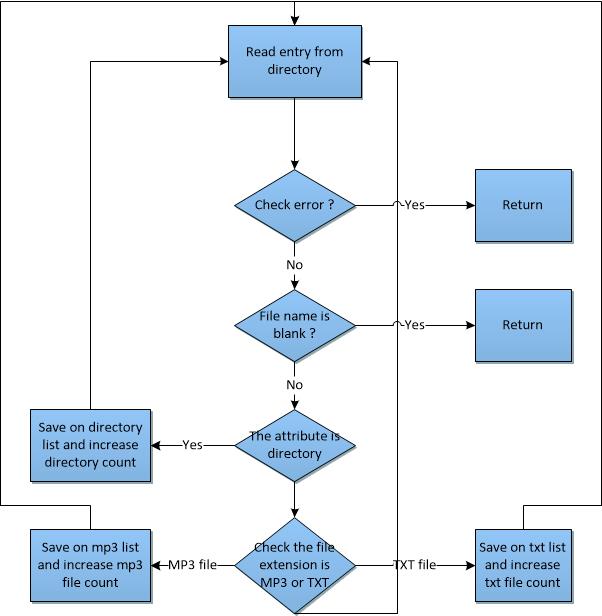
**GetListFileName**

**Prototype:** void GetListFileName()

**Description**: Read all file information in SD\_Card and sort them to mp3\_file\_list or txt\_file\_list, this is a part of middle ware for sdcard, must be called in initial routine.

**Returns**: None.

**Flowchart of GetListFileName**

****

### e. MP3 player function call flowchart



This picture details the function call flow in Playing state of the Main state machine. This flow chart begin when user press Select button in the Welcome Playing screen. That button assert Playing screen and playing state of Main State Machine.

## 3.2 Middle ware functions

All functions below are called from higher layer functions. It is not necessary to draw the flowchart describe middle ware functions. Most of functions are developed by using library functions and the appropriate algorithm.

### a. mp3driver.c:

**-** Provide functions to implement to transfer data between MCU and Mp3 decode module.

- VS1011E board information and development note, include sample source code for 89C51 written in C

<http://www.vlsi.fi/en/products/vs1011.html>

<http://www.vlsi.fi/player_vs1011_1002_1003/modularplayer/vs10xx_8c.html#_details>

-This module uses lots of SSI function, please refer to SSI library that described in Library function later in this document.

**MP3\_SCI\_Write**

**Prototype:** void MP3\_SCI\_Write (char address , unsigned int data)

**Parameters:** register address in codec, data.

**Description**: Function writes one byte to MP3 SCI : serial control interface

**Returns**: None

**MP3\_SCI\_Read**

**Prototype:** void MP3\_SCI\_Read (char start\_address, char words\_count, unsigned int \*data\_buffer)

**Parameters:** start address, word count to be read.

**Description**: Function reads words\_count words from MP3 SCI.

**Returns**: words are stored to data\_buffer.

**MP3\_SDI\_Write**

**Prototype:** void MP3\_SDI\_Write (char)

**Parameters:** data to be write.

**Description**: Function writes one byte to MP3 SDI.

**Returns**: None.

­­­

**MP3\_SDI\_Write\_32**

**Prototype:** void MP3\_SDI\_Write\_32 (char \*)

**Parameters:** data buffer.

**Description**: Function Write 32 bytes to MP3 SDI.

**Returns**: None.

**MP3\_Set\_Volume**

**Prototype:** void MP3\_Set\_Volume (char, char )

**Parameters:** left channel volume, right channel volume.

**Description**: Function set volume on the left and right channel.

**Returns**: None.

**MP3\_reset**

**Prototype:** void MP3\_reset (void)

**Description**: perform hardware reset VS1011E.

**Returns**: None.

**MP3\_pin\_init**

**Prototype:** void MP3\_pin\_init (void)

**Description**: initialize pins for communicate with VS1011E.

**Returns**: None.

**MP3\_Init**

**Prototype:** void MP3\_Init(void)

**Description**: initial VS1011E chip

**Returns**: None.

**MP3\_soft\_reset**

**Prototype:** void MP3\_soft\_reset(unsigned char)

**Parameters:** volume value after reset.

**Description**: perform software reset for VS1011E, and set new volume.

**Returns**: None.

**send\_audio\_data**

**Prototype:** void send\_audio\_data(char \*)

**Parameters:** block 512 bytes read from sdcard.

**Description**: send block of audio data to VS1011E.

**Returns**: None.

**initSSI**

**Prototype:** void initSSI(void)

**Description**: init SPI module for communicate with VS1011E.

**Returns**: None.

**MP3\_stream\_mode**

**Prototype:** void MP3\_stream\_mode(void)

**Description**: set VS1011E run in audio streaming mode

**Returns**: None.

### b. buttons.c:

**-** Provide function to init and determine which button pressed.

**Init\_buttons**

**Prototype:** void init\_buttons(void)

**Description**: initial state of GPIO pins related to buttons

**Returns**: None

**buttons\_process**

**Prototype:** void buttons\_process (void)

**Description**: perform state machine that processes events when user pressed a button

**Returns**: None

### c. httpclient.c:

**-** Provide function to open and stop tcp connection

- Related documents and sample code: <http://www.noyenskenneth.be/mediawiki/index.php?title=LM3S6965_simple_HTTP_webclient_for_RAW_lwip>

**hc\_open**

**Prototype:** int hc\_open(struct ip\_addr remoteIP , long localport)

**Parameter:** struct ip\_addr: remote ip, localport

**Description**: Public function for request a http page from remote ip, use localport

**Returns**: return integer value tells us success or error.

**stop\_connection**

**Prototype:** void stop\_connection(void)

**Description**: close current tcp connetion

**Returns**: None.

### d. window.c:

**-** Provide set of function to control all window of project (welcome window, music player window, ebook window…)

**welcomeWindow**

**Prototype:** void welcomeWindow(void);

**Parameters:**

Global variable:

G\_pucBackground[] : array data for the welcome picture

**Description**: Draw welcome screen

**Returns**: None.

**playingWindow**

**Prototype:** void playingWindow(void);

**Parameters:** None

**Description**: Draw the full playing window.

**Returns**: None.

**settingWindow**

**Prototype:** void settingWindow(void);

**Parameters:** None

**Description**: Draw the setting window.

**Returns**: None.

**ebookWindow**

**Prototype:** void ebookWindow(void);

**Parameters:** None

**Description**: Draw the Ebook reader window.

**Returns**: None.

**radioWindow**

**Prototype:** void radioWindow(void);

**Parameters:** None

**Description**: Draw Radio – Internet window

**Returns**: None.

**updatePlayingHeader**

**Prototype:** void updatePlayingHeader(void);

**Parameters:**

Global variable:

pHeader24[]: Array data for header background.

playing\_state: current playing status (Play, pause, stop,…)

pPlay[] & pPause[] : Array data for play or pause state indicator.

Mp3\_file\_list[] : Array of song’s file name

**Description**: Show the header of playing window: song name, playing status, time, volume indicator.

**Returns**: None.

**updatePlayingBody**

**Prototype:** void updatePlayingBody(void);

**Parameters:**

Global variable:

ulMp3FileCount: number of mp3 files found in SD card

pBody[] : Array data for body background image.

pHighline12[]: Array data for highlighted song background.

songListPos: the current position of the highlighted song in song list.

Mp3\_file\_list[]:Array of song’s file name

**Description**: Update playing body that show the list of song in SD card

**Returns**: None.

**updatePlayingTime**

**Prototype:** void updatePlayingTime(void);

**Parameters:**

Global variable:

minutePlayed & secondePlayed : Minute and second of playing song

**Description**: Show the playing time in the header area.

**Returns**: None.

**updateSettingHeader**

**Prototype:** void updateSettingHeader(void);

**Parameters:**

Global variable:

pHeader24[] : Array data for header background.

**Description**: Draw the setting header

**Returns**: None.

**updateSettingList**

**Prototype:** void updateSettingList(void);

**Parameters:**

Global variable:

settingList[]: Array of setting name list

settingHighline: The position of the current setting highlighted.

**Description**: Draw and update the setting detail.

**Returns**: None.

**updateSettingState**

**Prototype:** void updateSettingState(void);

**Parameters:**

stsRepeat: State variable for repeat function.

stsShuffle: State variable for shuffle function.

**Description**: Update when the user change setting to repeat and shuffle function.

**Returns**: None.

**drawPlayingStateBar**

**Prototype:** void drawPlayingStateBar(char);

**Parameters:**

Percent: the percentage of the song has been played.

**Description**: Draw the playing state bar in header. This method draws 1 piece of the bar only.

**Returns**: None.

**drawPlayingStateBarFromBegining**

**Prototype:** void drawPlayingStateBarFromBegining(char);

**Parameters:** None

**Description**: Draw the entire playing state bar in header.

**Returns**: None.

**clearPlayingStateBar**

**Prototype:** void clearPlayingStateBar(void);

**Parameters:** None

**Description**: Clear the playing state bar

**Returns**: None.

**updateV**o**lumeArea**

**Prototype:** void updateVolumeArea(void);

**Parameters:**

Global variable:

volume: The state of the volume, rank from 0 to 255.

**Description**: Update the volume indicator.

**eturns**: None.

**updateEbookHeader**

**Prototype:** void updateEbookHeader(void);

**Parameters:**

Global variable:

pHeader24[]: Array data for header background.

**Description**: Draw the header for ebook reader window.

**Returns**: None.

**updateRadioHeader**

**Prototype:** void updateRadioHeader(void);

**Parameters:**

Global variable:

pHeader24[]: Array data for header background.

**Description**: Draw the header for radio internet window.

**Returns**: None.

**updateEbookBody**

**Prototype:** void updateEbookBody(void);

**Parameters:**

Global variable:

ulTxtFileCount: Number of .txt file found in SD card.

Txt\_file\_list[] : list of .txt file in SD card.

**Description**: Draw the list of ebook file in SD card.

**Returns**: None.

**print\_ebook**

**Prototype:** void print\_ebook(void);

**Parameters:**

Global variable:

Txt\_file\_list[] : list of .txt file in SD card.

Ebook\_buffer[]: buffer for each row of the displaying text.

**Description**: Draw the content of txt file on screen.

**Returns**: None.

**draw\_welcome\_playing**

**Prototype:** void draw\_welcome\_playing(void);

**Parameters:**

Global variable:

pMusic[] : Array data for playing welcome window.

**Description**: Draw the playing welcome window.

**Returns**: None.

**draw\_welcome\_setting**

**Prototype:** void draw\_welcome\_setting(void);

**Parameters:**

Global variable:

pMusic[] : Array data for setting welcome window.

**Description**: Draw the setting welcome window.

**Returns**: None.

**draw\_welcome\_ebook**

**Prototype:** void draw\_welcome\_ebook(void);

**Parameters:**

Global variable:

pMusic[] : Array data for ebook welcome window.

**Description**: Draw the ebook welcome window.

**Returns**: None.

**draw\_welcome\_radio\_internet**

**Prototype:** void draw\_welcome\_radio\_internet(void);

**Parameters:**

Global variable:

pMusic[] : Array data for radio welcome window.

**Description**: Draw the radio welcome window.

**Returns**: None.

### e. Peripheral.c:

**-** Implement functions to select between SD\_Card, LCD or MP3.

**select\_CARD**

**Prototype:** void select\_CARD (void)

**Description**: enable SDCARD module.

**Returns**: None.

**deselect\_CARD**

**Prototype:** void deselect\_CARD (void)

**Description**: disable SDCARD module.

**Returns**: None.

**select\_LCD**

**Prototype:** void select\_LCD (void)

**Description**: enable LCD module.

**Returns**: None.

**deselect\_LCD**

**Prototype:** void deselect\_LCD (void)

**Description**: disable LCD module.

**Returns**: None.

**select\_MP3**

**Prototype:** void select\_MP3 (void)

**Description**: enable MP3 module.

**Returns**: None.

**deselect\_MP3**

**Prototype:** void deselect\_MP3 (void)

**Description**: disable MP3 module.

## 3.3 Driver and library function

To perform middle ware and application functions, this project must use lower API functions from third party libraries, Stellaris’s libraries, and low-level driver as described below

### a. Lwip-1.4.1

- This module requires user to modify startup\_rvmdk.S. For further reading, refer to sample projects that use **Lwip**, such as: enet\_lwip, enet\_ptpd, webserver …

- For further information, refer to the main page: <http://savannah.nongnu.org/projects/lwip/>

**lwIPTimer**

**Prototype:** void lwIPTimer(unsigned long ulTimeMS)

**Parameters:** ulTimeMS is the incremental time for this periodic interrupt.

**Description:** This function will update the local timer by the value in \e ulTimeMS.If the system is configured for use without an RTOS, an Ethernet interrupt will be triggered to allow the lwIP periodic timers to be serviced in the Ethernet interrupt.

**Returns**: None.

**lwIPInit**

**Prototype:** void lwIPInit(const unsigned char \*pucMAC, unsigned long ulIPAddr,

unsigned long ulNetMask, unsigned long ulGWAddr,

unsigned long ulIPMode)

**Parameters:**

- pucMAC is a pointer to a six byte array containing the MAC address to be used for the interface..

- ulIPAddr is the IP address to be used (static).

- ulNetMask is the network mask to be used (static).

- ulGWAddr is the Gateway address to be used (static).

- ulIPMode is the IP Address Mode. \b IPADDR\_USE\_STATIC will force static IP addressing to be used, \b IPADDR\_USE\_DHCP will force DHCP with fallback to Link Local (Auto IP), while \b IPADDR\_USE\_AUTOIP will force Link Local only.

**Description:** This function performs initialization of the lwIP TCP/IP stack for the Stellaris Ethernet MAC, including DHCP and/or AutoIP, as configured.

**Returns**: None.

**tcp\_close**

**Prototype:** err\_t tcp\_close(struct tcp\_pcb \*pcb)

**Parameters:** pcb the tcp\_pcb to close.

**Description:** Listening pcbs are freed and may not be referenced any more. Connection pcbs are freed if not yet connected and may not be referenced any more. If a connection is established (at least SYN received or in a closing state), the connection is closed, and put in a closing state. The pcb is then automatically freed in tcp\_slowtmr(). It is therefore unsafe to reference it (unless an error is returned).

**Returns**:

- ERR\_OK if connection has been closed.

- another err\_t if closing failed and pcb is not freed.

**tcp\_recved**

**Prototype:** void tcp\_recved(struct tcp\_pcb \*pcb, u16\_t len)

**Parameters:**

**-**  pcb the tcp\_pcb for which data is read.

- len the amount of bytes that have been read by the application.

**Description:** This function should be called by the application when it has processed the data. The purpose is to advertise a larger window when the data has been processed.

**Returns**: None.

**tcp\_recv**

**Prototype:** void tcp\_recv(struct tcp\_pcb \*pcb, tcp\_recv\_fn recv)

**Parameters:**

- pcb tcp\_pcb to set the recv callback

- recv callback function to call for this pcb when data is received.

**Description:** Used to specify the function that should be called when a TCP connection receives data.

**Returns**: None.

**tcp\_err**

**Prototype:** void tcp\_err(struct tcp\_pcb \*pcb, tcp\_err\_fn err)

**Parameters:**

- pcb tcp\_pcb to set the err callback.

- err callback function to call for this pcb when a fatal error has occured on the connection.

**Description:** Used to specify the function that should be called when a fatal error has occured on the connection.

**Returns**: None.

**tcp\_poll**

**Prototype:** void tcp\_poll(struct tcp\_pcb \*pcb, tcp\_poll\_fn poll, u8\_t interval)

**Description:** Used to specify the function that should be called periodically from TCP. The interval is specified in terms of the TCP coarse timer interval, which is called twice a second..

**Returns**: None.

**tcp\_sent**

**Prototype:** void tcp\_sent(struct tcp\_pcb \*pcb, tcp\_sent\_fn sent)

**Parameters:**

- pcb tcp\_pcb to set the sent callback.

- sent callback function to call for this pcb when data is successfully sent.

**Description:** Used to specify the function that should be called when TCP data has been successfully delivered to the remote host.

**Returns**: None.

**tcp\_write**

**Prototype:** err\_t tcp\_write(struct tcp\_pcb \*pcb, const void \*arg, u16\_t len, u8\_t apiflags)

**Parameters:**

- pcb Protocol control block for the TCP connection to enqueue data for.

- arg Pointer to the data to be enqueued for sending.

- len Data length in bytes

- apiflags combination of following flags :

+TCP\_WRITE\_FLAG\_COPY (0x01) data will be copied into memory belonging to the stack.

+TCP\_WRITE\_FLAG\_MORE (0x02) for TCP connection, PSH flag will be set on last segment sent.

**Description:** It waits in the expectation of more data being sent soon (as it can send them more efficiently by combining them together). To prompt the system to send data now, call tcp\_output() after calling tcp\_write().

**Returns**: ERR\_OK if enqueued, another err\_t on error.

**tcp\_output**

**Prototype:** err\_t tcp\_output(struct tcp\_pcb \*pcb)

**Parameters:**

- pcb Protocol control block for the TCP connection to send data.

**Description:** Find out what we can send and send it.

**Returns**: ERR\_OK if data has been sent or nothing to send and another err\_t on error.

**tcp\_new**

**Prototype**: struct tcp\_pcb \*tcp\_new(void)

**Description:** Creates a new TCP protocol control block but doesn't place it on any of the TCP PCB lists. The pcb is not put on any list until binding using tcp\_bind().

- Internal: Maybe there should be a idle TCP PCB list where these PCBs are put on. Port reservation using tcp\_bind() is implemented but allocated pcbs that are not bound can't be killed automatically if wanting to allocate a pcb with higher prio.

**Returns**: a new tcp\_pcb that initially is in state CLOSED.

**tcp\_bind**

**Prototype:** err\_t tcp\_bind(struct tcp\_pcb \*pcb, ip\_addr\_t \*ipaddr, u16\_t port).

**Parameters:**

- pcb the tcp\_pcb to bind (no check is done whether this pcb is already bound!)

- ipaddr the local ip address to bind to (use IP\_ADDR\_ANY to bind to any local address

- port the local port to bind to.

**Description:** Binds the connection to a local portnumber and IP address. If the IP address is not given (i.e., ipaddr == NULL), the IP address of the outgoing network interface is used instead.

**Returns**:

- ERR\_USE if the port is already in use.

- ERR\_VAL if bind failed because the PCB is not in a valid state.

- ERR\_OK if bound.

**tcp\_arg**

**Prototype:** tcp\_arg(struct tcp\_pcb \*pcb, void \*arg)

**Parameters:**

- pcb tcp\_pcb to set the callback argument.

- arg void pointer argument to pass to callback functions.

**Description:** Used to specify the argument that should be passed callback functions.

**Returns**: None.

**tcp\_connect**

**Prototype:** err\_ttcp\_connect(struct tcp\_pcb \*pcb, ip\_addr\_t \*ipaddr, u16\_t port, tcp\_connected\_fn connected)

**Parameters:**

- pcb the tcp\_pcb used to establish the connection.

- ipaddr the remote ip address to connect to.

- port the remote tcp port to connect to.

- connected callback function to call when connected (or on error).

**Description:** Connects to another host. The function given as the "connected" argument will be called when the connection has been established..

**Returns**:

- ERR\_VAL if invalid arguments are given.

- ERR\_OK if connect request has been sent.

- other err\_t values if connect request couldn't be sent.

### b. FatFs - FAT

- File system module R0.09b

- FatFs is a generic FAT file system module for small embedded systems. The FatFs is written in compliance with ANSI C and completely separated from the disk I/O layer. Therefore it is independent of hardware architecture. It can be incorporated into low cost microcontrollers, such as AVR, 8051, PIC, ARM, Z80, 68k and etc..., without any change. The FatFs module is free software opened for education, research and development. You can use, modify and/or redistribute it for personal projects or commercial products without any restriction under your responsibility. For further information, refer to the website: <http://elm-chan.org/fsw/ff/00index_e.html>

**f\_mount**

**Prototype:** FRESULT f\_mount (BYTE, FATFS\*)

**Parameters:**

- Drive: Logical drive number (0-9) to register/unregister the work area. Always 0 for Tiny-FatFs.

- FileSystemObject: Pointer to the work area (file system object) to be registered.

**Description:**

- The f\_mount function registers/unregisters a work area to the FatFs module. The work area must be given to the logical drive with this function before using any file function. To unregister a work area, specify a NULL to the FileSystemObject, and then the work area can be discarded.

- This function only initializes the work area and registers its address to the internal table; any access to the disk I/O layer does not occur. Actual mounting process is performed in any other file functions with path name when it is needed..

**Returns**:

- FR\_OK (0): The function succeeded.

- FR\_INVALID\_DRIVE: The drive number is invalid.

**f\_open**

**Prototype:** FRESULT f\_open (FIL\*, const char\*, BYTE)

**Parameters:**

- FileObject: Pointer to the file object structure to be created. After the f\_open function succeeded, the file can be accessed with the file object structure until it is closed.

- FileName: Pointer to a null-terminated string that specifies the file name to create or open.

- ModeFlags: Specifies the type of access and open method for the file. It is specified by a combination of following flags.

**Description:**

- The created file object is used for subsequent calls to refer to the file. When close an open file object, use f\_close function.

- Before using any file function, work area (file system object) must be given to each logical drive with f\_mount function. All file functions can work after this procedure.

- The mode flags, FA\_WRITE, FA\_CREATE\_ALWAYS, FA\_CREATE\_NEW, FA\_OPEN\_ALWAYS, are not supported in read-only configuration.

**Returns**:

- FR\_OK (0): The function succeeded and the file object is valid.

- FR\_NO\_FILE: Could not find the file.

- FR\_NO\_PATH: Could not find the path.

- FR\_INVALID\_NAME: The file name is invalid.

- FR\_INVALID\_DRIVE: The drive number is invalid.

- FR\_EXIST: The file already exists.

- FR\_DENIED: The required access was denied due to any of following reasons: write mode open of a file that has read-only attribute, file creation under existing a same name directory or read-only file, cannot be created due to the directory table or disk full.

- FR\_NOT\_READYL: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_WRITE\_PROTECTED: Write mode open or creation under the medium is write protected.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_NOT\_ENABLED: The logical drive has no work area.

- FR\_NO\_FILESYSTEM: There is no valid FAT partition on the disk.

**f\_read**

**Prototype:** FRESULT f\_read (FIL\*, void\*, WORD, WORD\*)

**Parameters:**

- FileObject: Pointer to the open file object.

- Buffer: Pointer to the buffer to store read data

- ByteToRead: Number of bytes to read

- ByteRead: Pointer to the WORD variable to return number of bytes read.

**Description:** The file pointer in the file object increases in number of bytes read. The ByteRead will become less than ByteToRead when the read pointer reached to end of the file or any error occurred during the read operation.

**Returns**:

- FR\_OK (0): The function succeeded.

- FR\_DENIED: The function denied due to the file has been opened in write only mode.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_NOT\_READY: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_INVALID\_OBJECT: The file object is invalid..

**f\_lseek**

**Prototype:** FRESULT f\_lseek (FIL\*, DWORD)

**Parameters:**

- FileObject: Pointer to the open file object.

- Offset: Number of bytes where from start of file.

**Description:** The f\_lseek function moves the file R/W pointer of an open file. The offset can be specified in only origin from top of the file. When an offset above the file size is specified in write mode, the file is extended to the offset and the data in the extended area is undefined. After the function succeeded, member fptr in the file object should be checked in order to make sure the R/W pointer has been moved correctly. In case of fptr is less than Offset, any of the followings has been occurred.

• In read-only mode, the Offset was clipped in file size.

• The drive gets full during the file extending process.

• There is any error in the FAT structure.

This function is not supported in minimization level of >= 3..

**Returns**:

- FR\_OK (0): The function succeeded.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_NOT\_READY: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_INVALID\_OBJECT: The file object is invalid.

**f\_close**

**Prototype:** FRESULT f\_close (FIL\*)

**Parameters:**

- FileObject: Pointer to the open file object structure to be closed.

**Description:** The f\_close function closes an open file object. If any data has been written to the file, the cached information of the file is written back to the disk. After the function succeeded, the file object is no longer valid and it can be discarded. If the file object has been opened in read-only mode, it may be discarded without closing process by this function..

**Returns**:

- FR\_OK (0): The file object has been closed successfully.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_NOT\_READY: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_INVALID\_OBJECT: The file object is invalid.

**f\_opendir**

**Prototype:** FRESULT f\_opendir (DIR\*, const char\*)

**Parameters:**

- DirObject: Pointer to the blank directory object to be created.

- DirName: Pinter to the null-terminated string that specifies the directory name to be opened.

**Description:** The f\_opendir function opens an existing directory and creates the directory object for subsequent calls. The directory object structure can be discarded at any time without any procedure. This function is not supported in minimization level of >=2.

**Returns**:

- FR\_OK (0): The function succeeded and the directory object is created. It is used for subsequent calls to read the directory entries.

- FR\_NO\_FILE: Could not find the directory.

- FR\_NO\_PATH: Could not find the path.

- FR\_INVALID\_NAME: The path name is invalid.

- FR\_INVALID\_DRIVE: The drive number is invalid.

- FR\_NOT\_READY: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_NOT\_ENABLED: The logical drive has no work area.

- FR\_NO\_FILESYSTEM: There is no valid FAT partition on the disk.

**f\_readdir**

**Prototype:** FRESULT f\_readdir (DIR\*, FILINFO\*)

**Parameters:**

- DirObject: Pointer to the open directory structure.

- FileInfo: Pointer to the file information structure to store the read item.

**Description:** The f\_readdir function reads directory entries in sequence. All items in the directory can be read by calling f\_readdir function repeatedly. When all directory items have been read and no item to read, the function returns a null string into f\_name[] member without any error. For details of the file information, refer to the FILINFO. This function is not supported in minimization level of >=2.

**Returns**:

- FR\_OK (0): The function succeeded.

- FR\_NOT\_READY: The disk drive cannot work due to no medium in the drive or any other reason.

- FR\_RW\_ERROR: The function failed due to a disk error or an internal error.

- FR\_INVALID\_OBJECT: The directory object is invalid..

### c. Rit128x96x4 driver

- Prototypes for the driver for the RITEK 128x96x4 graphical OLED displays.

- How to use this driver, please refer to sample project in Stellaris folder

**RIT128x96x4Clear**

**Prototype:** void RIT128x96x4Clear (void)

**Description:** This function will clear the display RAM. All pixels in the display will be turned off.

**Returns:** None.

**RIT128x96x4Disable**

**Prototype:** void RIT128x96x4Disable (void)

**Description:** This function initializes the SSI interface to the OLED display.

**Returns:** None.

**RIT128x96x4Enable**

Enable the SSI component of the OLED display driver.

**Prototype:** void RIT128x96x4Enable (unsigned long ulFrequency)

**Parameters:**

**-** ulFrequency specifies the SSI Clock Frequency to be used.

**Description:** This function initializes the SSI interface to the OLED display.

**Returns:** None.

**RIT128x96x4ImageDraw**

**Prototype**: void RIT128x96x4ImageDraw (const unsigned char\*pucImage, unsigned long ulX, unsigned long ulY, unsigned long ulWidth, unsigned long ulHeight)

**Parameters:**

- pucImage is a pointer to the image data.

- ulX is the horizontal position to display this image, specified in columns from the left edge of

the display.

- ulY is the vertical position to display this image, specified in rows from the top of the display.

- ulWidth is the width of the image, specified in columns.

- ulHeight is the height of the image, specified in rows.

**Description**:

- This function will display a bitmap graphic on the display. Because of the format of the display RAM, the starting column (ulX) and the number of columns (ulWidth) must be an integer multiple of two.

- The image data is organized with the first row of image data appearing left to right, followed immediately by the second row of image data. Each byte contains the data for two columns in the current row, with the leftmost column being contained in bits 7:4 and the rightmost column being contained in bits 3:0.

**Returns**: None.

**RIT128x96x4Init**

**Prototype**: void RIT128x96x4Init(unsigned long ulFrequency)

**Parameters**:

- ulFrequency specifies the SSI Clock Frequency to be used.

**Description**: This function initializes the SSI interface to the OLED display and configures the SSD1329 controller on the panel.

**Returns**: None.

**RIT128x96x4StringDraw**

**Prototype**: void RIT128x96x4StringDraw(const char\* pcStr, unsigned long ulX, unsigned long ulY, unsigned char ucLevel)

**Parameters**:

- pcStr is a pointer to the string to display.

- ulX is the horizontal position to display the string, specified in columns from the left edge of the display.

- ulY is the vertical position to display the string, specified in rows from the top edge of the display.

- ucLevel is the 4-bit gray scale value to be used for displayed text.

**Description**:

- This function will draw a string on the display. Only the ASCII characters between 32 (space) and 126 (tilde) are supported; other characters will result in random data being draw on the display (based on whatever appears before/after the font in memory). The font is mono-spaced, so characters such as “i” and “l” have more white space around them than characters such as “m” or “w”.

- If the drawing of the string reaches the right edge of the display, no more characters will be drawn. Therefore, special care is not required to avoid supplying a string that is “too long” to display.

**Note:** Because the OLED display packs 2 pixels of data in a single byte, the parameter ulX must be an even column number (for example, 0, 2, 4, and so on).

**Returns**: None.

### d. Uartstdio - Prototypes for the UART console functions.

Sample project that uses Uartstdio: uart\_echo

**UARTprintf**

**Prototype**: void UARTprintf(const char\* pcString, ...)

**Parameters**:

- pcString is the format string.

- ... are the optional arguments, which depend on the contents of the format string.

**Description**:

- This function is very similar to the C library fprintf() function. All of its output will be sent to the UART. Only the following formatting characters are supported:

* %c to print a character
* %d or %i to print a decimal value
* %s to print a string
* %u to print an unsigned decimal value
* %x to print a hexadecimal value using lower case letters
* %X to print a hexadecimal value using lower case letters (not upper case letters as would typically be used)
* %p to print a pointer as a hexadecimal value
* %% to print out a % character

- For %s, %d, %i, %u, %p, %x, and %X, an optional number may reside between the % and the format character, which specifies the minimum number of characters to use for that value; if preceded by a 0 then the extra characters will be filled with zeros instead of spaces. For example, “%8d” will use eight characters to print the decimal value with spaces added to reach eight; “%08d” will use eight characters as well but will add zeroes instead of spaces. The type of the arguments after pcString must match the requirements of the format string. For example, if an integer was passed where a string was expected, an error of some kind will most likely occur.

**Returns**: None.

**UARTStdioInit**

**Prototype**: void UARTStdioInit(unsigned long ulPortNum)

**Parameters**:

- ulPortNum is the number of UART port to use for the serial console (0-2)

**Description**:

- This function will initialize the specified serial port to be used as a serial console. The serial parameters will be set to 115200, 8-N-1. An application wishing to use a different baud rate may call UARTStdioInitExpClk() instead of this function.

- This function or UARTStdioInitExpClk() must be called prior to using any of the other UART console functions: UARTprintf() or UARTgets(). In order for this function to work correctly, SysCtlClockSet() must be called prior to calling this function.

- It is assumed that the caller has previously configured the relevant UART pins for operation as a UART rather than as GPIOs.

**Returns**: None.

### e. GPIO

- Defines and Macros for GPIO API.

- Sample project: blinky

**GPIOPadConfigGet**

**Prototype**: void GPIOPadConfigGet(unsigned long ulPort, unsigned char ucPin, unsigned long \*pulStrength, unsigned long \*pulPadType)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPin is the pin number.

- pulStrength is a pointer to storage for the output drive strength.

- pulPinType is a pointer to storage for the output drive type.

**Description**: This function gets the pad configuration for a specified pin on the selected GPIO port. The values returned in pulStrength and pulPinType correspond to the values used in GPIOPadConfigSet(). This function also works for pin(s) onfigured as input pin(s); however, the only meaningful data returned is whether the pin is terminated with a pull-up or down resistor.

**Returns**: None.

**GPIOPinRead**

**Prototype**: long GPIOPinRead(unsigned long ulPort, unsigned char ucPins)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

**Description**:

- The values at the specified pin(s) are read, as specified by ucPins. Values are returned forboth input and output pin and the value for pin that are not specified by ucPins are set to 0.

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin tobe accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 epresents GPIO port pin 1, and so on.

**Returns**: Returns a bit-packed byte providing the state of the specified pin, where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on. Any bit that is not specified by ucPins is returned as a 0. Bits 31:8 should be ignored.

**GPIOPinWrite**

**Prototype**: void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

- ucVal is the value to write to the pin(s).

**Description**:

- Writes the corresponding bit values to the output pin(s) specified by ucPins. Writing to a pin configured as an input pin has no effect.

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**: None.

**GPIOPinTypeEthernetLED**

**Prototype**: void GPIOPinTypeEthernetLED (unsigned long ulPort, unsigned char ucPins)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

**Description**:

- The Ethernet peripheral provides two signals that can be used to drive an LED (e.g. for link status/activity). This function provides a typical configuration for the pins.

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**: None.

**Note**: This function cannot be used to turn any pin into an Ethernet LED pin; it only configures an Ethernet LED pin for proper operation. Devices with flexible pin muxing also require a GPIOPinConfigure() function call.

**GPIOPinTypeGPIOInput**

**Prototype**: void GPIOPinTypeGPIOInput (unsigned long ulPort, unsigned char ucPins)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

**Description**:

- The GPIO pins must be properly configured in order to function correctly as GPIO inputs; this is especially true of Fury-class devices where the digital input enable is turned off by default. This function provides the proper configuration for those pin(s).

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**: None.

**GPIOPinTypeGPIOOutput**

**Prototype**: void GPIOPinTypeGPIOOutput (unsigned long ulPort, unsigned char ucPins)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

**Description**:

- The GPIO pins must be properly configured in order to function correctly as GPIO outputs; this is especially true of Fury-class devices where the digital input enable is turned off by default. This function provides the proper configuration for those pin(s).

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**: None.

**GPIOPinTypeUART**

**Prototype**: void GPIOPinTypeUART (unsigned long ulPort, unsigned char ucPins)

**Parameters**:

- ulPort is the base address of the GPIO port.

- ucPins is the bit-packed representation of the pin(s).

**Description**:

- The UART pins must be properly configured for the UART peripheral to function correctly. This function provides a typical configuration for those pin(s); other configurations may work as well depending upon the board setup (for example, using the on-chip pull-ups).

- The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**: None

**Note**: This function cannot be used to turn any pin into a UART pin; it only on figures a UART pin for proper operation. Devices with flexible pin muxing also require a GPIOPinConfigure() function call.

### f. SSI

- Prototypes for the Synchronous Serial Interface Driver.

**SSIBusy**

**Prototype**: tBoolean SSIBusy (unsigned long ulBase)

**Parameters**:

- ulBase is the base address of the SSI port.

**Description**:

- This function allows the caller to determine whether all transmitted bytes have cleared the transmitter hardware. If false is returned, then the transmit FIFO is empty and all bits of the last transmitted word have left the hardware shift register

**Returns**: Returns true if the SSI is transmitting or false if all transmissions are complete.

**SSIDataGet**

**Prototype**: void SSIDataGet (unsigned long ulBase, unsigned long ∗pulData)

**Parameters**:

- ulBase specifies the SSI module base address.

- pulData is a pointer to a storage location for data that was received over the SSI interface.

**Description**:

- This function gets received data from the receive FIFO of the specified SSI module and places that data into the location specified by the pulData parameter. If there is no data available, this function waits until data is received before returning.

**Returns**: None.

**Note**: Only the lower N bits of the value written to pulData contain valid data, where N is the data width as configured by SSIConfigSetExpClk(). For example, if the interface is configured for 8-bit data width, only the lower 8 bits of the value written to pulData contain valid data.

**SSIDataPut**

**Prototype**: void SSIDataPut (unsigned long ulBase, unsigned long ulData)

**Parameters**:

- ulBase specifies the SSI module base address.

- ulData is the data to be transmitted over the SSI interface.

**Description**:

- This function places the supplied data into the transmit FIFO of the specified SSI module. If there is no space available in the transmit FIFO, this function waits until there is space available before returning.

**Returns**: None.

**Note**: The upper 32 - N bits of ulData are discarded by the hardware, where N is the data width as configured by SSIConfigSetExpClk(). For example, if the interface is configured for 8-bit data width, the upper 24 bits of ulData are discarded.

### g. System Control

**SysCtlClockSet**

**Prototype**: void SysCtlClockSet (unsigned long ulConfig)

**Parameters**:

- ulConfig is the required configuration of the device clocking.

**Description**:

- This function configures the clocking of the device. The input crystal frequency, oscillator to be used, use of the PLL, and the system clock divider are all configured with this function.

- Usually for lm3s6965 evaluation boar:

SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_8MHZ);

**Returns**: None.

**Note**: For more informations, read the description of this function in driver/sysctl.h

**SysCtlPeripheralEnable**

**Prototype**: void SysCtlPeripheralEnable (unsigned long ulPeripheral)

**Parameters**:

- ulPeripheral is the peripheral to enable.

**Description**:

- This function enables peripherals. At power-up, all peripherals are disabled; they must be enabled in order to operate or respond to register reads/writes.

- The ulPeripheral parameter must be only one of the following values:

SYSCTL\_PERIPH\_ETH, SYSCTL\_PERIPH\_GPIOA, SYSCTL\_PERIPH\_GPIOB, SYSCTL\_PERIPH\_GPIOC, SYSCTL\_PERIPH\_GPIOD, SYSCTL\_PERIPH\_GPIOE, SYSCTL\_PERIPH\_GPIOF, SYSCTL\_PERIPH\_GPIOG, SYSCTL\_PERIPH\_GPIOH,

SYSCTL\_PERIPH\_SSI0, SYSCTL\_PERIPH\_SSI1, SYSCTL\_PERIPH\_UART0,

**Returns**: None.

**Note**: It takes five clock cycles after the write to enable a peripheral before the the peripheral is actually enabled. During this time, attempts to access the peripheral result in a bus fault. Care should be taken to ensure that the peripheral is not accessed during this brief time period.

### h. SysTick

This module required user to modify startup\_rvmdk.S. For further reading, refer to sample projects that use SysTick, such as: SD card, Ethernet, …

**SysTickEnable**

**Prototype**: void SysTickEnable(void)

**Description**:

- Calling this function will cause the SysTick counter to (re)commence counting from its current value. The counter is not automatically reloaded with the period as specified in a previous call to SysTickPeriodSet(). If an immediate reload is required, the \b NVIC\_ST\_CURRENT register must be written to force this. Any write to this register clears the SysTick counter to 0 and will cause a reload with the supplied period on the next clock.

**Returns**: None.

**SysTickIntEnable**

**Prototype**: void SysTickIntEnable(void)

**Description**: This function will enable the SysTick interrupt, allowing it to be reflected to the processor.

**Note**: The SysTick interrupt handler does not need to clear the SysTick interrupt source as this is done automatically by NVIC when the interrupt handler is called.

**Returns**: None.

### i. Interrupt

**IntMasterEnable**

**Prototype**: tBoolean IntMasterEnable(void)

**Description**: Allows the processor to respond to interrupts. This does not affect the set of interrupts enabled in the interrupt controller; it just gates the single interrupt from the controller to the processor.

**Note**: Previously, this function had no return value. As such, it was possible to include “interrupt.h” and call this function without having included “hw\_types.h”. Now that the return is a “tBoolean”, a compiler error will occur in this case. The solution is to include “hw\_types.h” before including “interrupt.h”.

**Returns**: None.

### j. Pin Connection:

Peripheral devices are connected to LM3S6965 board as follows:

LCD\_CS PIN A3

CARD\_CS PIN D0

MP3\_CS PIN D4

BSYNC PIN D7

MP3\_RST PIN D5

DREQPIN PIN D6

UP PIN E0

DOWN PIN E1

LEFT PIN E2

RIGHT PIN E3

SELECT PIN F1

LED PIN F0