adStar

- *SDK Reference Manual* -

32bits EISC Microprocessor*adStar*

Ver 2.02
January 13, 2015

Advanced Digital Chips Inc.

History

2011-10-31 Created Preliminary Specification

2011-12-20 Ver1.02
- "adStar SDK configuration" and "Using adStar" integrate Using adStar SDK and modified.
- UART ➔ uart_putdata(), uart_putstring() add..
- GRAPHIC ➔ loadtga(), loadtgap(), loadpng(), loadpngp(),
- Drawsetclipwindow(), drawsurfacescale(), drawsurfacescalerect () add.
- FONT ➔ Font function add.

2012-02-21 Ver1.1
- Add Bootloader chapter.
- Add lib_config.h chapter.
- Add Font chapter.
- Add SOUND ➔ sound_vol_wav, sound_loadwavp, sound_loadmp3p function.
- Change company address .

2012-02-29 Ver1.11
- Change image and content at 4.lib_config.h.

2012-03-12 Ver 1.2
- Add GRAPHIC ➔ createsurface_from. (Image rotate)
- Add Font ➔ bitmapfont ➔ bmfont_setautokerning,
- bmfont_write_vleft,
- bmfont_write_vright
- Add Font ➔ bitfont ➔bf_drawstring_vleft, bf_drawstring_vright 추가
- Change Linkter Script content at "2.3 Demo Program Execution".
- Change content at "3.1 Bootloader".
- Add "3.3 Use without BootLoader".

2012-03-21 Ver 1.21
- Change content for use boot.bin file in SD card at "3.1.2 라. User Define Mode".

2012-04-30 Ver 1.3.0
- Add content for UART Interrupt at "5.UART".
- Change "9.5 Sound output channel".
- Change "Export option image" at "11. Font <How to create font image >".
- Add "12. SPI".

2012-08-01 Ver 1.3.1
- Add "8.35 loadsurf ".

2012-09-22 Ver 1.3.2
- Add "11.6 bmfont_gettextwidth".

2013-01-28 Ver 1.3.3
- Change "11.9 bmfont_setautokerning".
- Add "2.5 How to use D16M Series & D8M Series".

2013-08-13	Ver 2.0.0

- Change image and content at “1.Software Development Enviroment”.
- Change “8. Graphic library”.

  setdrawtarget → set_draw_target , getdrawtarget → get_draw_target

  drawline → draw_line, drawrect → draw_rect, drawrectfill → draw_rectfill

  drawround → draw_roundrect, drawroundfill → draw_roundrectfill

  drawcircle → draw_circle, drawcirclefill → draw_circlefill

  drawellipse → draw_ellipse, drawellipsefill → draw_ellipsefill

  drawsurface → draw_surface, drawsurfacerect → draw_surface_rect

  drawsetclipwindow → draw_set_clip_window, releasesurface → release_surface

  drawsurfacescale → draw_surface_scale

  drawsurfacescalerect → draw_surface_scalerect
- Change “11. Font library”.
- Change “2.1 adStar SDK organization and basic definitions”.

2013-12-30	Ver 2.0.1

- Add 2.5 D16M Series & D8M Series chapter.
- Add 13. Debugging.

2015-01-13	Ver 2.0.2

- Add content at 5.11 debugprintf and 5.12 debugstring.

adStarSDK Reference Manual

Office

22F, Bldg A, KeumkangPenterium IT Tower,

810, Gwanyang_dong, Dongan-Gu, Anyang-si, Gyeonggi-do, 431-060, Korea

Tel : +82-31-463-7500

Fax : +82-31-463-7588

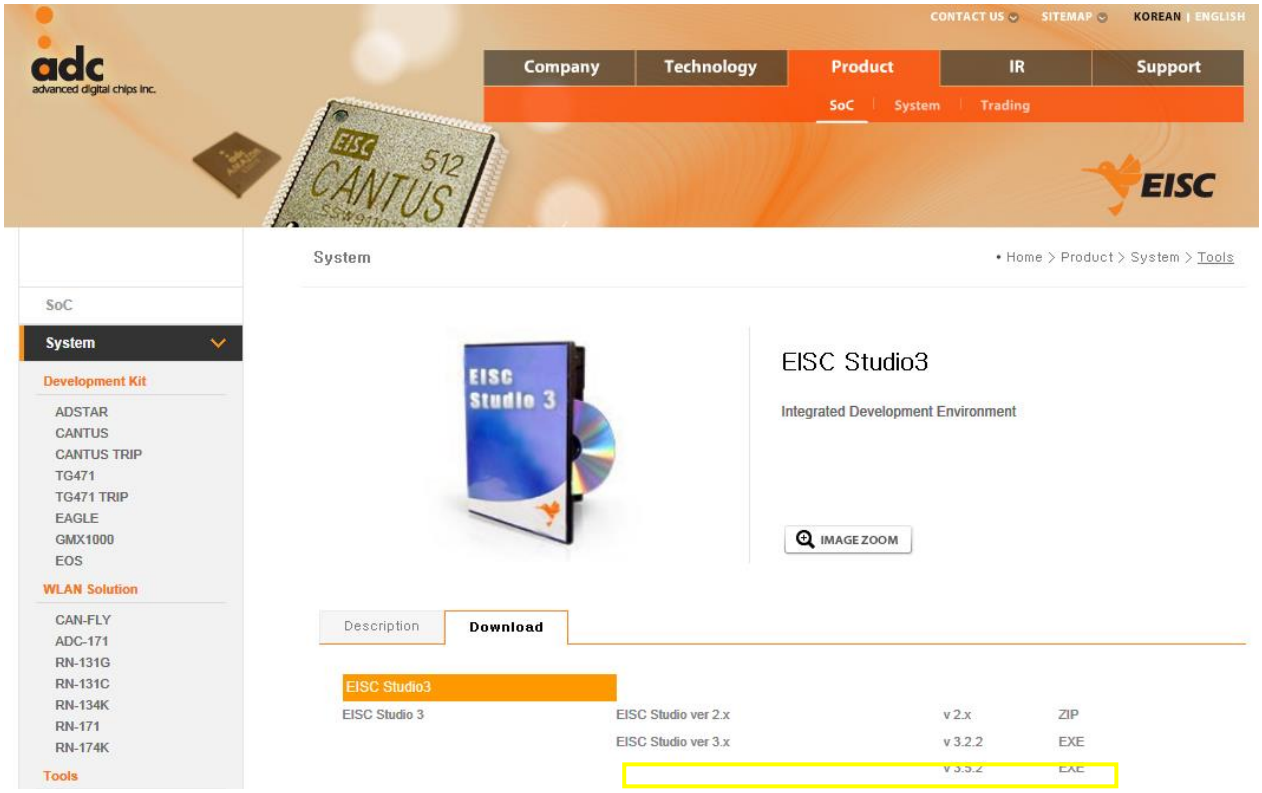
URL : http://www.adc.co.kr

— Table of Contents —

1. SOFTWARE DEVELOPMENT ENVIRONMENT

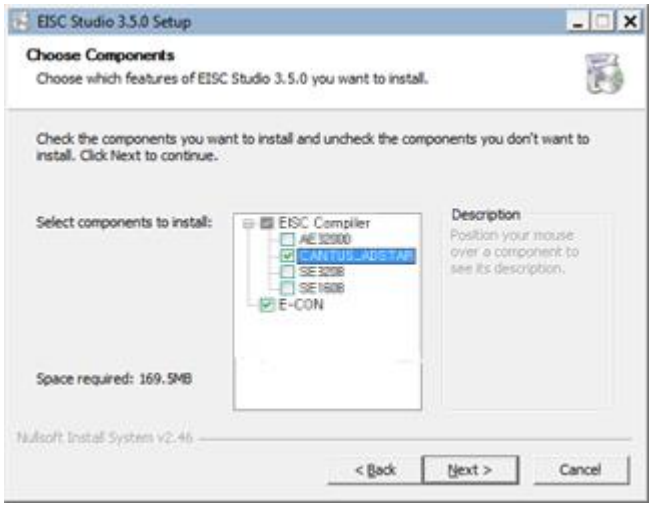
This chapter explains how to establish development environment of adStar based software. ADChips provides IDE called as EISC Studio3 that supports program code management, compilation, downloading and debugging. Therefore, users can set their development environment up easily by installing EISC Studio3 installation file from an ADChips' product downloading page. EISC Studio3 is compatible with Windows XP or higher.

1. Download EISC Studio ver 3.x – v3.6.4 from (http://www.adc.co.kr), Product → System → EISC Studio3 → Download.
   (Refer to an installation and usage guide on the page.)

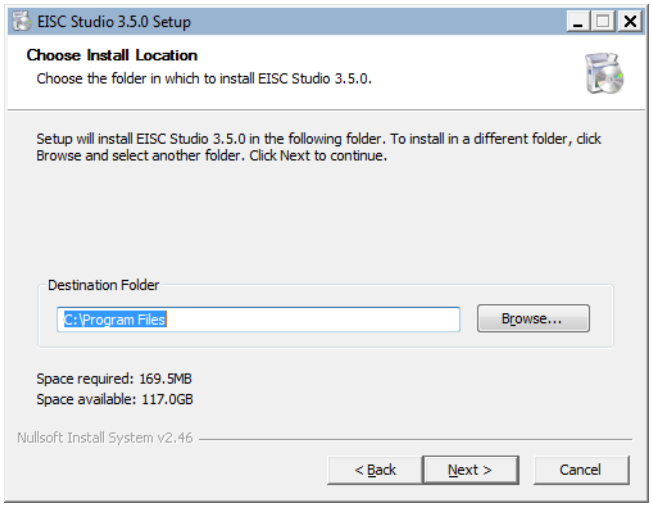


2. After executing downloaded file, ES3_setup_v3.6.4.exe then an installation window will be come up. To continue installation procedures, click 'Next'. Then, users should choose an appropriate E-CON driver and a compiler from a component selection window as below. Since, adStar contains AE32000 processor[1], user should choose CNATUS_ADSTAR of AE32000 compilers. If a user is trying to use E-CON[2] device first time, then install the E-CON Driver by choosing E-CON.

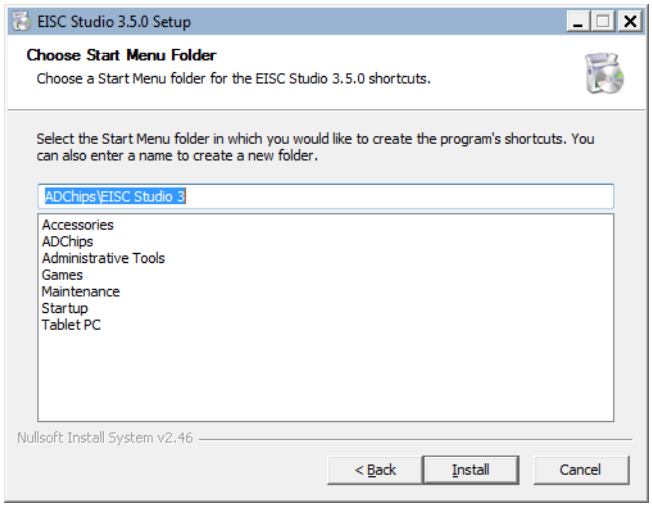
[1]ADChips developed 32bit Processor. (EISC Architecture)
[2]Device for a program download and debugging by connecting with adStar.

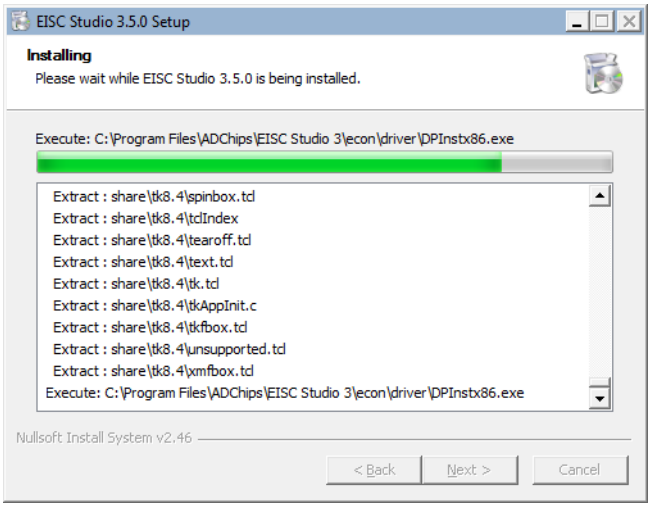


3. Click 'Next' after selecting the compiler and E-CON Driver. Then, user can decide a folder where they will be installed. A default folder name is "C:₩Program Files".
(Caution: when using 64bit OS, the default folder name is "C:₩Program Files(x86)". User must change the default folder name as c:₩Program Files.)

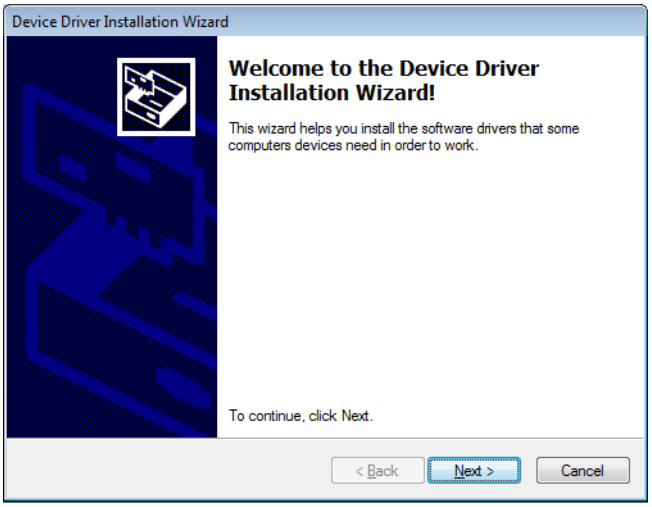


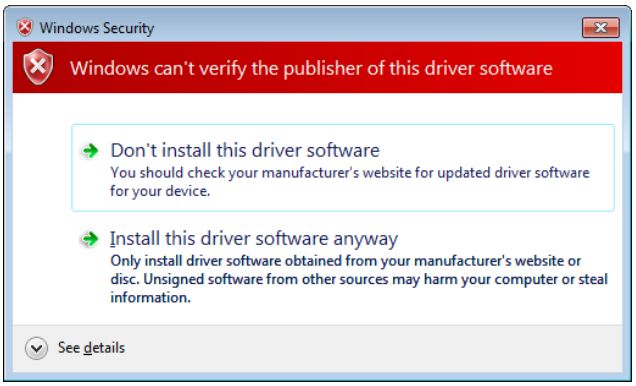
4. Below windows will be come up when click 'Next'. The chosen components are installed when click 'install'.



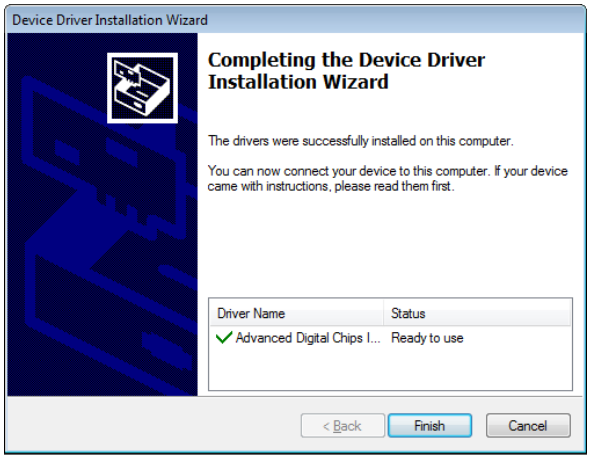


5. If E-CON is selected during the component selection procedure, following window for the E-CON driver installation is displayed. Clicking 'Next', start the driver installation. If Windows alarms secure warning, then choose "install this driver software".
(Caution: E-CON should not be connected with a computer during the installation procedures because could encounter problems.)

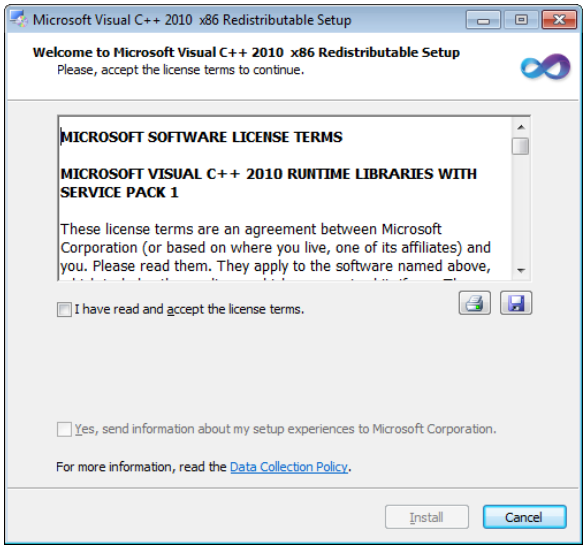




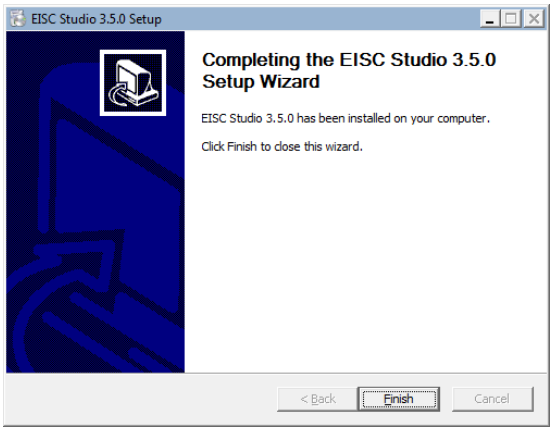
6. The end of the device driver installation loads below window, user can continue remained installation procedures of EISC Studio3 by clicking 'Next'.



7. If Microsoft Visual C++ 2010 Redistribute Package installation window come up, then check 'agree' and continue installation. It is necessary for executing EISC Studio3.

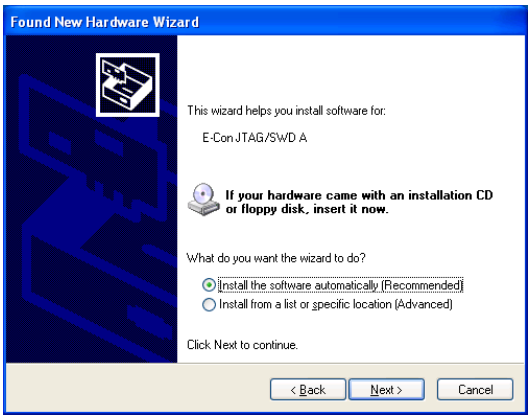


8. After finishing the installation, below window will be displayed. Clicking 'Next', installation will complete.

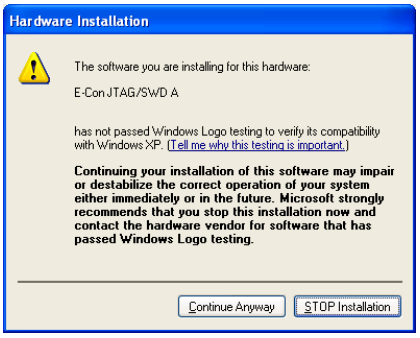


9. After installing EISC Studio 3, a download device; the E-CON driver should be installed. If the E-Con driver files are copied during the EISC Studio 3 installation, then install the E-Con driver by connecting E-Con with a computer. If not, copy the driver files by executing a DPInstx86.exe (32bit) or a DPInstx64 (64bit) in the econ₩driver folder and install the E-Con driver by connecting E-Con with a computer.
(DPInstx86.exe or DPInstx64.exe execution must be done with no connections between E-Con and a computer.)

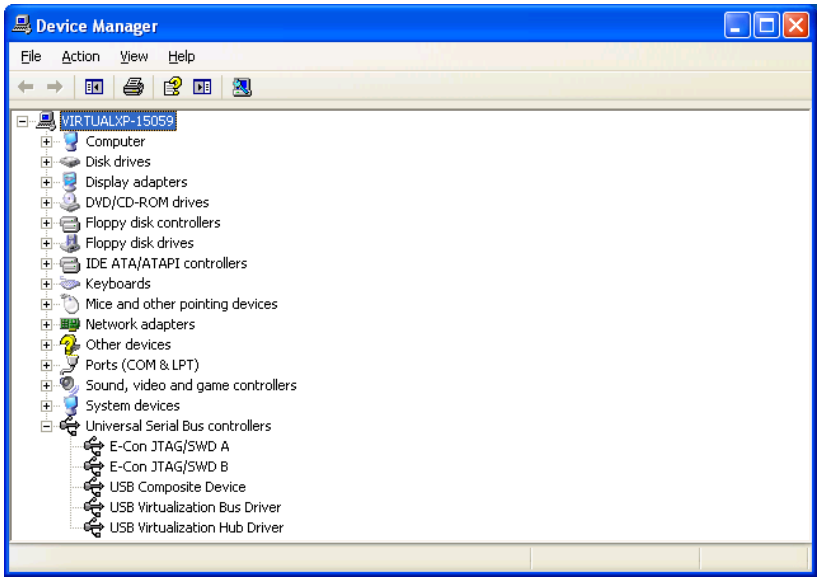
10. The E-Con driver files are copied and E-Con is connected with a computer, then the driver will be installed automatically in case of Windows 7. However, lower than Windows XP loads 'new hardware search wizard' window. Then, just check "install the software automatically" and click 'Next'.



During the installation, below warning messages could be ignored. Just keep clicking 'Next'.



11. E-Con consists of two channels, A and B. The B channel installation is same as what we explained above. User can check the driver installation result from the device manager like below. Refer to (www.adc.co.kr) Product→System→E-CON→Download→"E-CON Driver Install Guide" for more detailed information about the E-Con driver installation.

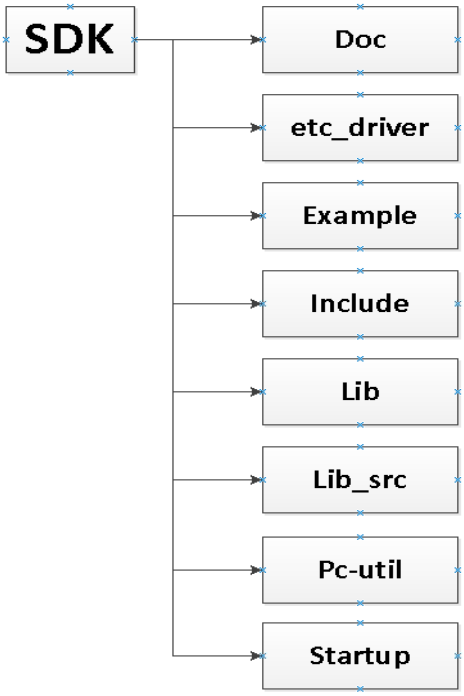

Device Manager
File Action View Help
VIRTUALXP-15059
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Floppy disk controllers
Floppy disk drives
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Ports (COM & LPT)
Sound, video and game controllers
System devices
Universal Serial Bus controllers
E-Con JTAG/SWD A
E-Con JTAG/SWD B
USB Composite Device
USB Virtualization Bus Driver
USB Virtualization Hub Driver

2. Using adStar SDK

ADChips provides adStar SDK for better development environment. adStar SDK can be downloaded from (www.adc.co.kr) Product →SoC→ ADSTAR → Tools & Software.

2.1 adStar SDK organization and basic definitions.

This shows how adStar SDK consists of.



Doc → A folder where adStar related documents (including this) are located.

Example →A foler where adStar STK Board example programs are located.
(Example -> flash_data contains images and sound files for the SDK examples.)

Include →adStar SDK's header files.

lib →adStar SDK's library files.

lib_src→adStar SDK's library source files.
(Originally, libraries are just source files not a built library files. Thus, the libadstar.a library file must be generated in the 'lib' folder by building the adStar.epx project in the lib_src folder. To do this, just press F7 or 'build'-'build project' after opening adStar.epx.)

Pc-util→A folder where utilities to use adStar are located. The adStar USB Driver exists.

Startup ➔all startup & link scrip code for the adStar development. Moreover, STK board basic configuration code exists.

The adStar SDK defines variable types as below for the development convenience.
Refer to below table for referring the SDK Source.

Signed char	S8, s8
Signed short	S16, s16
Signed int	S32, s32
Unsigned char	U8, u8, __u8, BYTE, uchar
Unsigned short	U16, u16, __u16, WORD, ushort
Unsigned int	U32, u32, __u32
Unsigned long	DWROD, ulong
Unsigned long long	U64, u64, __u64
Volatile unsigned char	vU8
Volatile unsigned short	vU16
Volatile unsigned int	vU32
Volatile unsigned long long	vU64
1	TRUE, true
0	FALSE, false

Registers are defined with 'R_' prefix, so 'R_' prefixed variables could be considered as registers in SDK Source.

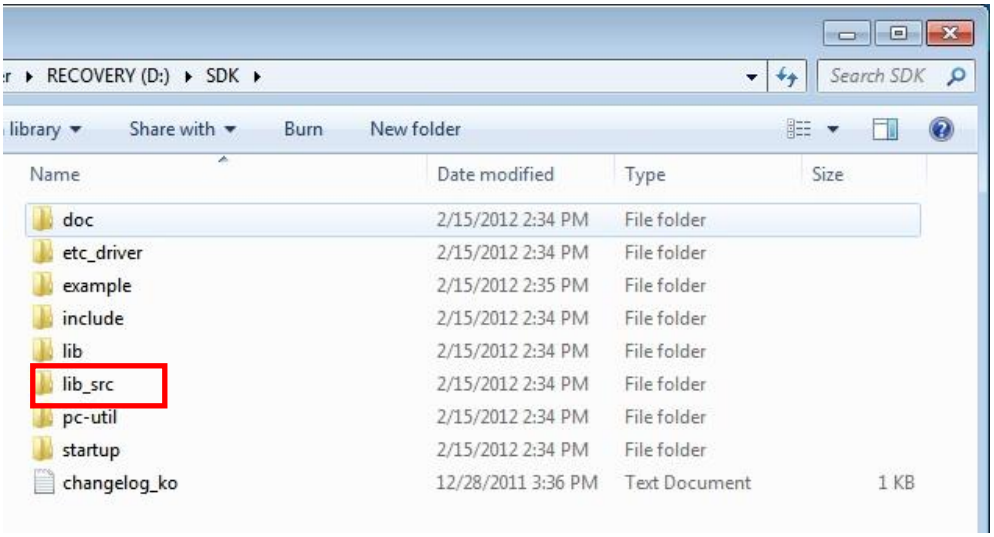
As an example, R_TM0CON is a channel 0 timer control register.

2.2 adStar SDK library build

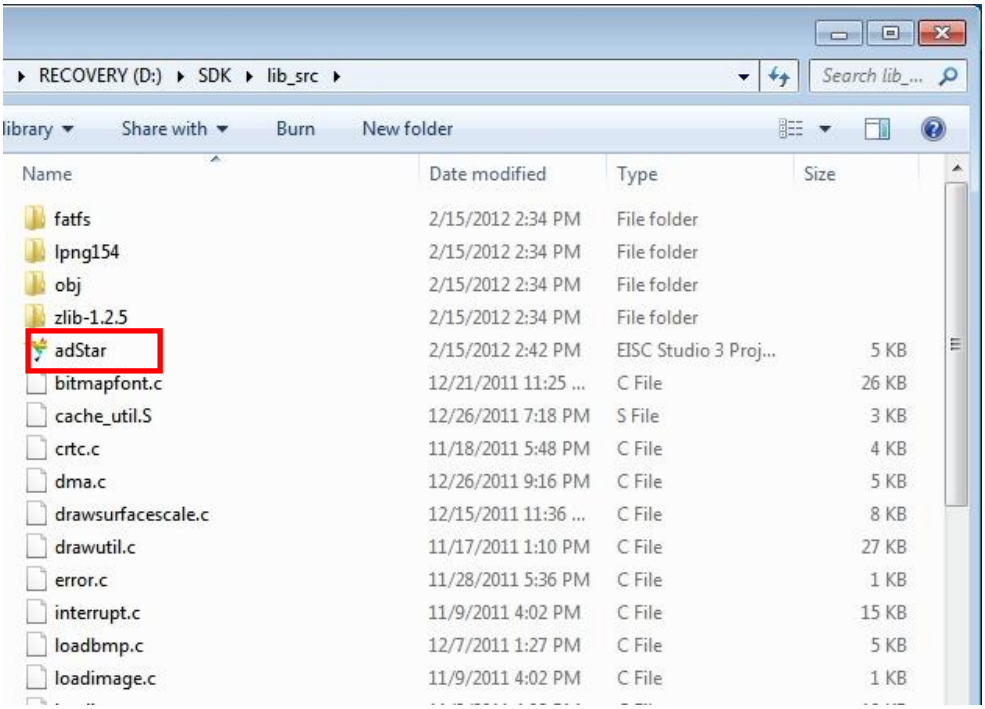
After downloading the adStar SDK, the adStar SDK Library should be built.

In order to build the library, open **'adStar.epx'** file in the **'lib_src'** folder and select 'build'-'build project'(or press F7 for 'project build').

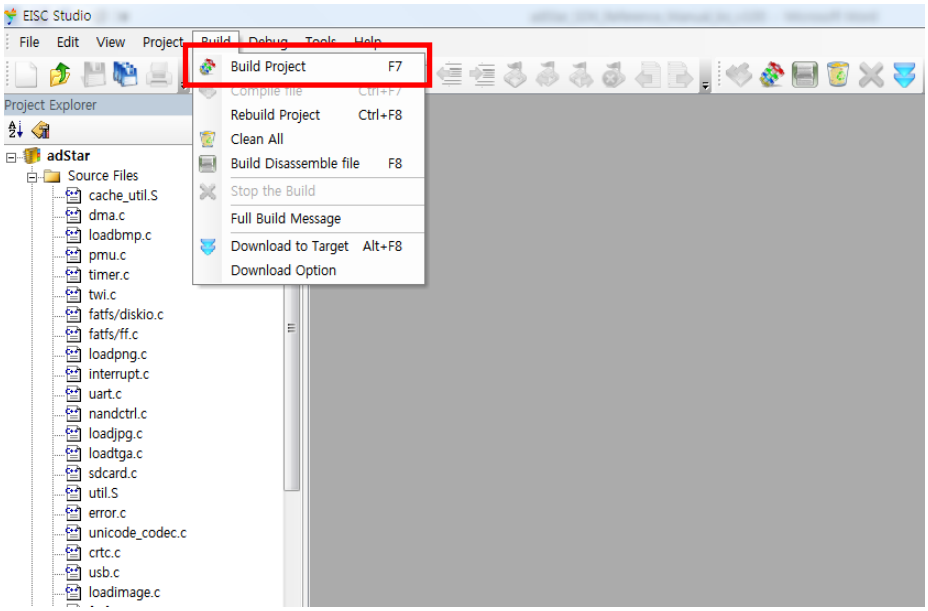
If the process is finished then the **'libadStar.a'** library file will be generated in the **'lib'** folder. (It should be done first because all the adStar SDK examples need the libraries.)



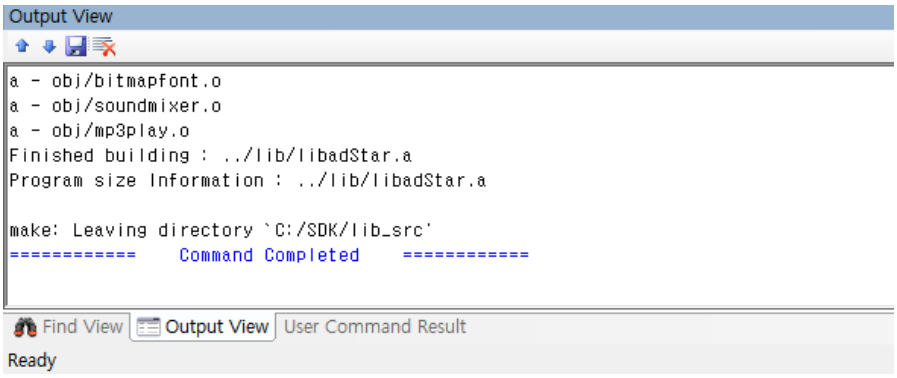
[lib_src folder]



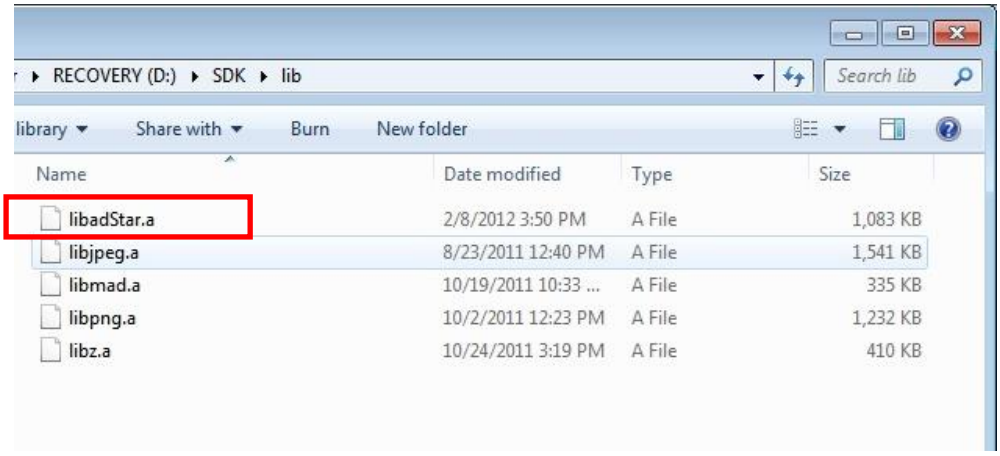
[adStar.epx file]



[Build Project]



[Build Project is finished well]



[libadStar.a file is generated]

If the adStar library is modified, then all the library build processes must be re-executed. In addition, the program that uses the adStar library must be built to adopt the modified library..

2.3 Demo Program Execution

User can easily check the adStar operations on the STK board with various examples in the SDK example folder.

This chapter explains how to use the **Demo** Project Source (of STK examples) such as build, download and execution.

At first, images and sound files Flash should be prepared to execute the demo example on Nand Flash. To copy the file to Nand Flash, execute the **usb_mass_storage** project in the example folder. User need to know how to build and download projects before executing projects.

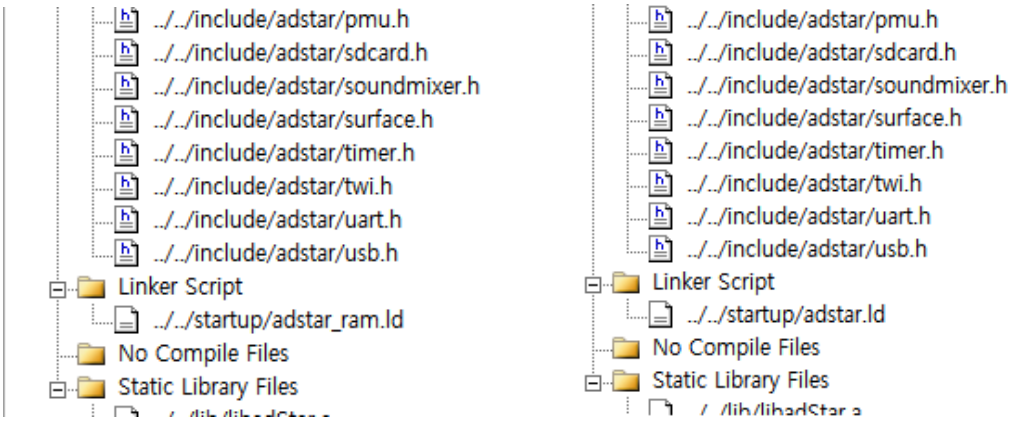
2.3.1 Project build & download

Open a project by double clicking **usb_mass_storage.epx** file in the **example - usb_mass_storage** folder.

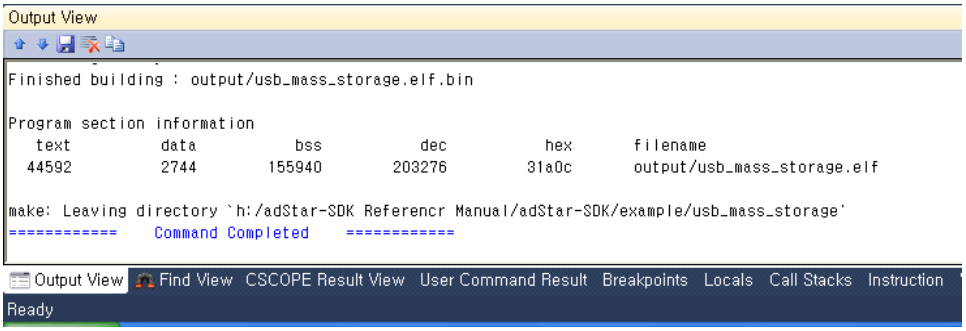
After open the project, change Linker Script file in project Explorer window. Linker Script file is two. one of linker script is adStar.ld file and other one is adStar_ram.ld file.

adStar_ram.ld file (use by default in SDK example) use when operate application in SDRam. So you need bootloader when using adStar_ram.ld file.

At this chapter, explain operate program without bootloader. So use adStar.ld file instead of adStar_ram.ld file. Click mouse right button at Linker Script, click Set Linker Script File and select adstar.ld file in startup folder.

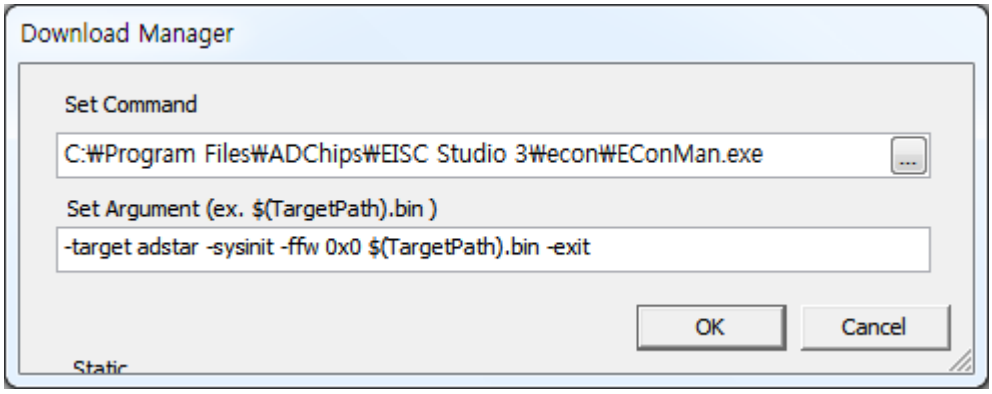


After change Linkger Script, choose **build → Build Project** or press F7 to build the project. Output View shows a build result as below when the build is finished.



When the build is finished well, user can see **usb_mass_storage.elf.bin** file in the output folder. Next, study how to download the built file, **usb_mass_storage.elf.bin** file, to the STK board. Download can be done with E-CON while opening a project to download on EISC Studio3. You follow the below steps.

1. Prepare the adStar board and the E-CON device for downloading.
   (EISC Studio3 and E-CON Driver should be installed. Refer to chapter 1, Software Development Environment for the installation.)
2. Turn on adStar board after connecting the board and the E-CON device.
3. Click 'Build' – 'Download Option'. Then the Download Manager window will come up as below. Set 'Set Command' as the E-CON download program, EConMan.exe that exists in EISC Studio3 installed folder – econ folder. (Can use the default value 'EconMan.exe'.) Set 'Set Argument' as "–target adstar –sysinit –ffw 0x0 $(TargetPath).bin -exit" and click 'OK').



Argument indicates,

-target is target name to download, adStar.

-sysinit is initialization command for download.

-ffw is command to download the build generated bin file, 0x0 is download address and $(TargetPaht).bin is download file name. $(TargetPath).bin indicates a bin file of the currently opened project.

-exit is command to exit automatically after the download is finished.

(Refer to www.adc.co.kr→ Product → System → E-CON → Download → E-CON.pdf file for detailed information of 'Set arguments')

4. Click, 'Build' – 'Download to Target'. Start the download.

2.3.2 Copying Files to Nand Flash (SD Card)

After the downloading, turn the STK board power off and connect an usb cable to a computer and the STK board's device port. Turn the board power on, then new storage device is appeared. This device is Nand Flash of the board, so copying files to the device indicates that the files could be used by the adStar programs.

In order to operate the Demo Project, should copy all image and sound files from 'example'-'flash data'.

To copy the files to a SD card instead of Nand Flash, use a definition in **mass_stor.c** of the **usb_mass_storage** project, '#define SDCARD_STORAGE'. In order to use the definition, remove comment flags on it, re-build and download the project to the board.

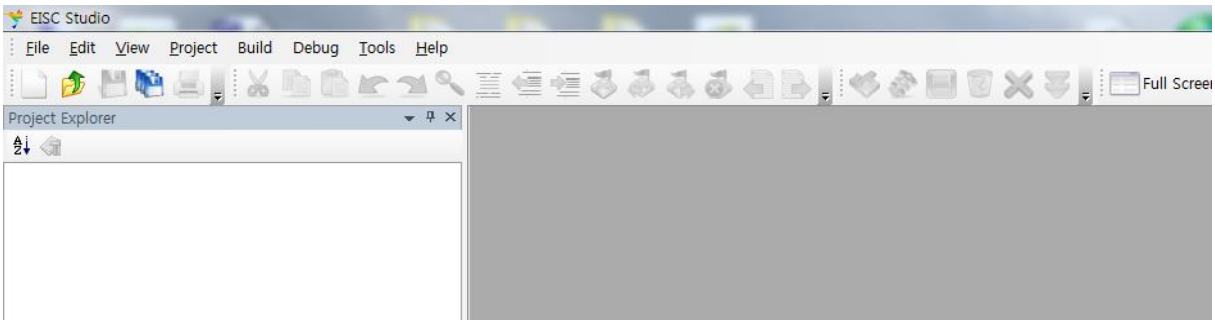
2.3.3 Demo Program build & download

After copying the image and sound files that will be used for the Demo project to Nand Flash, build the Demo project in the example folder. The built project could be downloaded via E-Con and can check the project operations.

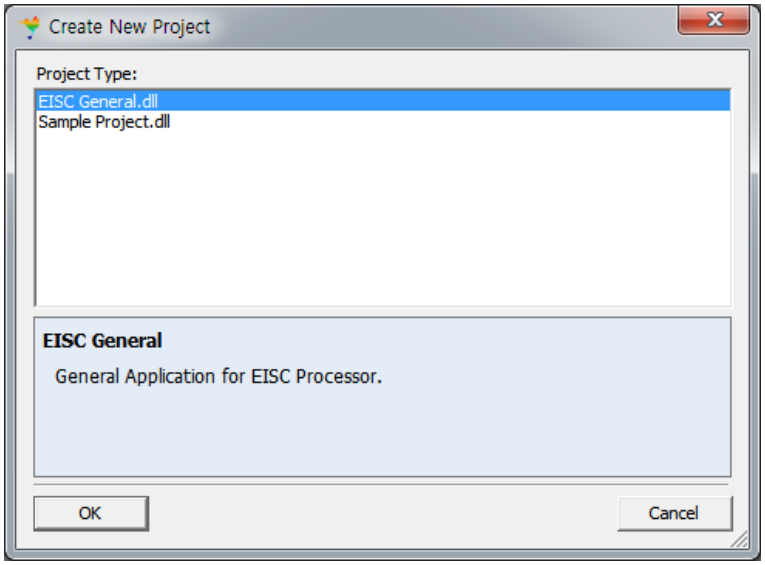
2.4 New adStar Project Creation

This chapter explains how to create a new project that displays "Hello adchips!" statement to UART.

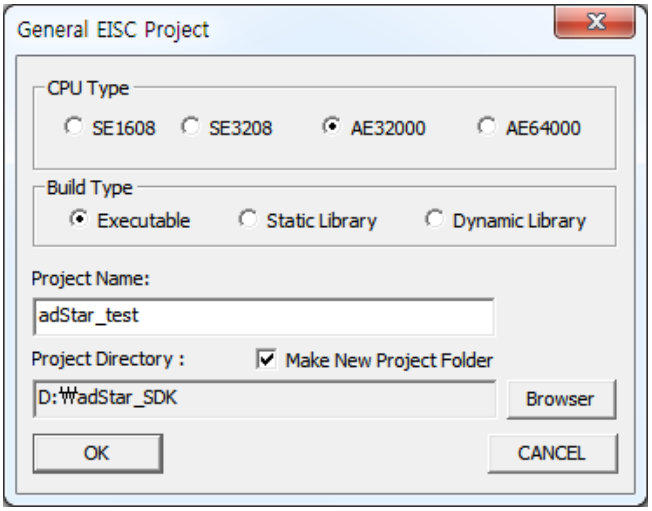
1. Execute EISC Studio3.



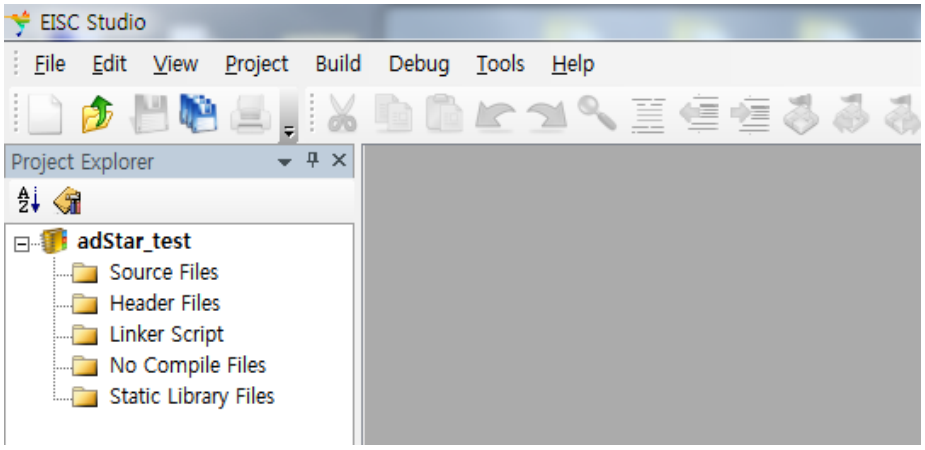
2. Click Menu, File → New → Project, then below window shows up. Then choose EISC General.dll as below and click 'OK'.



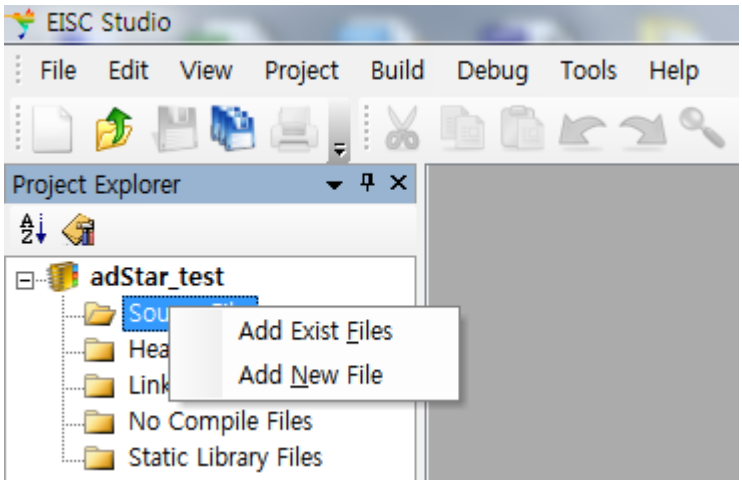
3. Then, a project configuration window shows up like below. User need to determine a CPU Type, a Build Type, a Project Name and a Project Directory. In order to use adStar, the CPU Type: AE32000, Build Type: Executable (because we will make an executable project). After that, we can confirm the Project Name and the Project directory. A check box (Make New Project Folder) besides the Project Directory determines that create a folder which has same name as the project or not. Checking on the box create a folder that has same name as the project in the Project Directory and create a Project file in it. Unless, create a Project file in the directory assigned as the Project Directory. After all the configurations are set, finish the new project creation process by clicking 'OK'.



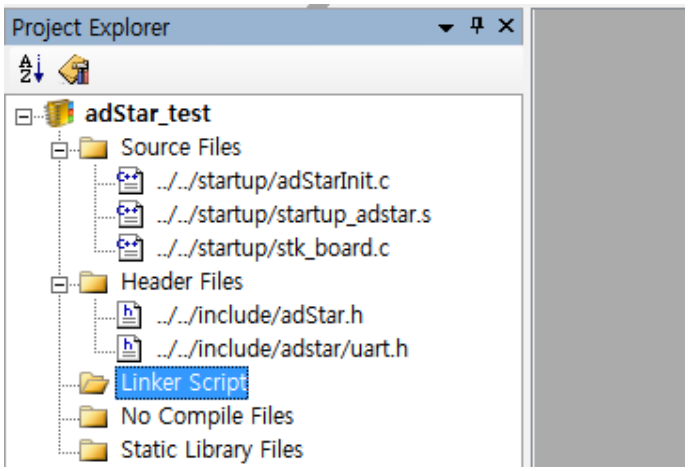
4. Can check the project name and a project tree from a Project Explorer window after the new project creation. Need to add essential files for the adStar project.



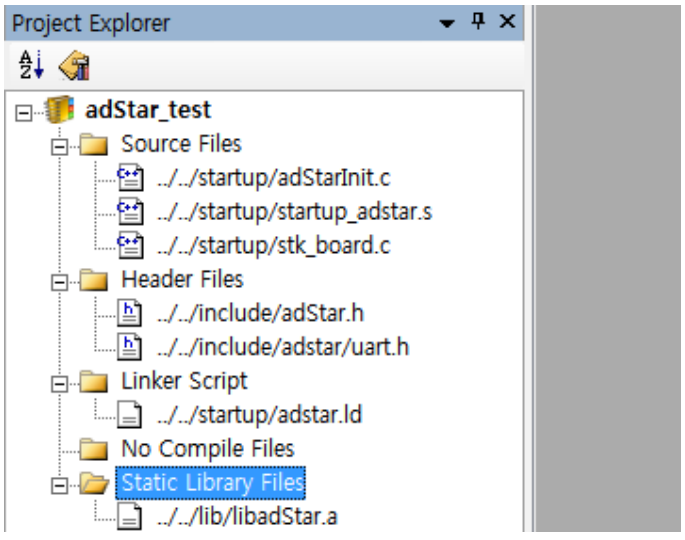
5. At first, click a right mouse button on 'Source Files' and choose 'Add Exist Files' to add the files. The additional files are **startup_adstart.s**, **adStarinit.c** and **stk_board.c** in SDK-'**startup**' folder. (add **stk_board.c** when using the stk board for adStar.)



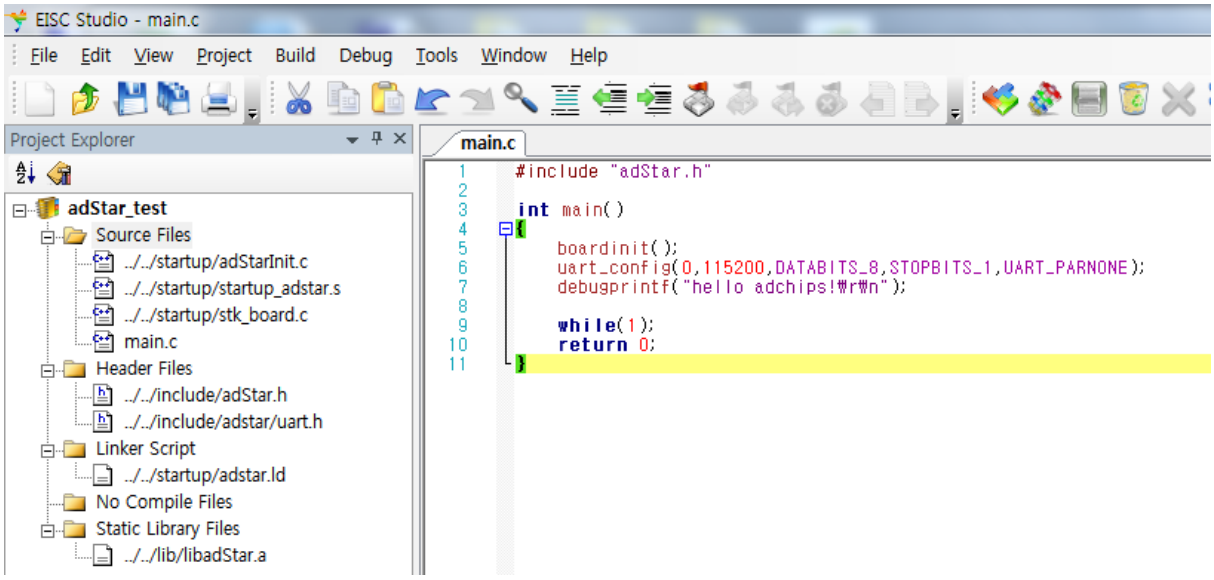
6. After that, same as the 'Source Files', click a right mouse buttion on 'Header Files' and add **adStar.h** file in the **include** folder. It is enough to add **adStar.h** file, but add other related header files in **include/adStar** folder for more easier function usage. We add **UART.h** file because we will make a UART using program.



7. Add **adstar.ld** file in the **startup** folder to the 'Linker Script' and add **libadStar.a** file in the **lib** folder to 'Static Library Files'. Also, add **libmad.a, libjpeg.a, libz.a** and **libpng.a** files for MP3 playing and JPG, PNG image file displaying (these are not necessary for currently making UART exmaple).
   (I'ts OK to add all library files in the lib folder because only needed library files are used.)

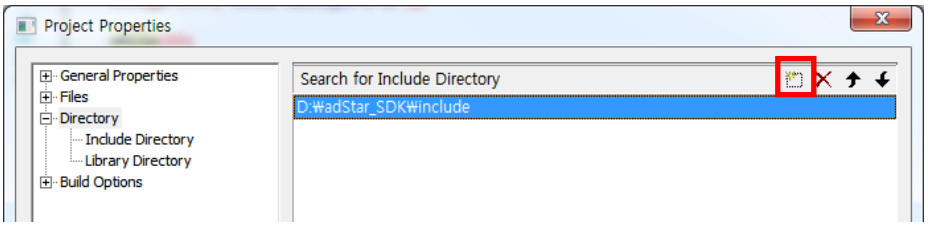


8. If the file addition processes are done as above, create main.c file for a main program by clicking a right mouse button on 'Source Files' and choose 'add new file'. We will explain how to make a code that displays "Hello adchips!" statement via UART.

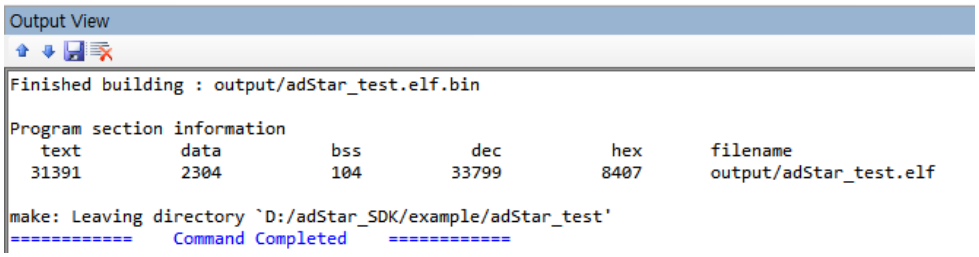


9. In main.c, first thing to do is that set a pin for an appropriate board by calling boardinit() function. (Since, boardinit() function is different with the using board, so set pin according to the board). Next, call uart_config() to initialize UART with denoted parameters.
   Finally, display "hello adchips!" by using debugprintf() function.

10. With all above processes, configure the project before the project build. Click right mouse button on the project name on the Project Explorer window and click 'Properties'. Add the include folder of SDK to 'Directory'-'Include Directory' as below. (Can add a folder by clicking the doted box on top-left)



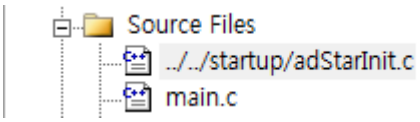
11. After including the include folder, click 'Build'-'Build Project' or press F7 to start the project build. If the building is O.K. then the Output View window shows below results and '**Project name.elf.bin**' file is created in the output folder.



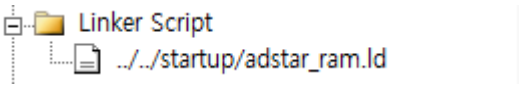
12. The created bin file could be downloaded to the STK board via E-Con. User can check that "hello adchips!" statement is displayed through UART channel 0.

2.5 D16M Series & D8M Series

adStar depending on the SDRam size is divided into D16M and D8M.
SDK example was set as D16M configure by default.
So, If you use D8M, you need to modify project.
At first, select adStarInit_8M.c file instead of adStarInit.c file. (in startup folder)



Next, select adStar_8M.ld or adStar_ram_8M.ld instead of adStar.ld or adStar_ram.ld. (in startup folder)



※ Caution : 1. Demo and Demo_select are not operating at D8M series.

2. Need to use adstar8m as target name when downloading program.
ex) –target adstar8m -sysinit -ffw 0x0 …..

3. Bootloader

adStar can Flash booting and Nand Booting.

So, bootloader is supported for Flash booting and Nand Booting.

At this chapter, explain the adStar bootloader.

3.1Bootloader. (example / bootloader)

Bootloader is booting program for Serial Flash.

Bootloader provide the following functions.

a. Remote Communication Mode.
b. Mass Storage Mode
c. Execute Mode
d. User Define Mode

Remote Communication Mode is that communication with adstar using RemoteManCLI program. Use this mode when download and run application program at DRAM using USB.

Mass Storage Mode is that NAND flash is used as storage device like usb memory at computer. Execute Mode is that copy application program from 0x14000 in flash to SDRam and run that. User Define Mode is copying boot.bin file (application binary file) from NAND Flash to SDRam and run that.

3.1.1 Use Bootloader

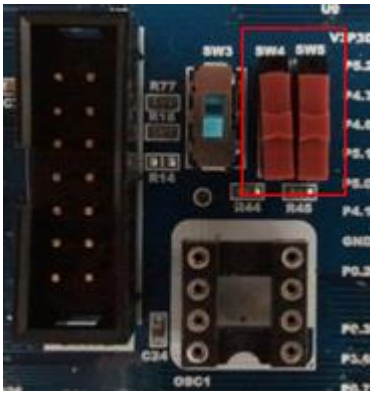
Bootloader is in SDK / example folder.

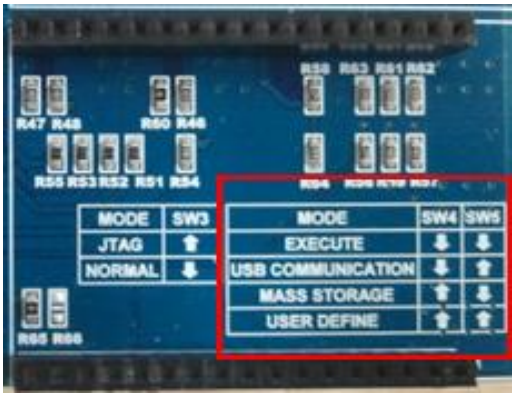
Bootloader project build and download. Then you can see that operate bootloader.

Bootloader has some mode. Following code is setting mode part.

```
67      *R_GPIDIR(0)=0xc;//mode switch
68      int mode = ((*R_GPILEV(0))>>2)&3;
69
70      switch(mode)
71      {
72      case 0: //execute mode
73          bin_execute();
74          break;
75      case 1:
76          usb_clock_init();
77          mass_storage_main();
78          break;
79      case 2:
80          RSP_Run();
81          break;
82      case 3:
83          debugstring("User Define Mode \r\n");
84          bin_execute_fat();
85          break;
86      }
```

In code, select mode by P0.2 (SW4) and P0.3 (SW5) input.





Like above picture, boot mode SW setting value is like following table.

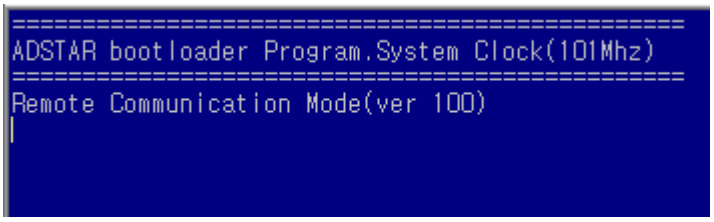
MODE	SW4	SW5
Execute Mode	down	down
USB Communication Mode	down	up
Mass Storage Mode	up	down
User Define Mode	up	up

3.1.2 Bootloader Mode

A. Remote Communication Mode

Remote Communication Mode is that communication with adstar using RemoteManCLI program. Use this mode when download and run application program at DRAM using USB.

Booting Remote Communication Mode then display debug message like following.



[Remote Communication Mode]

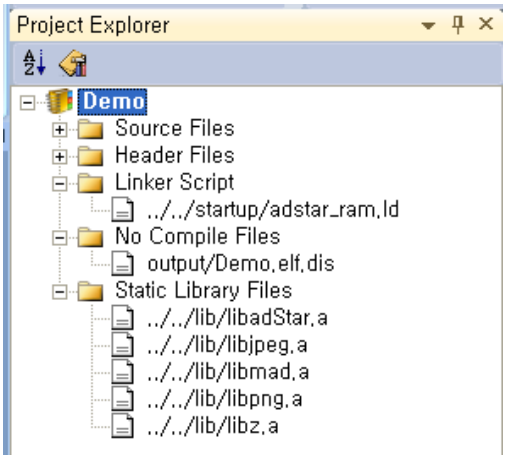
Because using adStar USB device at Remote Communication Mode, need to install adStar USB Driver. USB driver is in pc-util/usb_driver folder. Install driver way is two.

First way is to use DPInstx86.exe (DPInstx64.exe). Second way is to direct install at device manager. This manual is recommended first way. By using first way, install USB driver is like following.

1. Run the DPInstx86.exe (DPInstx64.ext) in usb_driver folder.
2. After complete installation, connect USB cable and booting as Remote Communication Mode.
3. Then 'Installing device driver software' will be shown in tray icon. If you use windows 7, installed it automatically. but if you windws xp, displayed 'Found New Hardware Wizard' window. Then click 'Next' after selecting 'install the software automatically'.
4. After USB driver installation, you can see usb driver in device manager.

[Device Manager]

Now, you can run application program by download to SDRam in Remote Communication Mode. Before downloading, open the application project and check link script.



[Linker Script]

If Linker Script is adstar_ram.ld, then run build.

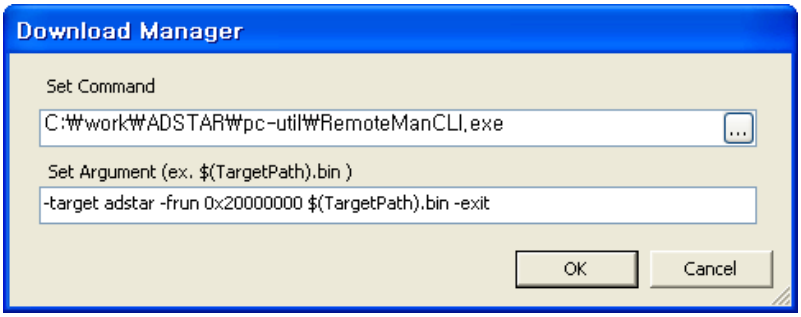
But if Linker Script is adstar.ld, after changing adstar_ram.ld, and run build.

In short,

Application that operated in flash is use adstar.ld.

Application that operated in SDRam is use adstar_ram.ld.

After checking the Linker Script, download to SDRam. Download way is like following.



1. Boot Remote Communication Mode.
2. Run build → download option.
3. Use RemoteMenCli as download program in 'pc-util'.
   option is like following.

   -target adstar
   → target name.
   -frun 0x20000000 $(TargetPath).bin
   → command to binary file download and run.
   Download to SDram start address 0x20000000

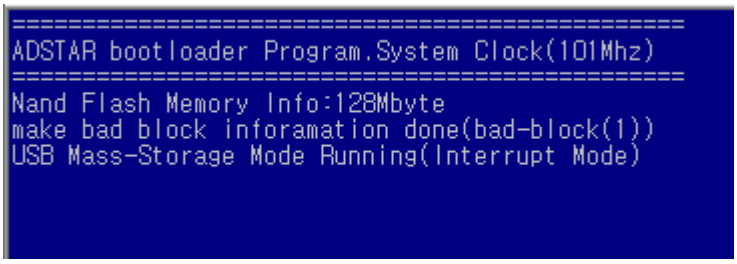
-exit

→ exit program.

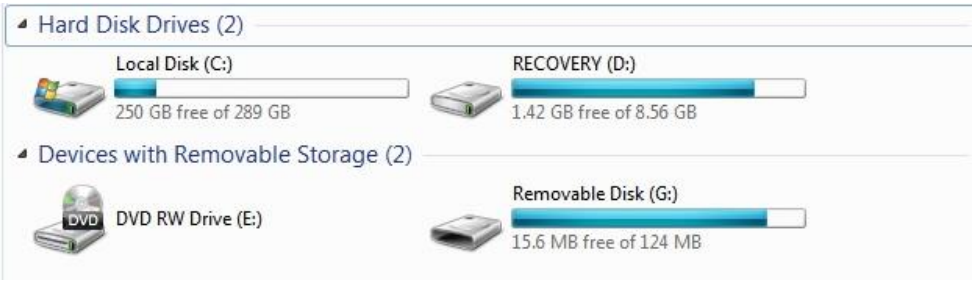
After set download option, run build → download to target.
Then, download binary and application is operated.

B. Mass Storage Mode

Mass Storage Mode is that NAND flash is used as storage device like usb memory at computer. Boot Mass Storage Mode, then display message following. And new storage device appeared.



[Mass Storage Mode]



This device is Nand Flash of the board, so copying files to the device indicates that the files could be used by the adStar programs.

C. Execute Mode Mode

Execute mode is that mode as soon running application.

Boot execute mode then display message following.

```
================================================
ADSTAR bootloader Program.System Clock(101Mhz)
================================================
Execute Mode,Run boot.bin from FLASH
startfp : 0x200039d8
START
================================================
ADSTAR Total Demo.System Clock(101Mhz)
================================================
main.c , 238 :get_pll(0)=0x608f3d0(101250000)
main.c , 239 :get_pll(1)=0x2dc6c00(48000000)
Nand Flash Memory Info:128Mbyte
make bad block inforamation done(bad-block(1))
total character count : 193
Sound Output(digital pwm) 2 channel
 CRTC 800 x 480 Setting Done
RGB565 Mode
total character count : 11365
freq : 44100 , bits : 16 , channel : 1, data-size:147456, 1 sec
freq : 44100 , bits : 16 , channel : 1, data-size:44288, 1 sec
freq : 44100 , bits : 16 , channel : 1, data-size:105984, 1 sec
freq : 44100 , bits : 16 , channel : 1, data-size:50688, 1 sec
```

[Execute Mode]

Execute mode is copying from flash data to SDRam. And run application.

You need to download to specified address. Specified address is 0x14000(sector number 20). And you need to use adstar_ram.ld as Linker Script.

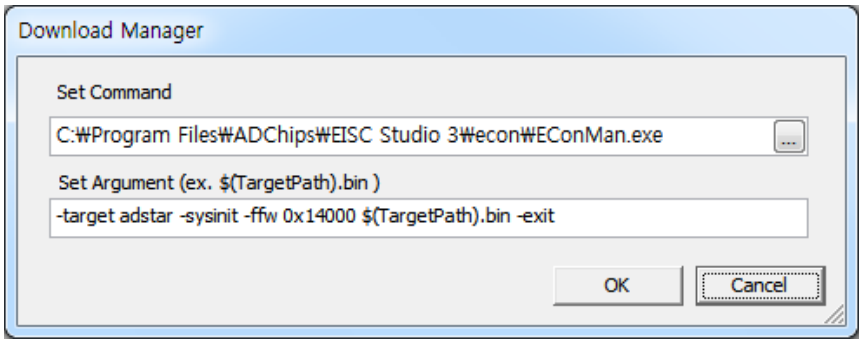
```
46  #define FLASH_APP_OFFSET  (1024*4*20)
47  static void bin_execute()
48  {
49      void (*entryfp)();
50      debugstring("Execute Mode,Run boot.bin from FLASH ₩r₩n");
51      memcpy((void*)0x20000000,(void*)FLASH_APP_OFFSET,(512*1024)-FLASH_APP_OFFSET);
52      dcache_invalidate_way();
53      entryfp = (void(*)())*(U32*)0x20000000;
54      debugprintf("startfp : %#x₩r₩n",entryfp);
55      entryfp();
56  }
```

[bootloader main.c]

There is address defined at main.c line 46. If using bootloader bigger size than original, need to change this address.

There is two way to download 0x14000 in flash.

1) Use EconMan.

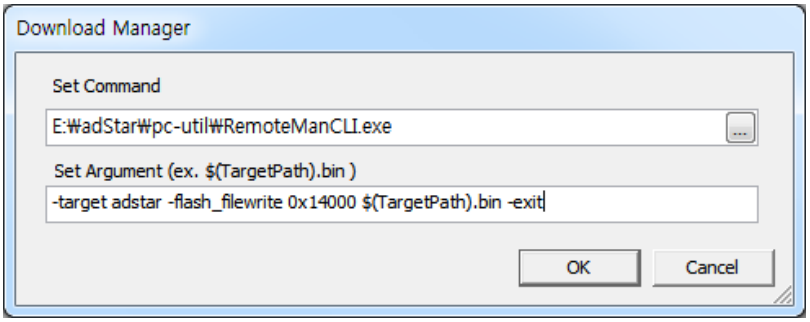


[download option]

Just change address from 0x0 to 0x14000.

2) Use RemoteMenCli.

After booting Remote Communication Mode, download by use following option.



[download option]

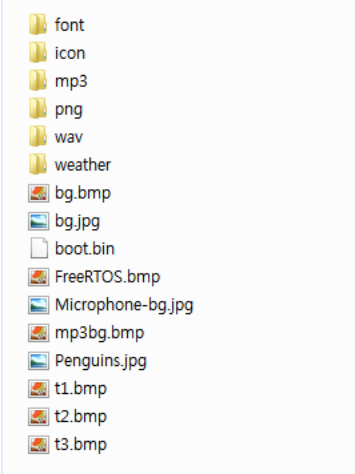
-flash_filewrite

→ flash download command.

After download, boot Execute Mode.

D. User Define Mode (execute_fat)

User Define Mode is copying boot.bin file (application binary file) from NAND Flash to SDRam and run that. So, change the file name of created file to boot.bin. And copy to NAND Flash by using Mass Storage Mode. After copying, run User Define Mode. Then you can see application is operated.



[boot.bin file of NAND Flash]

If you want using boot.bin in SD Card. you use '#define FAT_APP_TYPE DRIVE_SDCARD' instead of '#define FAT_APP_TYPE DRIVE_NAND' in main.c line 4.

3.2 Nand Boot Code

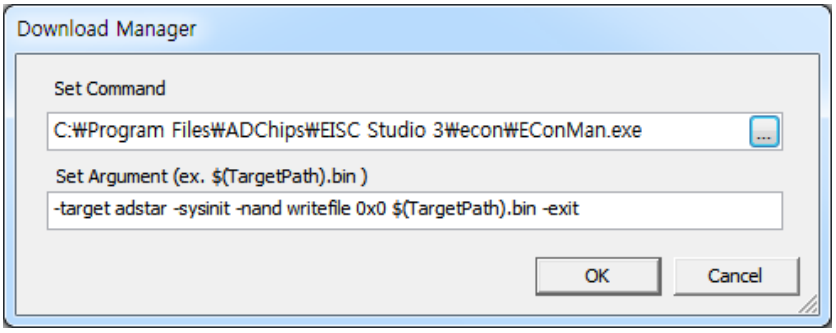
NAND Boot Code that used for NAND booting is used at adStar which does not embedded flash. If use NAND booting, need to set as NAND Flash Boot mode. Refer to datasheet for how set. NAND Boot mode is that copy 2Kbyte NAND Flash data in block 0 to SRam. Because NAND Boot mode is copied 2Kbyte NAND Flash data in block 0 to SRam, NAND Boot code size is limited as 2Kbyte. So, it is recommended to use as default status. NAND Boot Code operation is that copy 7 blocks data from block 1 to SDRam. And run application.

```
282
283  #define BOOTLOADER_STARTADDR 0x20000000
284  #define COPY_BLOCK 7//128K*7Mbyte
285  int main()
286  {
287      void (*startfp)();
288      boardinit();
289      myuart_init();
290      *R_NFMCFG   = 0x2222;
291      nand_reset();
292      mydebugstring("Nand Booting.₩r₩n");
293      //U16 id = nand_id_read();
294      copydata_from_nand((U8*)BOOTLOADER_STARTADDR,1,COPY_BLOCK);
295      startfp = (void (*)())(*(volatile unsigned int*)BOOTLOADER_STARTADDR);
296      startfp();
297      while(1);
298      return 0;
299  }
300
```

[main.c of nand boot code]

3.2.1 Use Nand Boot Code

Open NAND Boot code, build and download to NAND Flash block 0. Download option is like following.

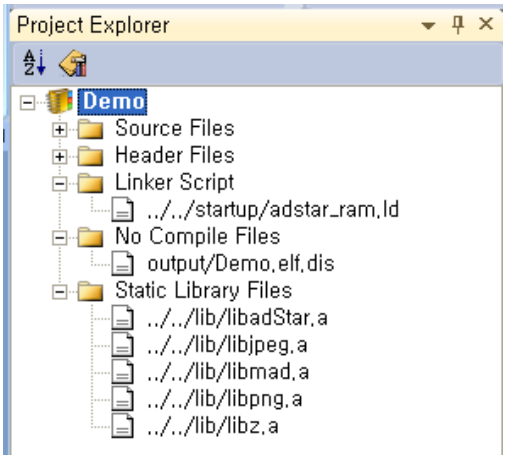


[Download Option]

-nand writefile

➔ NAND Flash download command.

Next, explain the method about downloading application code which operated by copying to SDRam at NAND Boot code. Code that operated at SDRam is used adStar_ram.ld as Linker Script.



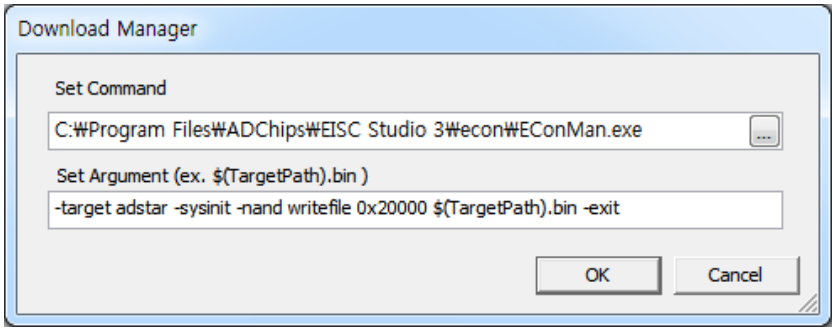
[Linker Script]

After checking Linker Script, build and download.

As before, you use '–nand writefile' command. Download address is different from the previous download. Download to NAND Flash block number 1.

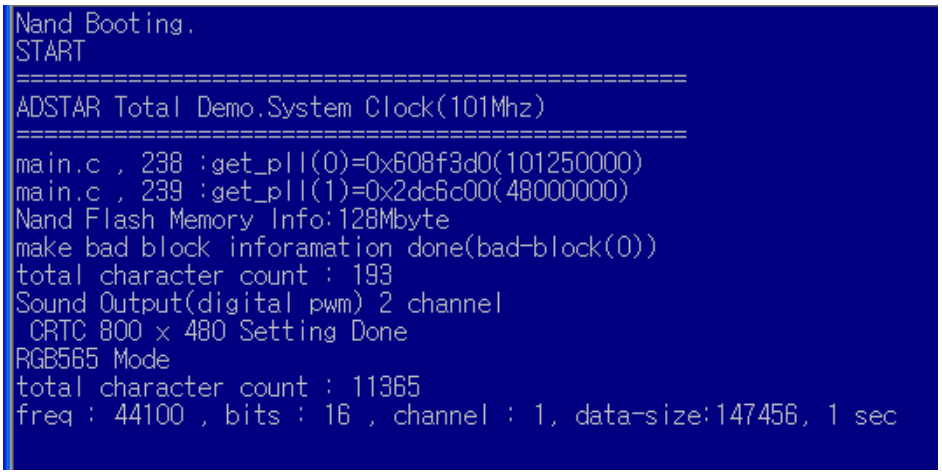
You need to caution for different block number 1 address of each NAND FLASH.

(small page -> block 1 address : 0x4000, large page -> block 1 address : 0x20000)



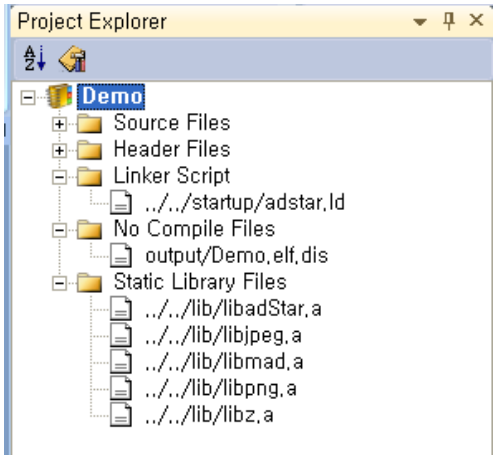
[Download Option]

After download, turn on board. Then, display NAND Booting message. And application that downloaded to block 1 will operated.

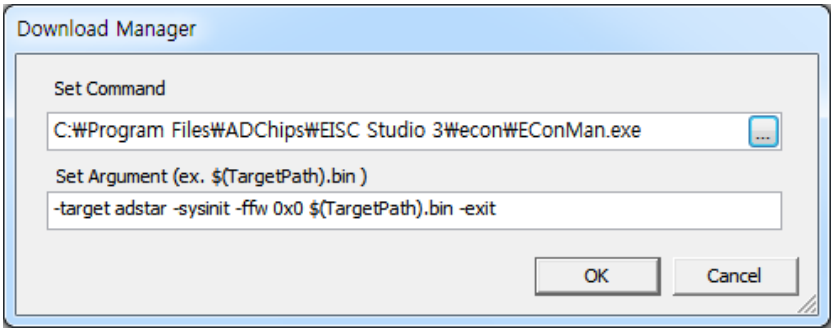


3.3 Use without BootLoader

In this chapter, explain the method about operate adStar without bootloader. For operating adstar without bootloader, you need to change Linker Script file and download address. Use adstar.ld as Linker Script file. And use download address 0. Open project and change Linker Script. After build project, download to 0 in flash. When you reboot, you can see that application is operated without bootloader.



[Linker Script]



[Download Option]

4. lib_config.h

lib_config.h in SDK library source project file (adstar.epx) is config file. At this chapter, explain the lib_config.h file.

lib_config.h file begins as explain for nested interrupt setting.

```
23  #define SUPPORT_NESTED_INTNUM_EIRQ0 0
24  #define SUPPORT_NESTED_INTNUM_TIMER0 0
25  #define SUPPORT_NESTED_INTNUM_SOUNDMIXER 0
26  #define SUPPORT_NESTED_INTNUM_DMA0 1 //DMA is used by sound , 1 is recommended
27
28  #define SUPPORT_NESTED_INTNUM_FRAMESYNC 0
29  #define SUPPORT_NESTED_INTNUM_GPIOA 0
30  #define SUPPORT_NESTED_INTNUM_UART0 0
31  #define SUPPORT_NESTED_INTNUM_DMA1 1
32
33  #define SUPPORT_NESTED_INTNUM_EIRQ1 0
34  #define SUPPORT_NESTED_INTNUM_TIMER1 0
35  #define SUPPORT_NESTED_INTNUM_PMU 0
36  #define SUPPORT_NESTED_INTNUM_DMA2 1
37
38  #define SUPPORT_NESTED_INTNUM_SPI0 0
39  #define SUPPORT_NESTED_INTNUM_GPIOB 0
40  #define SUPPORT_NESTED_INTNUM_UART1 0
41  #define SUPPORT_NESTED_INTNUM_DMA3 1
42
43  #define SUPPORT_NESTED_INTNUM_RESV1 0
44  #define SUPPORT_NESTED_INTNUM_TIMER2 0
45  #define SUPPORT_NESTED_INTNUM_USB 0
46  #define SUPPORT_NESTED_INTNUM_DMA4 1
47
48  #define SUPPORT_NESTED_INTNUM_USBHOST 0
49  #define SUPPORT_NESTED_INTNUM_GPIOC 0
50  #define SUPPORT_NESTED_INTNUM_UART2 0
51  #define SUPPORT_NESTED_INTNUM_DMA5 1
52
53  #define SUPPORT_NESTED_INTNUM_RESV2 0
54  #define SUPPORT_NESTED_INTNUM_TIMER3 0
55  #define SUPPORT_NESTED_INTNUM_NAND 0
56  #define SUPPORT_NESTED_INTNUM_DMA6 1
```

When set to 1, means you use nested interrupt on selected interrupt.
DMA interrupt has been set 1 by default. Because DMA interrupt use on mp3 decoding.
You can set up to max 14 nested interrupt.

```
95   */
96   #define TWI_RESP_TIME_OUT_COUNT (7200*100)  // About 100ms @ AHB 101MHz
97
98  /*****************************************************************************
99      External NAND/SD-Card/USB Memory Supports
100 *****************************************************************************/
101
102  #define CONFIG_NAND 1
103  #define CONFIG_SDCARD 1
104
105 #if (CONFIG_SDCARD != 0)
116  #define SDCARD_PIN_USE_67 0
117  #endif
```

< Set the TWI timeout count and storage device to using >

```
120 /*****************************************************************************
121     Image file supports
122 *****************************************************************************/
123
124  #define CONFIG_SUPPORT_BMP 1
125  #define CONFIG_SUPPORT_TGA 1
126  #define CONFIG_SUPPORT_JPG 1
127  #define CONFIG_SUPPORT_PNG 1
```

< Set the image format to using >

```
153
154 /*****************************************************************************
155     UART Config
156 *****************************************************************************/
157
158
159
160  #define UART_BUF_SIZE 512
161  #define CONFIG_UART_RX_INTERRUPT
162  //#define CONFIG_UART_TX_INTERRUPT
163
164
165
166
167  #define DEBUG_CHANNEL 0
```

< Set the uart interrupt and uart buf size >

```
169 /*****************************************************************************
170     SYSTEM CLOCK
171 *****************************************************************************/
172
173
174
175
176
177
178  //#define OSC_CLOCK 8000000
179  #define OSC_CLOCK 10000000
180
181
182 /*****************************************************************************
183 *   SOUND Mixer
184 *****************************************************************************/
185 '
186
187
188
189
190
191  #define WAVE_BUF_MAX (512*1024) //if you use MP3, > 5Kbyte
192  #define DEFAULT_VOLUME 255//max 255
193
194
195
196 */
197  #define SND_OUTPUT_CHANNEL  2
198 '
199
200 */
201  #define SND_OUTPUT_HZ   48000
202
```

< Set the crystal clock, sound buffer size and sound channel >

At system clock part, set the crystal value you use.

```
#define OSC_CLOCK  10000000    (default)
// #define OSC_CLOCK  8000000
```

5. UART

5.1 uart_config

BOOL uart_config(**int** ch, **int** baud, **UART_DATABITS** databits, **UART_STOPBITS** stopbit,
UART_PARITY parity)

Function to initialize UART.

Parameter

ch UART initial channel value.

baud UART baud rate value.
SDK uses 115200 as default.

databits UART transmission data bit setting. SDK defines data bit as below.

```
typedefenum{
    DATABITS_5 = 5,
    DATABITS_6 = 6,
    DATABITS_7 = 7,
    DATABITS_8 = 8
}UART_DATABITS;
```

stopbit UART stop bit setting. SDK defines stop bit as below.

```
Typedefenum{
    STOPBITS_1 = 1,
    STOPBITS_2 = 2
}UART_STOPBITS;
```

parity UART parity bit setting. SDK defines parity bit as below.

```
Typedefenum{
    UART_PARNONE = 0,
    UART_PARODD,
    UART_PAREVEN
}UART_PARITY;
```

Return Value

TRUE(1) or FALSE(0)

5.2 uart_putch

BOOL uart_putch(**int** n, **char** ch)

Display the 'ch' value through the UART 'n' channel.

Parameter

n	Channel to display the 'ch' value. adStar has 5 channels.
ch	Value to be displayed through UART.

Return Value

TRUE(1) or FALSE(0)

5.3 uart_putdata

BOOL uart_putdata(**int** n, **U8*** buf, **int** len)

Display 'len' length characters stored in 'buf' through the UART n channel.

Parameter

n	Channel to display the 'ch' value. adStar has 5 channels.
buf	Buffer that stores characters to be displayed.
len	Length of characters to be displayed.

Return Value

TRUE(1) or FALSE(0)

5.4 uart_putstring

BOOL uart_putstring(**int** n, **U8*** buf)

Display characters stored in 'buf' through the UART n channel.

Parameter

n	Channel to display the 'ch' value. adStar has 5 channels.
buf	Buffer that stores characters to be displayed.

Return Value

TRUE(1) or FALSE(0)

5.5 uart_getch

BOOL uart_getch(**int** n, **char*** ch)

Store received 1 Byte value (1 character) through the UART n channel to 'ch'.

Parameter

n	UART channel to be input the value. adStar has 5 channels.
ch	Variable to be input a value through UART.

Return Value

TRUE(1) or FALSE(0)

5.6 uart_getdata

int uart_getdata(**int** n, **U8*** buf, **int** bufmax)

Read 'bufmax' length of characters through the UART n channel and store the characters to 'buf'.

Parameter

n	UART channel to be input the value. adStar has 5 channels.
Buf	Buffer to be input the value through UART.
Bufmax	Number of values (characters) to read. (Byte unit)

Return Value

The total number of elements successfully read is returned as byte unit. If the value same as the 'Bufmax' value, this function is successfully finished, unless an error occurred.

5.7 uart_rx_flush

void uart_rx_flush(**int** ch)

Initialize rxfifo of the UART n channel.

Parameter

ch	UART channel to be initialized.

Return value

None.

5.8 uart_tx_flush

void uart_tx_flush(**int** ch)

Initialize txfifo of the UART n channel.

Parameter

ch UART channel to be initialized.

Return value

None.

5.9 set_debug_channel

Void set_debug_channel(**int** ch)

Set debugging message displaying UART channel that is used by debugging functions such as **debugprintf**, **debugstring**, **PRINTVAR and PRINTLINE** .

Parameter

ch Debugging purpose UART channel.

Return value

None.

5.10 get_debug_channel

int get_debug_channel( )

Return the debugging purpose UART channel.

Parameter

None

Return Value

Current debugging purpose UART channel value.

5.11 debugprintf

void debugprintf(**const char***const format, . . .)

It has same role with C language's 'printf', displays numbers, characters and variable values through UART. The output UART channel is configured by **set_debug_channel()** function. The default debugging channel is 0.

Note. Though set to use Tx interrupt, this function operates in a poling.

Usage

debugprintf("result number : %d₩r₩n",result);

→Displays the variable value of 'result' in decimal number with "result number: "string through UART and breaks line. The usage is same as 'printf'. However, '₩r₩n' must be denoted when line breaking.

5.12 debugstring

void debugstring(**const char*** str)

Function to display string. It is used only for string displaying. Output UART channel is the debugging channel from **set_debug_channel()** function. The default channel is 0.

Just for reference, it is possible to display only string with **debugprintf()** function.

Note. Though set to use Tx interrupt, this function operates in a poling.

Usage

debugstring("=== adStar Start ===₩r₩n");

→string in "" is displayed through UART.

5.13 PRINTLINE

It is a macro function, displays a line number where this function is called through UART. It uses the debugging channel. The default channel is 0.

Usage

PRINTLINE;

→ Displays a line number where this function is called.

5.14 PRINTVAR(A)

It is a macro function, displays a line number and a value of 'A' where this function is called through UART. It uses the debugging channel. The default channel is 0

Usage

int a = 10;
PRINTVAR(a);;
→ Displays a value of 'a'. Even a register value could be displayed.

5.15 UART Example

```
#include "adStar.h"

int main()
{
    boardinit();
    uart_config(0, 115200, DATABITS_8, STOPBITS_1, UART_PARNONE );
                    // Sets the UART 0 channel as below.
                    // Baud rate = 115200
                    // Data bit = 8bit
                    // Stop bit = 1
                    // Parity = none
    debugstring("==========================================₩r₩n");
    debugprintf("ADSTAR UART Example. System clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
    debugstring("==========================================₩r₩n");
    U8 ch;
    While(1)
    {
        if(uart_getch(0, &ch))      // Reads 1Byte data from UART 0 and stores 'ch'.
        {
            uart_putch(0, ch);    // Displays UART 0 input data to UART 0.
        }
    }
}
```

6. INTERRUPT

6.1 init_interrupt

void init_interrupt(**void**);

The interrupt initialization function. It should be called to use the interrupt related function. **startup_adstar.s** of the adStar SDK calls and initializes the interrupt.

6.2 set_interrupt

BOOL set_interrupt(**INTERRUPT_TYPE** intnum, **void** (*fp)());

Call back function registration function of the interrupt is called.

Caution: the interrupt callback functions of UART, Sound mixer are already made and registered from SDK, no redundant operations are avoided.

Parameter

intnum	interrupt type (refer to next chapter about the interrupt type)
(*fp)()	callback function of the interrupt

Return value

TRUE or FALSE

6.3 enable_interrupt

void enable_interrupt(**INTERRUPT_TYPE** intnum, **BOOL** b);

Function to enable the interrupt. Register the interrupt function with **set_interrupt()** and enable it with **enable_interrupt()**.

Parameter

intnum	interrupt type (refer to next chapter about the interrupt type)
b	1 or TRUE => interrupt enable
	0 or FALSE => interrupt disable

Return value

TRUE or FALSE

INTERRUPT_TYPE1

INTERRUPT_TYPE		INTERRUPT_TYPE	
INTNUM_EIRQ0	External interrupt 0	INTNUM_UART0	UART 0 interrupt
INTNUM_EIRQ1	External interrupt 1	INTNUM_UART1	UART 1 interrupt
INTNUM_TIMER0	Timer interrupt 0	INTNUM_UART2	UART 2 interrupt
INTNUM_TIMER1	Timer interrupt 1	INTNUM_UART3	UART 3 interrupt
INTNUM_TIMER2	Timer interrupt 2	INTNUM_UART4	UART 4 interrupt
INTNUM_TIMER3	Timer interrupt 3	INTNUM_PMU	PMU interrupt
INTNUM_SOUNDMIXER	Sound mixer interrupt	INTNUM_SPIO	SPI 0 interrupt
INTNUM_DMA0	DMA interrupt 0	INTNUM_SPI1	SPI 1 interrupt
INTNUM_DMA1	DMA interrupt 1	INTNUM_USB	USB device interrupt
INTNUM_DMA2	DMA interrupt 2	INTNUM_USBHOST	USB host interrupt
INTNUM_DMA3	DMA interrupt 3	INTNUM_NAND	NAND Flash interrupt
INTNUM_DMA4	DMA interrupt 4	INTNUM_SDHC	SDHC interrupt
INTNUM_DMA5	DMA interrupt 5	INTNUM_ADC	ADC interrupt
INTNUM_DMA6	DMA interrupt 6	INTNUM_WATCHDOG	Watch dog interrupt
INTNUM_DMA7	DMA interrupt 7	INTNUM_TWI	TWI interrupt
INTNUM_FRAMESYNC	Frame sync interrupt	INTNUM_CAPOVERFLOW	Capture overflow interrupt
INTNUM_GPIOA	GPIO A interrupt		
INTNUM_GPIOB	GPIO B interrupt	INTNUM_PWM	PWM interrupt
INTNUM_GPIOC	GPIO C interrupt		
INTNUM_GPIOD	GPIO D interrupt		
INTNUM_GPIOE	GPIO E interrupt		
INTNUM_GPIOF	GPIO F interrupt		
INTNUM_GPIOG	GPIO G interrupt		
INTNUM_GPIOH	GPIO H interrupt		
INTNUM_GPIOI	GPIO I interrupt		
INTNUM_GPIOJ	GPIO J interrupt		

6.4 Interrupt Example

```
#include “adStar.h“

void EIRQ0ISR( )
{
	debugprintf(“EIRQ0 Interrupt₩r₩n”);
}

int main()
{
	boardinit();
	uart_config(0, 115200, DATABITS_8, STOPBITS_1, UART_PARNONE );
	set_interrupt( INTNUM_EIRQ0, EIRQ0ISR );
	//Registers a callback function of the interrupt when External Interrupt 0 occurred.
	enable_interrupt( INTNUM_EIRQ0, TRUE );
	//Enables External Interrupt 0.

	while(1)
	return 0;
}
```

7. TIMER

7.1 set_timer

BOOL set_timer(**int** nCh, **U32** ms)

Function to configure and operate the nCh channle timer. Determines the timer channel and period to be used and enables the timer operations at timer control register.
When this function is called after the timer interrupt is registered with **set_interrupt()** function, the timer interrupt occurred with the configured period.

Parameter

nCh	Configured timer channel value.
ms	timer interrupt period. (ms unit)

Return Value

TRUE(1) or FALSE (0)

7.2 stop_timer

BOOL stop_timer(**int** nCh)

Stops the nCh channel timer. Disables the nCh channel timer operations.

Parameter

nCh	Timer channel value to be disabled.

Return Value

TRUE(1) or FALSE (0)

7.3 delayms

BOOL delayms(**U32** ms)

Takes ms delay. (ms unit)

Parameter

ms	Time amount of delay, ms unit.

Return Value

TRUE(1) or FALSE (0)

7.4 TIMER Example

```
#include "adStar.h"

void TIMER0ISR( )
{
        debugprintf("==TIMER0ISR==₩r₩n");
}

int main()
{
        boardinit();
        uart_config(0, 115200, DATABITS_8, STOPBITS_1, UART_PARNONE );
        set_interrupt( INTNUM_TIMER0, TIMER0ISR );
                        //Register a callback function of the interrupt of Timer 0 Interrupt.
        set_timer( 0, 1000);
                        // Timer0 interrupt occurred every 1 second.
        delayms(5000);
                        // 5 seconds delay.
        stop_timer( 0 );
                        // Disables Timer 0 Interrupt.  No more timer.
        while(1)
        return 0;
}
```

8. GRAPHIC

8.1 setscreen

void setscreen(**SCREENRES** size, **U32** screenmode)

Set LCD resolution and RGB mode. It should be set first for LCD displaying.
Configure the LCD controller according to the set LCD resolution.

Parameter

size — LCD resolution value, the LCD Controller is configured according to the resolution value. SCREENRES are defined as below. (Each LCD has different characteristics. Thus, if LCD is not working correctly with the proper resolution then the values in **crt.c**'s **setscreen()** function should be modified by the LCD characteristics.)

```
Typedefenum {
        SCREEN_480x272 = 0,
        SCREEN_640x480,
        SCREEN_800x480,
       } SCREENRES;
```

Screenmode — RGB mode setting value, defined as below.

SCREENMODE_RGB888
SCREENMODE_RGB565

Return value

None

8.2 createframe

SURFACE* createframe(**U32** w, **U32** h, **U32** bpp)

Function to create a frame (memory region) on a screen. Creates w width x h height region that images or shapes could be drawn. More than one frame is needed for the screen output.

Parameter

w	Sets width of the frame.
h	Sets height of the frame.
bpp	Sets bit per pixel value of the frame.

Return value

Returns the SURFACE structure that has the frame information. SURFACE is defined as below.

```
typedefstruct {
        U32 w;
        U32 h;
        U32 bpp;
        void*  pixels;
        U32 pixtype;
        PALETTE* pal;
        U8 ashiftbit;
        U8 rshiftbit;
        U8 gshiftbit;
        U8 bshiftbit;
        void* reserve;
} SURFACE;
```

8.3 setframebuffer

void setframebuffer(**SURFACE*** surf)

Function to choose a frame to be displayed on the screen. If the frame that is created by createframe() is chosen, then only the frame is shown on the screen. (Single frame)
For screen output, the frame must be chosen by calling one of these functions; setframebuffer(), setdoubleframebuffer().

Parameter

surf	createframe() created SURFACE structure.

Return value

None.

8.4 setdoubleframebuffer

void setdoubleframebuffer(**SURFACE*** surf, **SURFACE*** surf2)

Function to set two frames will be displayed on the screen. set two frames and show the frames alternately on the screen using **getbackframe()** and **flip()** functions (double frame). For convenience, the showing frame is called as a front frame and the other frame is called as a back frame.
(To use **setdoubleframebuffer()** function, two frames should be created.)
For screen output, the frame must be chosen by calling one of these functions; **setframebuffer()**, **setdoubleframebuffer()**.

Parameter

surf	**createframe()** created SURFACE structure First frame pointer for the screen output.
surf2	**createframe()** created SURFACE structure Second frame pointer for the screen output.

Return value

None.

8.5 setframebufferxy

void setframebufferxy(**SURFACE*** surf, **U32** x, **U32** y);

Function to control the on-screen frame's start location and shows the frame on the screen from the particular point of it. Could be used when the frame size is bigger than the screen size and max values of x/y could be difference value of the screen resolution within the frame size.
(Example: 800*480 resolution, frame size: 1024*512, then Max x = 224, Max y = 32)
Before using **setframebufferxy()**, the on-sceen frame must be chosen by calling one of these functions; **setframebuffer()**, **setdoubleframebuffer()**.

Parameter

surf	**createframe()** created SURFACE structure.
x	starting x coordinate to be displayed among all coordinate of the frame
y	starting y coordinate to be displayed among all coordinate of the frame.

Return value

None.

8.6 set_draw_target

Void set_draw_target(**SURFACE*** surf)

Function to choose frame to display images or to draw shapes. By denoting the frame created by **createframe()**, makes a drawer to draw things to the frame when the draw related functions are called.

Parameter

surf frame pointer. **createframe()** created SURFACE structure

Return value

None.

8.7 get_draw_target

SURFACE* get_draw_target( );

Bring the frame pointer that is a drawtarget currently.

Parameter

None.

Return value

frame pointer. **createframe()** created SURFACE structure.

8.8 getbackframe

SURFACE*getbackframe()

Function returns hidden screen frame's pointer, when using two frames with **setdoubleframebuffer()** function. Mostly, it is used to draw to the off-screen frame with **setdrawtarget()** function.

Parameter

None.

Return value

Returns **createframe()** created SURFACE structure. Has the frame information.

8.9 getfrontframe

SURFACE*getfrontframe()

Function returns the on-screen frame pointer when using two frames with **setdoubleframebuffer()** function.

Parameter

None.

Return value

Returns **createframe()** created SURFACE structure. Has the frame information.

8.10 flip

void flip()

Function to switchover between frames when using two frames with **setdoubleframebuffer()** function.

Parameter

None.

Return value

None.

8.11 getscreenwidth

U32 getscreenwidth( )

Function to return a current window's width size. The value is configured by **setscreen()** function is returned.

Parameter

None.

Return value

Returns window's width size.

8.12 getscreenheight

U32 getscreenheight( )

Function to return a current window's height size. The value is configured by **setscreen()** function is returned.

Parameter

None.

Return value

Returns window's height size.

8.13 getscreenpitch

U32 getscreenpitch()

Function to return the pitch value of the screen. The pitch value is configured by **setscreen()** function is returned. The pitch value is calculated by screen width × bpp÷ 8.

Parameter

None.

Return value

Returns screen's pitch value.

8.14 getscreenbpp

U32 getscreenbpp()

Function to return the bpp(bit per pixel) value of the screen. The bpp value is configured by **setscreen()** function is returned.

.

Parameter

None.

Return value

Returns screen's bbp value.

8.15 drawputpixel

void drawputpixel(**int** x, **int** y, **U8** r, **U8** g, **U8** b);

Function to draw a single pixel. Draws a pixel at specific location with specific color. It gives RGB values directly, so **drawsetrgb( )** function has no effect on it.

Parameter

x	x coordinate where draws the pixel
y	y coordinate where draws the pixel
r	r value of pixel RGB value. (0 ~ 255)
g	g value of pixel RGB value. (0 ~ 255)
b	b value of pixel RGB value. (0 ~ 255)

Return value

None.

8.16 draw_line

void draw_line(**U32** x1, **U32** y1, **U32** x2, **U32** y2, **EGL_COLOR** color)

Function to draw a line from x1, y1 coordinate to x2, y2 coordinate. The line color is configured by MAKE_COLORREF( ) macro function.

Parameter

x1	x coordinate where the line starts.
y1	y coordinate where the line starts.
x2	x coordinate where the line ends.
y2	y coordinate where the line ends.
color	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return Value

None.

8.17 draw_rect

void draw_rect(**S16** x, **S16** y, **U16** w, **U16** h, **EGL_COLOR** c)

Function to draw a rectangle (with width w / height h) that starts from coordinates x, y.

Parameter

x	x coordinate where the rectangle starts

y	y coordinate where the rectangle starts
w	rectangle's width.
h	rectangle's height.
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.18 draw_rectfill

void draw_rectfill(**U32** x, **U32** y, **U32** w, **U32** h, **EGL_COLOR** c)

Function to draw a filled rectangle (with width w / height h) that starts from coordinates x, y.

Parameter

x	x coordinate where the rectangle starts
y	y coordinate where the rectangle starts
w	rectangle's width.
h	rectangle's height.
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.19 draw_roundrect

void draw_roundrect(**S16** x0, **S16** y0, **U16** w, **U16** h, **U16** corner, **EGL_COLOR** c)

Function to draw a rounded corner rectangle (with width w / height h) that starts from coordinates x, y. The corner values adjust the round strength.

Parameter

x0	x coordinate where the rectangle starts
y0	y coordinate where the rectangle starts
w	rectangle's width.
h	rectangle's height.
corner	corner round strength. Bigger values indicates wider round.
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.20 draw_roundrectfill

void draw_roundrectfill(**S16** x0, **S16** y0, **U16** w, **U16** h, **U16** corner, **EGL_COLOR** c)

Function to draw a rounded corner rectangle (with width w / height h) that starts from coordinates x, y. The rectangle is filled. The corner values adjust the round strength.

Parameter

x0	x coordinate where the rectangle starts
y0	y coordinate where the rectangle starts
w	rectangle's width.
h	rectangle's height.
corner	corner round strength. Bigger values indicates wider round.
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.21 draw_circle

void draw_circle(**U32** x, **U32** y, **U32** rad, **EGL_COLOR** color)

Function to draw a circle that has coordinate (x, y) as a center with a radius (rad).

Parameter

x	x coordinate of the center
y	y coordinate of the cetner
rad	radius of the circle
color	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.22 draw_circlefill

void draw_circlefill(**U32** x0, **U32** y0, **U32** rad, **EGL_COLOR** c)

Function to draw a filled circle that has coordinate (x, y) as a center with a radius (rad).

Parameter

x	x coordinate of the center
y	y coordinate of the cetner
rad	radius of the circle
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.23 draw_ellipse

void draw_ellipse (**S16** x0, **S16** y0, **U16** Xradius, **U16** Yradius, **EGL_COLOR** c);

Function to draw an ellipse that has coordinate (x0, y0) as a center with a radius along the x axis (Xradius) and a radius along the y axis (Yradius).

Parameter

x0	x coordinate of the center
y0	y coordinate of the cetner
Xradius	radius along the x axis of the ellipse
Yradius	radius along the y axis of the ellipse
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.24 draw_ellipsefill

void draw_ellipsefill (**S16** x0, **S16** y0, **U16** Xradius, **U16** Yradius, **EGL_COLOR** c);

Function to draw a filled ellipse that has coordinate (x0, y0) as a center with a radius along the x axis (Xradius) and a radius along the y axis (Yradius).

Parameter

x0	x coordinate of the center
y0	y coordinate of the cetner
Xradius	radius along the x axis of the ellipse
Yradius	radius along the y axis of the ellipse
c	Color value. Use MAKE_COLORREF(r, g, b) macro function.

Return value

None.

8.25 loadbmp

SURFACE* loadbmp(**char*** fname);

Function to load a bmp file. Loads a bmp file, creates a memory region, creates a SUFRACE structure with **drawsurface()** function and stores it.

.

Parameter

fname	bmp file name

Return value

A pointer to the image information and data stored memory region.

8.26 loadbmpp

SURFACE* loadbmpp(**U8*** startaddr)

Function to load a bmp image from memory. It loads the image from memory not from a file like Loadbmp function.

Parameter

startaddr	Image stored memory address .

Return value

A pointer to the image information and data stored memory region.

8.27 loadjpg

SURFACE* loadjpg(**char*** fname);

Function to load a jpg file. Loads a jpg file, creates a memory region, creates a SUFRACE structure with **drawsurface()** function and stores it.
(**libjpeg.a** file should be added to the project for the jpg image output.)

Parameter

fname	jpg file name

Return value

A pointer to the image information and data stored memory region.

8.28 loadjpgp

SURFACE* loadjpgp(**U8*** databuf, **U32** len)

Function to load a jpg image from memory. It loads the image from memory not from a file like Loadjpg function.

Parameter

databuf	Image stored memory address.
len	length of the image data

Return value

A pointer to the image information and data stored memory region.

8.29 loadtga

SURFACE* loadtga(**char*** fname)

Function to load a tga file. Loads a tga file, creates a memory region, creates a SUFRACE structure with **drawsurface()** function and stores it.

Parameter

fname	tga file name

Return value

A pointer to the image information and data stored memory region.

8.30 loadtgap

SURFACE* loadtga(**U8*** startaddr)

Function to load a tga image from memory. It loads the image from memory not from a file like Loadtga function.

Parameter

Startaddr Image stored memory address.

Return value

A pointer to the image information and data stored memory region.

8.31 loadpng

SURFACE* loadpng(**char*** filename)

Function to load a png file. Loads a png file, creates a memory region, creates a SUFRACE structure with **drawsurface()** function and stores it.
(libz.a, libpng.a file should be added to the project for the png image output.)

Parameter

filename png file name

Return value

A pointer to the image information and data stored memory region.

8.32 loadpngp

SURFACE* loadpngp(**U8*** pngbuf, **U32** datalen)

Function to load a png image from memory. It loads the image from memory not from a file like Loadpng function.

Parameter

Pngbuf Image stored memory address
Datalen length of the image data

Return value

A pointer to the image information and data stored memory region.

8.33 loadsurf

SURFACE* loadsurf(**char*** fname)

Function to load a suf file. Loads a suf file, creates a memory region, creates a SUFRACE structure with **drawsurface()** function and stores it.
suf file is created by MakeSurface2.exe in pc-util folder. MakeSurface2.exe is converting program (BMP, JPG, PNG ➔ suf). suf file loading time is shortest.

Parameter

fname	File name

Return value

A pointer to the image information and data stored memory region.

8.34 loadimage

SURFACE*loadimage(**char*** fname)

Function to load an image. It can load any kinds of image files such as bmp, jpg, tga and png. Loads an appropriate image load function from inside of this function according to the image types.

Parameter

fname	Image file name

Return value

A pointer to the image information and data stored memory region.

8.35 draw_surface

BOOL draw_surface(**SURFACE*** src_surf, **int** dx, **int** dy)

Function to output an image.

Parameter

src_surf	A pointer to the memory region storing the returned image data from load function.
dx	x coordinate where the image locates.
dy	y coordinate where the image locates.

Return value

TRUE or FALSE

8.36 draw_surface_rect

BOOL draw_surface_rect(**SURFACE*** src_surf, **int** dx, **int** dy, **int** sx, **int** sy, **int** w, **int** h)

Function to output a specific region of an image. Specified region of the image (start point, size) is displayed at coordinate (dx,dy). It is used to output parts of the image.

Parameter

src_surf	A pointer to the memory region storing the returned image data from load function.
dx	x coordinate where the image locates.
dy	y coordinate where the image locates.
sx	x coordinate of the image to be displayed.
sy	y coordinate of the image to be displayed.
w	horizontal length of the image to be displayed. It displays w length of the image from the coordinate sx.
h	vertical length of the image to be displayed. It displays h length of the image from the coordinate sy.

Return value

TRUE or FALSE

8.37 draw_set_clip_winodw

void draw_set_clip_window(**SURFACE*** dst, **CLIP_RECT*** pRect)

Function to limit the region that an image could be displayed. The image cannot be displayed outside of the region from pRect.

Parameter

dst	Destination frame.
pRect	Display area.

```
typedef_struct
{
    int  x;
    int  y;
    int  endx;
```

```
    int  endy;
} CLIP_RECT;
```

Return value

None.

8.38 draw_surface_scale

BOOL draw_surface_scale(**SURFACE*** src_surf, **int** dx, **int** dy, **int** dw, **int** dh)

Function to output an image with scaling. The image is up-scaled, if dw/dh values are bigger than the original image's width/height length. The image is down-scaled when vice-versa
.

Parameter

src_surf	A pointer to the memory region storing the returned image data from load function.
dx	x coordinate where the image is displayed.
dy	y coordinate where the image is displayed.
dw	horizontal length of the displaying image. If this value is bigger than the original image's width then the horizontal length will be up-scaled or (smaller) down-scaled.
dh	vertical length of the displaying image. If this value is bigger than the original image's height then the vertical length will be up-scaled or (smaller) down-scaled.

Return value

TRUE or FALSE

8.39 draw_surface_scalerect

BOOL drawsurfacescalerect(**SURFACE*** src_surf, **int** dx, **int** dy, **int** dw, **int** dh,
int sx, **int** sy, **int** sw, **int** sh)

Function to merge the functions of **drawsurfacescale()** function and **drawsurfacerect()** function. Up/down scaled outputs the specific part of the image. Outputs a region of the image that is originated from (sx, sy) to sw,sh lengths to the coordinate (dx, dy) with dw, dh lengths. Up-scaled when sw,sh values are smaller than dw, dh, down-scaled vice-versa.

Parameter

src_surf	A pointer to the memory region storing the returned image data from load function.
dx	x coordinate where the image is displayed.
dy	y coordinate where the image is displayed.
dw	horizontal length of the displaying image. If this value is bigger than the original image's width then the horizontal length will be up-scaled or (smaller) down-scaled.
dh	vertical length of the displaying image. If this value is bigger than the original image's height then the vertical length will be up-scaled or (smaller) down-scaled.
sx	initial x coordinate of the part of the original image to be displayed.
sy	initial y coordinate of the part of the original image to be displayed.
sw	horizontal length of the output image. From sx to sw
sh	vertical length of the output image. From sy to sh

Return value

TRUE or FALSE

8.40 release_surface

void release_surface(**SURFACE*** surf);

Function to release the image load function created memory region. It is better to release unused image's memory region with this function because available memory is limited.

Parameter

surf	A memory region pointer storing returned image data from load function.

Return value

None.

8.41 createsurface_from

SURFACE* createsurface_from(**SURFACE*** src, **U32** option);

Make copy of the surface. Can make rotated copy. Option is like following.

```
#define PIVOT_RIGHT (1)        // 90 degree right rotation
#define PIVOT_LEFT (1<<1)      // 90 degree left rotation.
#define PIVOT_180 (1<<2)       // 180 degree.
#define PIVOT_VFLIP (1<<4)     // up down invert.
#define PIVOT_HFLIP (1<<5)     // right left invert
```

Parameter

src	Original surface.
option	copy option.

Return value

copied surface..

Example

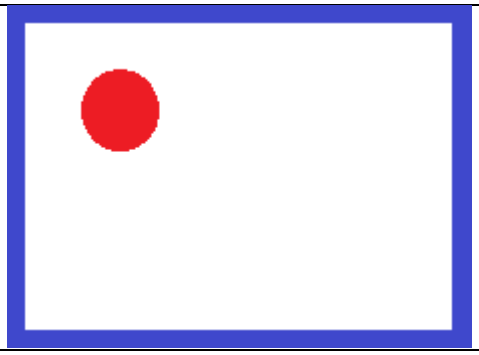
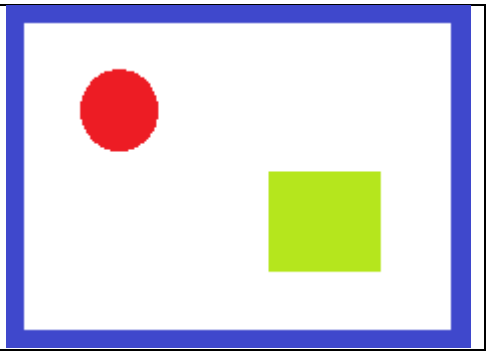
```
SURFACE* surface = loadbmp("test.bmp");
SURFACE* surface_right =createsurface_from(surface, PIVOT_RIGHT)
SURFACE* surface_left =createsurface_from(surface, PIVOT_LEFT)
SURFACE* surface_180 =createsurface_from(surface, PIVOT_180)
SURFACE* surface_vflip =createsurface_from(surface, PIVOT_VFLIP)
SURFACE* surface_hflip =createsurface_from(surface, PIVOT_HFLIP)

drawsurface(surface,0,0);
drawsurface(surface_right,200,0);
drawsurface(surface_left,0,200);
```

8.42 Single frame & double frame usage example

< SINGLE FRAME >

Single frame creates only one on-screen frame.

It draws images and shapes on currently showing screen.

Showing frame		

Like above figure, a single frame is created and images and shapes are drawn to the same frame.

How to configure and use the single frame.

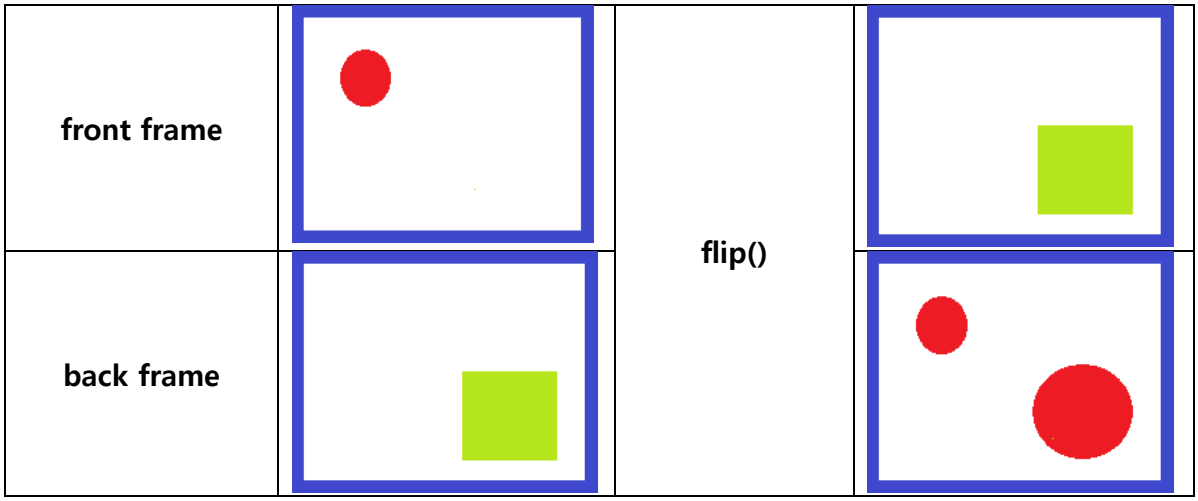
```
setscreen(SCREEN_800x480, SCREENMODE_RGB565 )
SURFACE* frame = createframe(800, 480, 16);
setframebuffer(frame);
setdrawtarget(frame);

drawcirclefill(50,50,10,MAKE_COLORREF(255,0,0));
drawrect(100,100,100,100,MAKE_COLORREF(0,255,0));
```

Creates a frame, configures the frame with **setframebuffer()** function and **setdrawtarget()** function ,and draws shapes or images.

<DOUBLE FRAME >

Double frame creates two frames. One frame is showing and the other is not on the screen. In convenience, the showing frame is called as the front frame and the other is called as the back frame.

Double frame has advantages that system can draw images and shapes on the back frame without any influences on the front frame.



As depicted in the above figure, even if the shapes are drawn on the back frame, no changes occurred on the screen (no effects on the front frame). When calling flip() function after drawing shapes on the back frame, switching between the frames occurred. As a result, the back frame is shown on the screen (becomes the front frame) and the front frame is gone back to the screen (becomes the back frame).

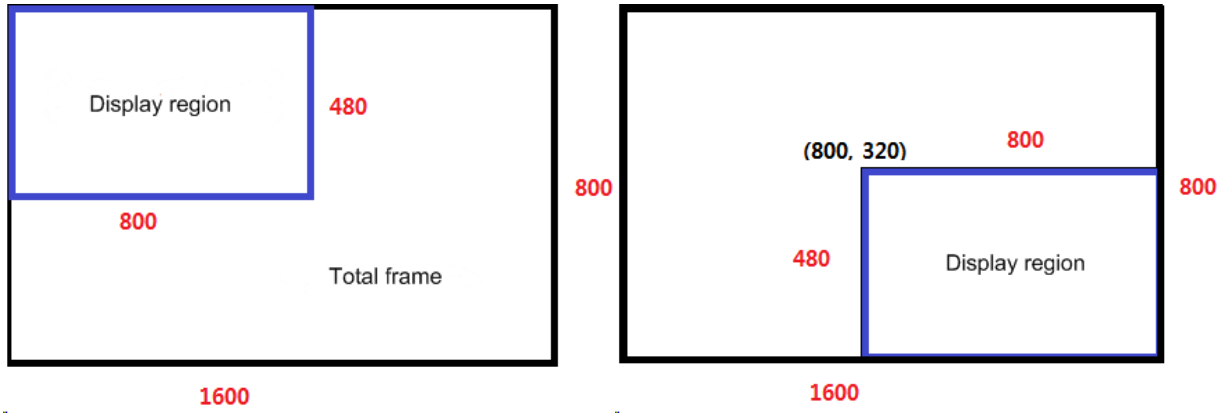
How to use the double frame.

```
setscreen(SCREEN_800x480, SCREENMODE_RGB565 )
SURFACE* frame1 = createframe(800, 480, 16);
SURFACE* frame2 = createframe(800, 480, 16);

setdoubleframebuffer(frame1, frame2);

set_draw_target(getbackframe());
draw_circlefill(50,50,10,MAKE_COLORREF(255,0,0));
flip();

set_draw_target(getbackframe());
draw_rectfill(600,300,100,100,MAKE_COLORREF(0,255,0));
flip();
```

```
set_draw_target(getbackframe());
draw_circlefill(600,300,20,MAKE_COLORREF(255,0,0));
```

Creates two frames, makes the back frame to a draw target with **setdrawtarget()** function and draws shapes or images. After that, brings the back frame on the screen by making the back frame to the front frame with **flip()** function. The front frame becomes the back frame and shapes and images are drawn on the frame with **setdrawtarget()** function that makes the frame as a draw target.

8.43 Set frame buffer xy usage example



Using setframebufferxy function, when a bigger frame than the screen size, only particular parts could be displayed on the screen as depicted in the above figure.

```
setscreen(SCREEN_800x480, SCREENMODE_RGB565);
SURFACE *frame = createframe(1600,800,16);
setframebuffer(frame);
set_draw_target(frame);
.....
.....
setframebufferxy(frame,800,320);    // Like the right side figure of the above, the parts of
                                       the frame is shown (from (800, 320) of the frame).
```

Configures the frame size is bigger than the screen resolution and chooses the frame that has bigger resolution than the screen resolution with **setframebuffer( )** function. After that, configures the origin of the parts to be shown then the frame is shown on the screen from the origin

.

8.44 Graphic Example

< SINGLE FRAME >

```
#include "adStar.h"

int main()
{
        boardinit();
        uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
        debugstring("================================================₩r₩n");
        debugprintf("ADSTAR Graphic.System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
        debugstring("================================================₩r₩n");

          crtc_clock_init();    // Configures CRT Controller.

          setscreen(SCREEN_800x480,SCREENMODE_RGB565);// Configures resolution and draw mode.
          SURFACE *frame = createframe(800,480,16);// Generates a frame.
          setframebuffer(frame);    // Selects a frame to display on a screen.
          setfrawtarget(frame);// Selets a frame to drow image or shapes.

          lcdon();
          draw_rectfill(0,0,getscreenwidth(),getscreenheight(),MAKE_COLORREF(255,255,255));
                              // After clears screen, drows white rectangle from 0,0 to 800,480.

          draw_line(30,30,100,100,MAKE_COLORREF(255,0,0));
                                              // Drows line from 30,30 to 100,100.
          draw_rect(10,10,100,100,MAKE_COLORREF(255,0,0));
                                              // Drows rectangle from 10,10 with hight 100 and width 100.
          draw_circle(100,100,10,MAKE_COLORREF(255,0,0));
                                             // Drows a circle that its origin is 100,100 and a radius is 10.
          draw_circlefill(200,200,20,MAKE_COLORREF(255,0,0));
                                           // Drows a filled circle that its origin is 200,200 and a radius is 20.
          draw_roundrect(50,50,100,100,5,MAKE_COLORREF(255,0,0));
                                // Drows a rounded rectangle from 50,50 with hight is 100 and width is 100.
          draw_ellipse (300, 100, 20, 40,MAKE_COLORREF(255,0,0));
           // Drows an ellipse that its origin is 300,100 and radius of x-axis and y-axis are 20 and 40,
              respectively.
```

```
        while(1);
        return 0;
}
```

< DOUBLE FRAME >

```
#include "adStar.h"

int main()
{
    boardinit();
    uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
    debugstring("================================================₩r₩n");
    debugprintf("ADSTAR Graphic.System Clock(%dMhz)₩r₩n",GetAHBclock()/1000000);
    debugstring("================================================₩r₩n");

    FATFS fs;
    f_mount(DRIVE_NAND, &fs);

      crtc_clock_init();   //Set CRT Controller.
      setscreen(SCREEN_800x480,SCREENMODE_RGB565);//Set resolution and the draw mode
      SURFACE *frame1 = createframe(800,480,16);          //Create frame1.
      SURFACE *frame2 = createframe(800,480,16);          //Create frame2.
                                          // Creates 2 frames in order to use double frame.
      setdoubleframebuffer(frame1,frame2);            //Configure as double frame
      set_draw_target(getbackframe());                  // Set the draw target as the back frame.

      lcdon();
      SURFACE* image1 = loadbmp("test1.bmp");   // load test1.bmp file from Nand Flash.
      draw_surface(image1,0,0);                    // Display the bmp image on (0, 0). (back frame)
      flip();                                 // switch over frames, show the back frame on the screen.

      set_draw_target(getbackframe());// Set the draw target as the back frame

      SURFACE* image2 = loadjpg("test2.jpg");// load test2.jpg file from Nand Flash.
     draw_surface_rect(image2,0,0,100,100,200,200);// Display parts of the image from (100, 100)
                                          //with vertical/horizontal lengths 200 (both) on (0, 0).
      flip();                  // switch over frames, show the back frame on the screen (jpg image).
```

```
        release_surface(image1);            // remove not using image1 from memory.
        while(1);
         return 0;
}
```

9. Sound

This chapter explains the adStar's sound output.

adStar supports 8bit/16bit, signed/unsigned, mono/stereo WAV and MP3.

9.1 sound_init()

BOOL sound_init( )

Function to initialize a soundmixer for the sound output.

Parameter

None.

Return value

None.

9.2 sound_loadwav

WAVE* sound_loadwav(**char*** filename);

Function to load a WAV file. Creates WAVE structure by loading the WAV file and stores the structure to the memory.

Parameter

filename WAV file

Return value

WAV file information and data stored WAVE structure.

9.3 sound_loadmp3

WAVE* sound_loadmp3(**char*** filename);

Function to load a MP3 file. Creates MP3 structure by loading the MP3 file and stores the structure to the memory.

Parameter

filename MP3 file

Return value

MP3 file information and data stored MP3 structure.

9.4 sound_release

BOOL sound_release(**WAVE*** pWave);

Function to release the memory that is created by **load_soundwav()** or **load_soundmp3()** functions. It is better to return disusing memory by calling **sound_release()** function about the disusing sound, because available memory space is limited.

Parameter

pWave **sound_loadwav( )** or **sound_loadmp3( )** created WAVE structure.

Return value

TRUE(1) or FALSE(0)

9.5 sound_play

BOOL sound_play(**WAVE*** pWave);

Function to play sound that is loaded by **load_soundwav()** or **load_soundmp3().**

Parameter

pWave **sound_loadwav( )** or **sound_loadmp3( )** created WAVE structure.

Return value

TRUE(1) or FALSE(0)

➔adStar can choose 1 channel of sound output among 4 channels. Channel 0&1 are outputs via an I2S and channel 2&3 are outputs via a digital modulator. SDK configures channel 2 as a default output. In order to change this, substitute digit "2" (it indicates channel 2) of "#define SND_OUTPUT_CHANNEL 2" in libconfig.h file (include folder) to other digit.

9.6 sound_stop

BOOL sound_stop(**WAVE*** pWave)

Function to stop playing sound.

Parameter

pWave WAVE structure of playing sound.

Return value

TRUE(1) or FALSE(0)

9.7 sound_vol

void sound_vol(**U8** vol);

Function to control sound volume.

Parameter

vol volume ranges are 0 - 255.

Return value

None

9.8 sound_pause

BOOL sound_pause(**WAVE*** pWave);

Function to pause playing sound.

Parameter

pWave WAVE structure of playing sound.

Return value

TRUE(1) or FALSE(0)

9.9 sound_resume

BOOL sound_resume(**WAVE*** pWave);

Function to resume paused sound.

Parameter

pWave WAVE structure of paused sound.

Return value

TRUE(1) or FALSE(0)

9.10 sound_isplay

BOOL sound_isplay(**WAVE*** pWave);

Function to confirm whether the pWave sound is playing.

Parameter

pWave WAVE structure of the sound that is examined that it is playing or not

Return value

TRUE(1 = play) or FALSE(0 = stop)

9.11 sound_ispause

BOOL sound_ispause(**WAVE*** pWave);

Function to confirm whether the pWave sound is paused.

Parameter

pWave WAVE structure of the sound that is examined that it is paused or not.

Return value

TRUE(1 = pause) or FALSE(0 = none pause)

9.12 Sound Example

< WAV File Play >

```
#include "adStar.h"

int main()
{
	boardinit();
	uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
	debugstring("==========================================₩r₩n");
	debugprintf("ADSTAR Wave Play.System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
	debugstring("==========================================₩r₩n");

	FATFS fs;
	f_mount(DRIVE_NAND, &fs);

	sount_init();                    // sound mixer initialization for the sound output
	WAVE* sound = sount_loadwav("test.wav");    // wav file loading

	sound_play(sound);                          // wav file playing

	delayms(1000);

	If(sound_isplay(sound)) // Examine whether the wav file is playing or not after one  second
	{
		sound_stop(sound);                      // Stop if it is playing
	}

	sound_release(sound);                   // Remove no more using sound from memory.

	  ...
	  ...

	while(1);
	return 0;
}
```

< MP3 File Play >

```
#include “adStar.h”

int main()
{
	boardinit();
	uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
	debugstring("==================================================₩r₩n");
	debugprintf("ADSTAR MP3Play.System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
	debugstring("==================================================₩r₩n");

	FATFS fs;
	f_mount(DRIVE_NAND, &fs);

	sound_init();                          // sound mixer initialization for the sound output
	WAVE* sound = sount_loadmp3(“test.mp3”);     // mp3 file loading

	sound_play(sound);                                  // mp3 file playing

	delayms(1000);

	sound_pause(sound);                           // pause the sound after one second

	delayms(1000);

	If(sound_ispause(sound))               // examine whether the sound is paused or not
	{
		sound_resume(sound);                              // resume the sound if it is paused
	}

	…
	…

	while(1);
	return 0;
}
```

10. FILE SYSTEM

adStar SDK can use Nand Flash and SD card as file storage devices by adopting the file system. This chapter explains how to configure Nand Flash and SD card as file storage devices. Refer to **adStar SDK/doc** folder – fatfs manual for more detailed information about the file system (SDK/doc/fatfs/doc/00index_e.html).

10.1 f_mount

FRESULT f_mount(BYTE, FATFS*);

Function to configure Nand Flash or SD card to use as file storage devices by mounting them.

Parameter

BYTE device numbers of Nand Flash or SD Card. These are defined as;
#define DRIVE_NAND 0
#define DRIVE_SDCARD 1

FATFS device information holding structure

Result value

Result of the operations.
0 is success. Refer to fatfs manual about the others.

10.2 f_chdrive

FRESULT f_chdrive(BYTE);

Function to change drive between the mounted drives.

Parameter

BYTE device numbers of Nand Flash or SD Card.
#define DRIVE_NAND 0
#define DRIVE_SDCARD 1

Return value

Result of the operations.
0 is success. Refer to fatfs manual about the others.

10.3 f_chdir

FRESULT f_chdir(const TCHAR*);

Function to move directory on the mounted drive.

Parameter

TCHAR* directory name to move

Return value

Result of the operations.

0 is success. Refer to fatfs manual about the others.

10.4 FILE System Example

```
int main()
{
        board_init();
        uart_config(0, 115200, DATABITS_8, STOPBITS_1, UART_PARNONE );

       FATFS nand_fs, sdcard_fs;
        f_mount(DRIVE_NAND, &nand_fs);          // Nand Flash mount
        f_mount(DRIVE_SDCARD, &sdcard_fs);     // SD Card mount

       SURFACE* bmp = loadbmp("test.bmp");  // Load an image file from Nand Flash

        f_chdrive(DRIVE_SDCARD);                  // change a drive to SD Card

       WAVE* wav = sound_loadwav("test.wav");  // Load a sound file from SD Card 에

        f_chdir("font");                            // Move to font directory of SD Card.
        //f_chdir("..");                           // Move to upper directory.
        //f_chdir("/..");                          // Move to root directory.
        …
        …
       while(1)
       return 0;
}
```

11. Font

adStar SDK supports two kinds of fonts. Image font and bit type font are them; this chapter explains the functions to use the fonts.

It is easy to various types of fonts with Image font, even if it need to create font images by using a program. Bit type font could be used with already provided fonts without any font creation jobs, however it is hard to change the font types.

<< bmp font >>

11.1 create_bmpfont

EGL_FONT* create_bmpfont(**const char** *fontFile)

Function to prepare to use an bmp font by loading a font file.

Parameter

fontFile font file name

Result value

bmp font pointer.

11.2 release_bmpfont

void release_bmpfont(**EGL_FONT** *pFont)

Function to release no more using bmp font.

Parameter

pFont bmp font pointer.

Result value

None.

11.3 bmpfont_draw

int bmpfont_draw(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Draw text with bmp font.

Parameter

pFont	bmp font pointer.
x	x coordinate.
y	y coordinate.
str	string to be displayed.

Result value

width of displayed string.

11.4 bmpfont_draw_vleft

int bmpfont_draw_vleft(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Draw string that rotated 90 degrees to the left with bmp font.

Parameter

pFont	bmp font pointer.
x	x coordinate.
y	y coordinate.
str	string to be displayed.

Result value

width of displayed string.

11.5 bmpfont_draw_vright

int bmpfont_draw_vright(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Draw string that rotated 90 degrees to the right with bmp font.

Parameter

pFont	bmp font pointer.
x	x coordinate.
y	y coordinate.
str	string to be displayed.

Result value

width of displayed string.

11.6 egl_font_set_color

EGL_COLOR egl_font_set_color(**EGL_FONT** *Font, **EGL_COLOR** clr)

Change font color.

Parameter

pFont bmp font pointer.

clr Font color. Use MAKE_COLORREF(r,g,b) macro function.

Result value

Old font color.

11.7 bmpfont_makesurface

SURFACE* bmpfont_makesurface(**EGL_FONT** *pFont, **char** *text)

Function to converse frequently used characters into an image. The characters could be displayed by draw_surface function when necessary. If no more using font, have to release by release_surface() function.

Parameter

pFont bmp font pointer.

text strings to be converted into an image.

Result value

Pointer to SURFACE structure.

11.8 bmpfont_setkerning

bool bmpfont_setkerning(**EGL_FONT** *pFont, **int** k)

Function to set character spacing of the bmp font.

Parameter

pFont bmp font pointer.

k character gap

Return value

TRUE(1) or FALSE(0)

11.9 bmpfont_setautokerning

bool bmpfont_setautokerning(**EGL_FONT** *pFont, **bool** b)

Set whether character spacing same.

Parameter

pFont bmp font pointer.

b TRUE or FALSE

Return value

TRUE(1) or FALSE(0)

<< bit font >>

11.10 create_bitfont

EGL_FONT* create_bitfont( )

Function to initialize to use the bit type font.

Return value

bit font pointer.

11.11 release_bitfont

void* release_bitfont(**EGL_FONT*** pFont)

Function to release no more using bit font.

Parameter

pFont bit font pointer.

Return value

bit font pointer.

11.12 bitfont_draw

int bitfont_draw(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Function to display characters using the bit type font.

Parameter

pFont	bit font pointer.
x	x coordinate of the characters output
y	y coordinate of the characters output
str	characters to be displayed

Result value

width of displayed string.

11.13 bitfont_draw_vleft

int bitfont_draw_vleft(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Draw string that rotated 90 degrees to the left with bit font.

Parameter

pFont	bit font pointer.
x	x coordinate of the characters output.
y	y coordinate of the characters output.
str	string to be displayed.

Result value

width of displayed string.

11.14 bitfont_draw_vright

int bitfont_draw_vright(**EGL_FONT** *pFont, **int** x, **int** y, **const char** *str)

Draw string that rotated 90 degrees to the right with bit font.

Parameter

pFont	bit font pointer.
x	x coordinate of the characters output.
y	y coordinate of the characters output.

str string to be displayed.

Result value

width of displayed string.

11.15 bitfont_makesurface

SURFACE* bf_makesurface(**EGL_FONT** *pFont, **char** *str)

Function to converse frequently used characters into an image. The characters could be displayed by **drawsurface()** function when necessary. If no more using font, have to release by release_surface() function.

Parameter

pFont bit font pointer.

str characters to be converted into the image.

Return value

pointer to SURFACE structure

11.16 FONT Example

< bm font example >

```
#include "adStar.h"

int main()
{
        boardinit();
        uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
        debugstring("================================================₩r₩n");
        debugprintf("ADSTAR FONT Example. System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
        debugstring("================================================₩r₩n");

        FATFS fs;
        f_mount(DRIVE_NAND,&fs);
        f_chdrive(DRIVE_NAND);

        crtc_clock_init();
        SURFACE*  frame = createframe(800,480,16);
        setscreen(SCREEN_800x480,SCREENMODE_RGB565);
        lcdon();

        setframebuffer(frame);
        setdrawtarget(frame);
        draw_rectfill(0,0,getscreenwidth(),getscreenheight(),MAKE_COLORREF(255,255,255));

        f_chdir("font");        //move to the font folder where font file exist.
        EGL_FONT*  pFont = create_bmpfont("nanum_8bit_16px_han.fnt");
                                // set to load and use nanum_8bit_16px_han.fnt file.
        egl_font_set_color(pFont, MAKE_COLORREF(0, 0, 255));
                                // change the color of  font as blue.
        bmpfont_draw(pFont, 100,100,"Test nanum font");
                                // display the characters onto coordinate (100,100)
        SURFACE*  fontsurf = bmpfont_makesurface(pFont, "bmfont_makesurface surface");
                                //convert the characters to an image. It could be used as an image using
                                //drawsurface function.
        drawsurface(fontsurf,100,300);
```

```
                    // display the converted image onto coordinate (100,300).
        while(1);
        return 0;
}
```

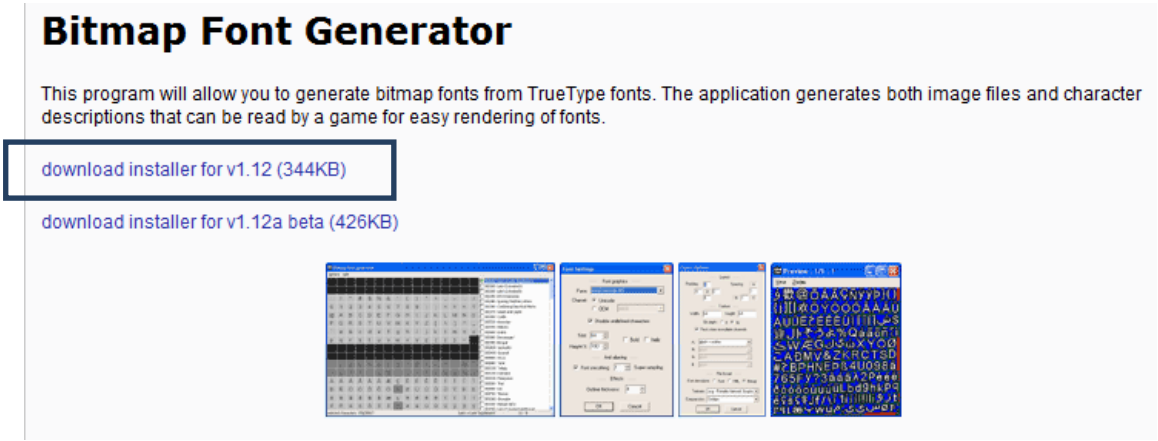
< bit font example >

```
#include "adStar.h"

int main()
{
       boardinit();
        uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);
        debugstring("================================================₩r₩n");
        debugprintf("ADSTAR FONT Example. System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
        debugstring("================================================₩r₩n");

        FATFS fs;
        f_mount(DRIVE_NAND,&fs);
        f_chdrive(DRIVE_NAND);

        crtc_clock_init();
        SURFACE*  frame = createframe(800,480,16);
        setscreen(SCREEN_800x480,SCREENMODE_RGB565);
        lcdon();

        setframebuffer(frame);
        setdrawtarget(frame);
        drawrectfill(0,0,getscreenwidth(),getscreenheight(),MAKE_COLORREF(255,255,255));

        EGL_FONT* pFont = create_bitfont();          // set to use the bit type font.
        egl_font_set_color(pFont, MAKE_COLORREF(255,0,0);
                                        // change the color of the bit type font as red.
        bitfont_draw(pFont, 100,100,"bit type font test");
                        // display the characters onto coordinate (100, 100) using the bit type font.
        SURFACE*  fontsurf = bitfont_makesurface(pFont, "bf_makesurface surface");
                    // convert the characters into an image. It could be used as an image using
                    //drawsurface function.
```

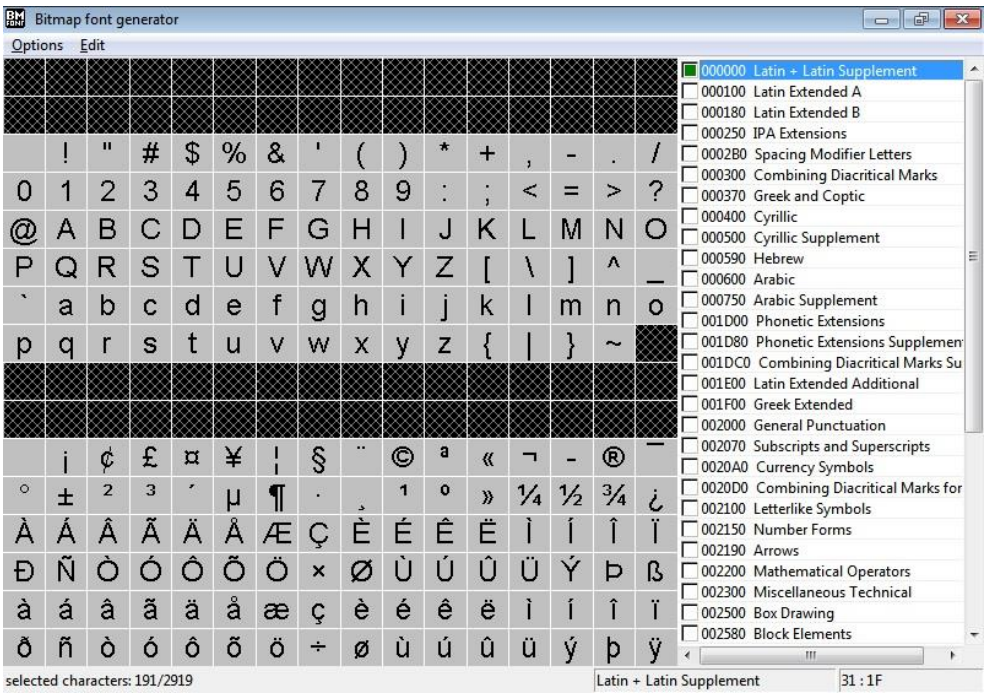
```
        drawsurface(fontsurf,100,300);
                        // display converted image onto coordinate (100, 300).
        while(1);
        return 0;
}
```

< how to create the image font >

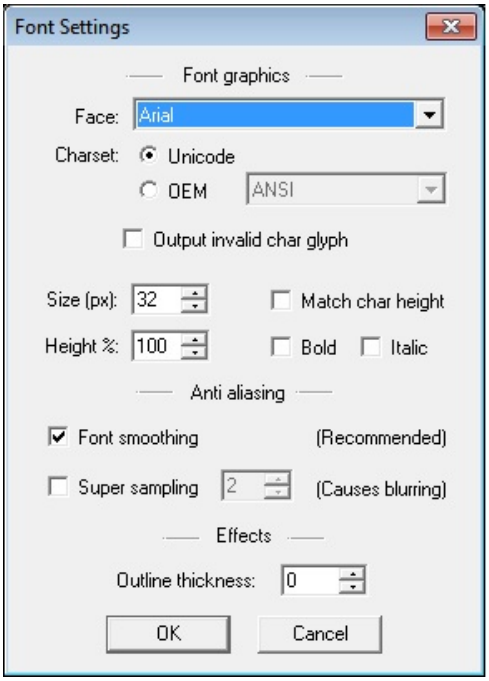
1. Download and install the Bitmap Font Generator v1.12 from http://www.angelcode.com/products/bmfont.



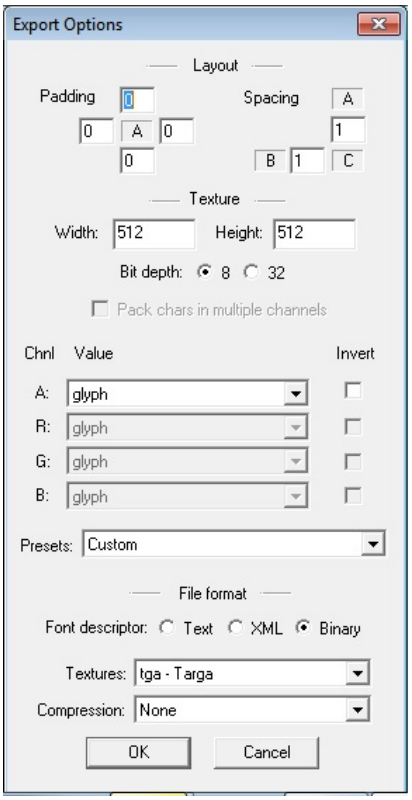
2. Next, execute the program shows up the below window.



3. Determine desired font and its size by choosing 'Options' → 'Font settings'.

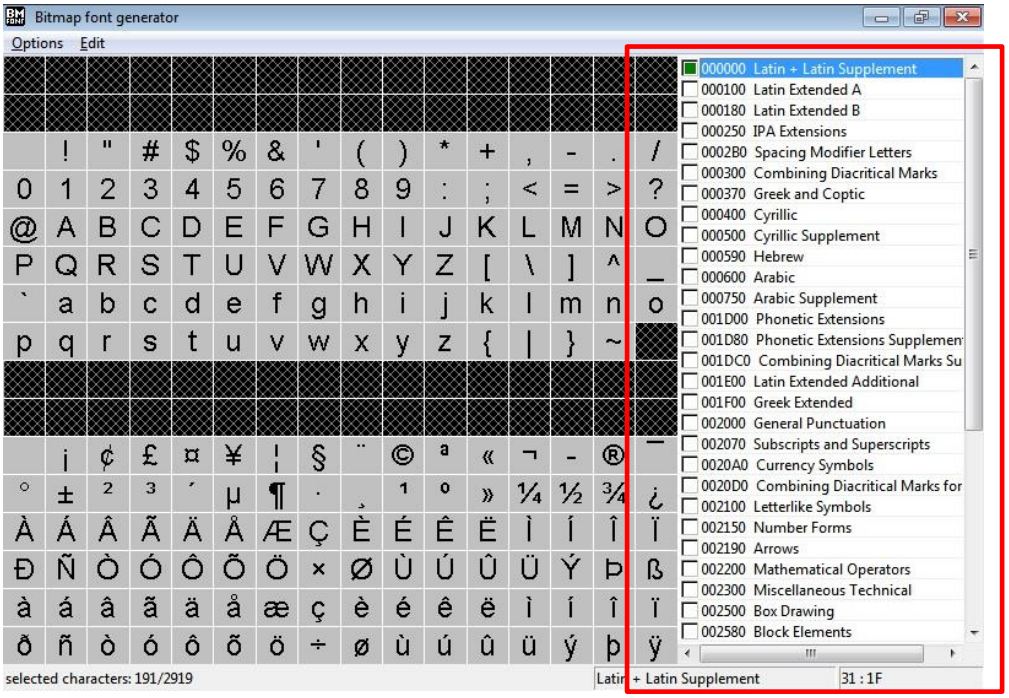


4. Set as the below window after executing 'Options' → 'Export options'.



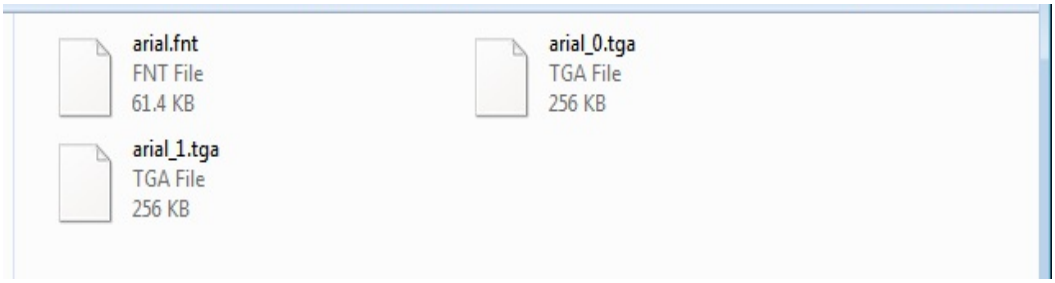
Width and Height are size of a single bitmap file's width and height. Bigger the size reduces total number of the bitmap files and wasted region could exist.

5. Check the desired font on the main window that is showed up at first execution. A font selected on the left side screen with a mouse won't be created as images.



6. Save the font with 'Option' → 'Save bitmap font as', FNAME.fnt and FNAME_xx.tga files are created. The font could be used by loading FNAME.fnt with bmfont_init function.

7. fnt → stores location information of the font and auxiliary information for displaying
tga → actual font image



12. SPI

12.1 spi_master_init

void spi_master_init(**int** ch)

Function to SPI master initial. Can select wanted channel.

Parameter

ch channel value. (0 or 1)

Result value

None.

12.2 spi_set_freq

int spi_set_freq(**int** ch, **int** mode, **U32** freq);

Function to set SPI baud rate.

Parameter

ch channel value. (0 or 1)

mode master / slave value. (SPI_MASTER / SPI_SLAVE)

freq frequency value.

Return value

Baud rate value.

12.3 spi_master_xfer

void spi_master_xfer(**int** ch, **U8** *wbuf, **int** wlength, **U8** *rbuf, **int** length, **int** continue_xfer)

Send wbuf data to the slave as many as wlength. And bring rbuf data to the master as many as rlength at slave.

Parameter

ch channel value. (0 or 1)

wbuf write data buffer.

wlength write data length.

rbuf	read data buffer.
rlength	read data length.
continue_xfer	whether transfer continue.

Return value

None.

12.4 spi_wait_empty_fifo

void spi_wait_empty_fifo(**int** ch)

Check whether tx fifo empty.

Parameter

ch	channel value.

Return value

None.

12.5 SPI Example

<SPI EEPROM Example>

=== w25xxx_flash.h ===

```
#ifndef __W25Xxx_H__
#define __W25Xxx_H__
int w25xxx_write_buffer (U32 addr, U8 *buffer, int length);
int w25xxx_read_buffer (U32 addr, U8 *buffer, int length);
int w25xxx_erase (U32 start_addr, int length);
int w25xxx_block_erase (U32 addr, int length);
int w25xxx_wait_ready (void);
int w25xxx_check_id (void);

#define W25XXX_PAGE_SIZE        (256)
#define W25XXX_SECTOR_SIZE      (4096)
#define W25XXX_BLOCK_SIZE       (16*W25XXX_SECTOR_SIZE)

#define W25X16_TOTAL_SIZE       (2*1024*1024)
#define W25X32_TOTAL_SIZE       (64*W25XXX_BLOCK_SIZE)
```

```
#define W25X64_TOTAL_SIZE       (128*W25XXX_BLOCK_SIZE)
#define W25X32_FLASH_MAXADDR    W25X32_TOTAL_SIZE

// W25Xxx: 3.3V -> Max ??MHz
 #define W25Xxx_SPI_FREQ    23000000    // 23MHz

 // W25Xxx Flash instruction
 #define W25Xxx_WREN         0x06    // Write Enable
 #define W25Xxx_WRDI         0x04    // Write Disable
 #define W25Xxx_RDSR         0x05    // Read Status Register
//#define W25Xxx_WRSR          0x01    // Write Status Register
 #define W25Xxx_READ         0x03     // Read Data Bytes
 #define W25Xxx_FAST_READ    0x0B    // Read Data Bytes at Higher Speed
 #define W25Xxx_DUAL_READ    0x3B    // Read Data Bytes at Higher Speed
 #define W25Xxx_Page_Program       0x02    // Page Program
 #define W25Xxx_BERASE       0xD8    // Block Erase
 #define W25Xxx_SERASE       0x20    // Sector Erase
 #define W25Xxx_CERASE       0xC7    // Sector Erase
 #define W25Xxx_READID       0x90    // Read ID /* ID: 0x00, 0xEF, (0x14, 0x15, 0x16) */
 #define W25Xxx_READJEDECID  0x9F    // Read ID /* ID: 0x00, 0xEF, (0x14, 0x15, 0x16) */

 #define W25XXX_MAKERID      0xEF
 #define W25X16_DEVID        0x14
 #define W25X32_DEVID        0x15

 #define W25Xxx_PDOWN     0xB9    // Power-down
 #define W25Xxx_PON          0xAB    // Release Power-down / Device ID

 #define NORMAL            W25Xxx_READ
 #define FAST              W25Xxx_FAST_READ
 #define BLOCK             W25Xxx_BERASE
 #define SECTOR            W25Xxx_SERASE
 #define ALL               W25Xxx_CERASE

 #define W25Xxx_SR_WIP  (1 << 0)    // Wait In Progress bit
 #define W25Xxx_SR_WEL  (1 << 1)    // Wait Enable Latch bit

 #define W25Xxx_ID1     0x00 // Manufacture ID
 #define W25Xxx_ID2     0xEF // Memory Type
 #define W25X16_ID3     0x14 // Memory Capacity
 #define W25X32_ID3     0x15 // Memory Capacity
```

```
#define W25X64_ID3      0x16 // Memory Capacity

#define W25Xxx_JEDECID1      0xEF // Manufacture ID
#define W25Xxx_JEDECID2      0x30 // Memory Type
#define W25X16_JEDECID3      0x15 // Memory Capacity
#define W25X32_JEDECID3      0x16 // Memory Capacity
#define W25X64_JEDECID3      0x17 // Memory Capacity

#endif //__W25Xxx_H__
```

=== w25xxx_flash.c ===

```
#include "adstar.h"
#include "w25xxx_flash.h"
#include "spi.h"

int w25xxx_write_buffer (U32 addr, U8 *buffer, int length)
{
    int wlen;
    U8 wbuf[4];
    int ret = 0;
        int i;

        U32 startaddr = addr;
        U32 endaddr = addr+length-1;
        U32 startpage;
        U32 endpage;

        startpage = startaddr/W25XXX_PAGE_SIZE;
        endpage = endaddr/W25XXX_PAGE_SIZE;

        for(i=startpage;i<endpage+1;i++)
        {
                U32 offset = addr%W25XXX_PAGE_SIZE;
                if(offset + length > W25XXX_PAGE_SIZE)
                {
                        wlen = W25XXX_PAGE_SIZE  - offset;
                }
                else
                {
                        wlen = length;
```

```
                }

                asm("clr 13");

                wbuf[0] = W25Xxx_WREN;
                spi_master_xfer(SPI_CH1,wbuf, 1, NULL, 0, 0);
                ret = w25xxx_wait_ready();
                if (ret != 0)
                        break;

                wbuf[0] = W25Xxx_Page_Program;

                wbuf[1] = (addr>>16);
                wbuf[2] = (addr>>8);
                wbuf[3] = addr;

                spi_master_xfer(SPI_CH1,wbuf, 4, NULL, 0, 1); // Write Command & Address
                                                              // Continue to write data next
                spi_master_xfer(SPI_CH1,buffer, wlen, NULL, 0, 0); // Write Data

                /*  Write Cycle Time of W25Xxx FLASH: 3.3V -> 11ms, 1.2ms */
                ret = w25xxx_wait_ready();
                if (ret != 0)
                        break;

                buffer += wlen;
                addr += wlen;
                length -=wlen;

                wbuf[0] = W25Xxx_WRDI;
                spi_master_xfer(SPI_CH1,wbuf, 1, NULL, 0, 0);
                ret = w25xxx_wait_ready();
                if (ret != 0)
                        break;

                asm("set 13");
        }
        return ret;
}
```

```
/*
 * mode: NORMAL, FAST
 */
int w25xxx_read_buffer (U32 addr, U8 *buffer, int length)
{
    int ret = 0;
    U8 wbuf[4];

        wbuf[0] = W25Xxx_READ;
        wbuf[1] = (addr>>16);
        wbuf[2] = (addr>>8);
        wbuf[3] = addr;
        asm("clr 13");
        spi_master_xfer(SPI_CH1,wbuf, 4, buffer, length, 0);

        ret = w25xxx_wait_ready();
        asm("set 13");

    return ret;
}

/*
 * SECTOR erase
 */
int w25xxx_erase (U32 addr, int length)
{
    U8 wbuf[4];
    int ret = 0;
    unsigned int esize = 0;
    unsigned int sector_size = W25XXX_SECTOR_SIZE;
    unsigned int block_size = W25XXX_BLOCK_SIZE;

    length += sector_size - 1;
    length &= ~(sector_size - 1);

    addr &= ~(sector_size - 1);

    while (length) {

        if (length >= block_size) {
            esize = block_size;
```

```
        }
        else if (length >= sector_size) {
            esize = sector_size;
        }

        wbuf[0] = W25Xxx_WREN;
        spi_master_xfer(SPI_CH1, wbuf, 1, NULL, 0, 0);
        ret = w25xxx_wait_ready();
        if (ret != 0)
            break;

        if (esize == block_size) {
            wbuf[0] = W25Xxx_BERASE;
        }
        else if (esize == sector_size) {
            wbuf[0] = W25Xxx_SERASE;
        }

        wbuf[1] = (addr>>16);
        wbuf[2] = (addr>>8);
        wbuf[3] = addr;

        spi_master_xfer(SPI_CH1, wbuf, 4, NULL, 0, 0);

        ret = w25xxx_wait_ready();
        if (ret != 0)
            break;

        addr += esize;
        length -= esize;

        wbuf[0] = W25Xxx_WRDI;
        spi_master_xfer(SPI_CH1,wbuf, 1, NULL, 0, 0);
        ret = w25xxx_wait_ready();
        if (ret != 0)
            break;
    }
    return 0;
}

int w25xxx_wait_ready (void)
```

```
{
    U8 inst = W25Xxx_RDSR;
    U8 status;
    int delay = 0;
    int ret = 0;

    do {
        spi_master_xfer (SPI_CH1, &inst, 1, &status, 1, 0);
        delay++;
        if (delay > 0x80000) {
            ret = -1;
            break;
        }
    } while (status & W25Xxx_SR_WIP);
    return ret;
}

int w25xxx_check_id (void)
{
    U8 buf[4];

    buf[0] = W25Xxx_READID;
    buf[1] = 0;
    buf[2] = 0;
    buf[3] = 0;

    spi_master_xfer(SPI_CH1, buf, 4, buf, 2, 0);

    w25xxx_wait_ready();

    if ((buf[0] != W25XXX_MAKERID) || (buf[1] != W25X32_DEVID)) {
        debugprintf(" 0x%x ", buf[0] );
        debugprintf("0x%x ₩r₩n", buf[1] ); // W25X32_ID3
       return -1;
    }
    return 0;
}
```

=== main.c ===

```
#include "adstar.h"
#include "w25xxx_flash.h"

extern void boardinit();

void SPI_pin_init()
{
        *R_PAF4 &= ~(0xf<<12);
        *R_PAF4 |= 0x5<<12;        //spi_sck1, spi_cs1

        *R_PAF5 &= ~(0xf<<0);
        *R_PAF5 |= 0x5<<0;       //spi_miso1, spi_mosi1
}

int main()
{
        boardinit();
        uart_config(0,115200,DATABITS_8,STOPBITS_1,UART_PARNONE);

        debugstring("================================================₩r₩n");
        debugprintf("ADSTAR SPI_Flash.System Clock(%dMhz)₩r₩n",get_ahb_clock()/1000000);
        debugstring("================================================₩r₩n");

        SPI_pin_init();

        int ret;

        spi_master_init(SPI_CH1);
        int baud = spi_set_freq(SPI_CH1,SPI_MASTER, W25Xxx_SPI_FREQ);
        debugprintf("SPI SCK : %d KHz₩n₩r", SPI_SCK(baud)/1000 );

        ret = w25xxx_check_id();
    if (ret != 0) {
        debugstring("W25Xxx ID Check Fail₩r₩n!₩n");
                while(1);
    }
        U8 rbuf[256];
        U8 wbuf[256];
        int i;
```

```
	for(i=0;i<256;i++)
		wbuf[i]=i;
	for(i=0;i<256;i++)
		rbuf[i]=0;

	w25xxx_erase(0,W25XXX_SECTOR_SIZE);
	w25xxx_write_buffer(0,wbuf,256);
	w25xxx_read_buffer(0,rbuf,256);
	for(i=0;i<256;i++)
	{
		if(wbuf[i]!=rbuf[i])
		{
			debugprintf("wbuf[%d] = %d  !=  rbuf[%d] = %d ₩r₩n",i,wbuf[i],i,rbuf[i]);
			debugstring("Test Error₩r₩n");
			while(1);
		}
		else
		{
			debugprintf("wbuf[%d] = %d  ==  rbuf[%d] = %d ₩r₩n",i,wbuf[i],i,rbuf[i]);
		}
	}
	while(1);
	return 0;
}
```

13. Debugging

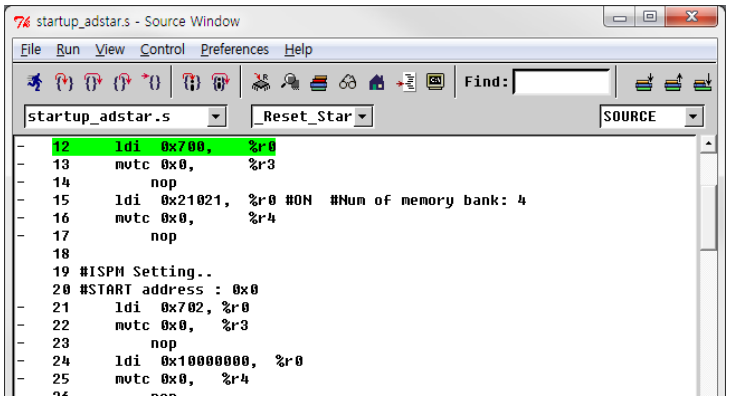
Open project to debugging and run project → properties. Set true on Build Uption → Tool Chain → Debug Info(-g). Set None(-O0) on Optimize Level(-O0). Run Build → Rebuild Project.

There is two case about debugging.

1. Case that start program at 0 without bootloader. (Use adstar.ld as Linker Script)
2. Case that start program at Ram using bootloader. (Use adstar_ram.ld as Linker Script)

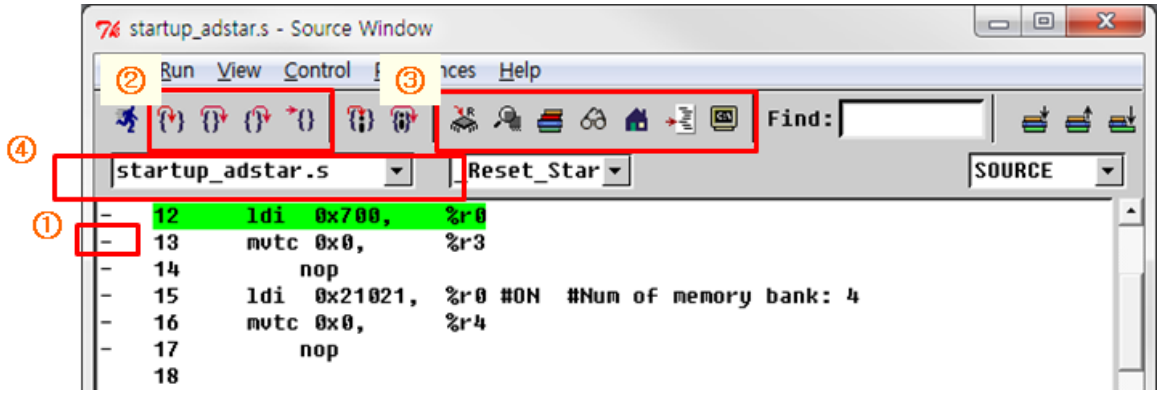
➔ Case that start program at 0

1) binary file download to 0 in flash.
2) After set debug mode (SW3 up), connect ECon and power on.
3) Run Debug → Start Debugger (F5)
4) When appeared following window, you can start debugging.



If the green code line is shown like above image, it means the connection is success.

5) Explain Debugger window



① : When Click '-', break point set / release

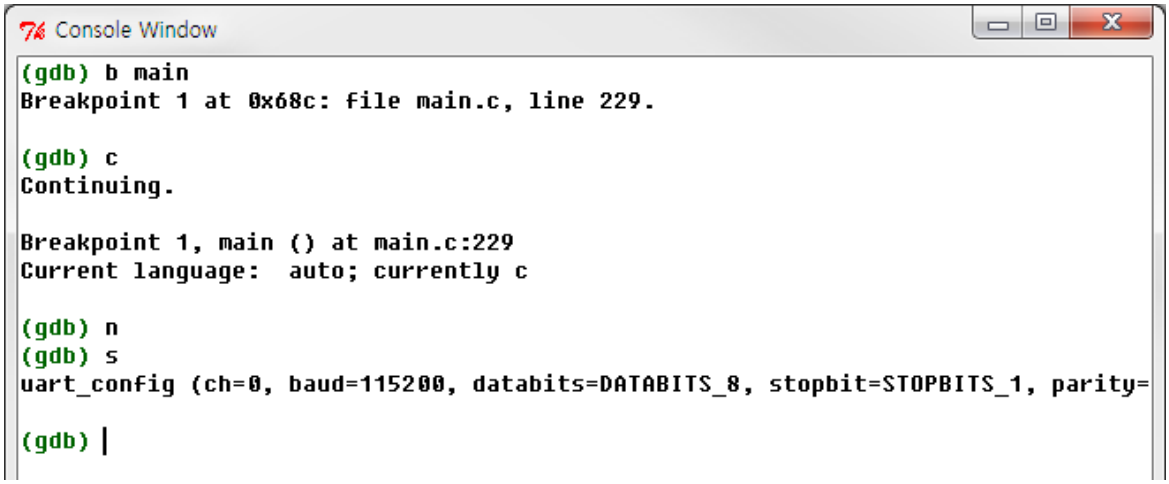
② : From the front, 'step', 'next', 'finish', 'continue'

③ : From the front, display 'register', 'memory', 'stack', 'watch expressions', 'local variable'. Last 'c: ' is that displaying 'console window.

Do debugging by using GDB command on console window.

④ : Move source code.

6) Debugging on console window.



b : Set break point.

c : Continue command. Run to next break point.

n : Next command. Run one code.

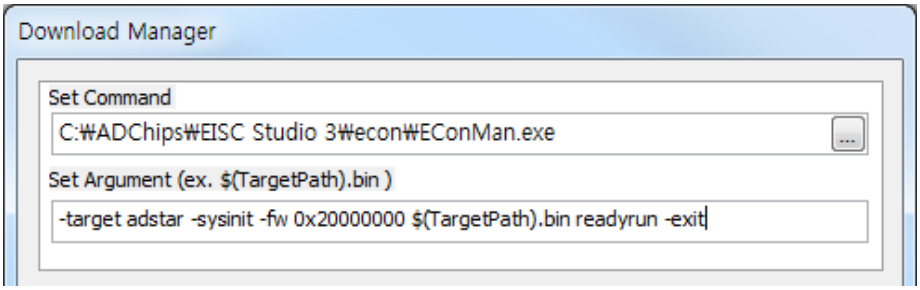
s : Step command. Run one code.

q : exit.

➔ Case that start program at Ram

1) After set debug mode (SW3 up), connect ECon and power on.

2) Run Build → Download Option and type argument like following.



-target adstar -sysinit -fw 0x20000000 $(TargetPath).bin readyrun -exit

3) After write download option, run Debug → Start Debugger (F5).
If the green code line is shown like following image, it means the connection is success. Start debugging like previous.