AM OpenAT Development Manual

Ver.	Date	Content	Author	Comment
V0.1	2012-12-20	Created	Lifei	
V0.2	2012-12-26	1. Add 8. FAQ	Lifei	

Table of Content

1. General

AM OpenAT platform is a completely open development environment. The platform program decides to start customer program or not by verifying the contents of customer program area. Association between platform program and customer program is established by several header files. The customer program can be complied, downloaded and debugged individually, without replying on the library files of platform.

Main characteristics of AM OpenAT platform include:

1. AT command is used for communication and related functions, to ensure that the customer-developed programs based on AT can be transplanted fast.

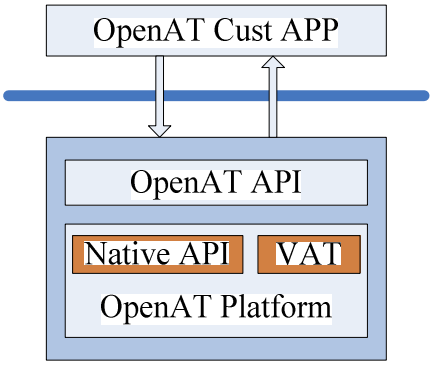
2. The system and peripheral drivers are provided through Native API, which has perfect functions and helps customer to develop complex applications.

3. It allows customers to start multiple threads. It can replace the external MCU of customer for cost-reducing purpose.

4. Smooth transplantation and individual compilation for customer’s codes. The platform release package is very concise.

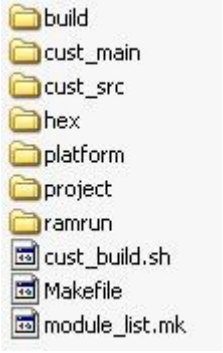
2. Software framework

The software framework of AM OpenAT open platform is as follows:



3. Project directory structure

The project directory released by AM OpenAT open platform is shown in the following figure:



build: Temporary files directories generated during compilation of customer programs.

cust_main: Demo code of customer main code, as a part of demo project for customer's reference. Customer can transplant the realization of this file according to their requirements.

cust_src: Directory for customer's source code.

hex: Directory for customer's compiled executable file.

platform: Platform file directory. Normally, customer doesn't need to modify the files in this directory.

project: Directory for control files of compilation for customer's projects. It allows one set of codes to generate the executive files for different projects.

ramrun: Download driver, lod files in which will be used as described in chapter 5.1.

cust_build.sh & Makefile & module_list.mk: Compilation control file.

4. Compilation of customer's projects

4.1 Installing compiling environment

The latest installation package of compiling environment for RDA platform is:

CSDTK3.7_Cygwin1.5.25_Svn_1.5.4_Full_Setup.rar

Please ask FAE for the latest installation package.

After installation, check "start->programs", see as below:

4.2 Source code transplantation and compilation

Before customer's codes are compiled, the following work should be done:

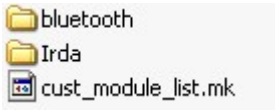
1. Put customer's codes in cust_src directory, and before that, write or rewrite the source code according to AM OpenAT API.

2. Makefile modification

2.1 Modify cust_src/cust_module_list.mk

The file mainly lists the modules to be compiled by customer, which can be easily configured according to different projects and functions.

[eg.] If in cust_src directory there are two modules, see as below:



The cust_module_list.mk after modification is as follows:

```
1
2 ifeq "${IRDA_SUPPORT}" "TRUE"          <<-- Note 1
3 LOCAL_MODULE_DEPENDS += cust_src/Irda <<-- Note 2
4 endif
5
6 ifdef PROJECT_FOR_xxx_CUST    <<-- Note 3
7 ifeq "${BT_SUPPORT}" "TRUE"   <<-- Note 4
8 LOCAL_MODULE_DEPENDS += cust_src/bluetooth
9 endif
10 endif
11
12
```

Note 1: If support IRDA function, this module will be compiled.

Note 2: Add IRDA module directory.

Note 3: If compiled PROJECT_FOR_xxx_CUST, Bluetooth will be compiled.

Note 4: If support Bluetooth function, this module will be compiled.

2.2 Add Makefile for new module

The Makefile of module is mainly for controlling the compilation of this module, which can control compile options, etc.

This Makefile can be modified with cust_main/Makefile as template, for the detailed modifying contents please refer to the following figure:

```
#************************************************************#
# Copyright (C), AirM2M Tech. Co., Ltd.
# Author: lifei
# Description: AMOPENAT 开放平台
# Others:
# History:
#   Version.  Date:       Author:   Modification:
#   V0.1      2012.12.14  lifei     创建文件
#************************************************************#

# Name of the module, with toplevel path, e.g. "phy/tests/dishwasher"
LOCAL_NAME := cust_main <<<<<--------------------Note 1

# Space-separated list of modules (libraries) your module depends upon.
# These should include the toplevel name, e.g. "phy/dishes ciitech/hotwater"
LOCAL_DEPEND_LIBS :=

# Add includes from other modules we do not wish to link to
LOCAL_API_DEPENDS := \

LOCAL_ADD_INCLUDE := \                          <<<<--------Note 2
                     platform/std_inc \
                     platform/OpenAT_inc \

# Set this to any non-null string to signal a module which
# generates a binary (must contain a "main" entry point).
# If left null, only a library will be generated.
IS_ENTRY_POINT := no

## ------------------------------------ ##
##      Add your custom flags here          ##
## ------------------------------------ ##
MYCFLAGS +=  <<<<------------------------------------------- Note 3

## ------------------------------------ ##
##      List all your sources here            ##
## ------------------------------------ ##
C_SRC := ${notdir ${wildcard src/*.c}} <<<<------------ Note 4

## ------------------------------------ ##
##  Do Not touch below this line         ##
## ------------------------------------ ##
include ${SOFT_WORKDIR}/platform/compilation/cust_rules.mk <<<<-- Note 5
```

Note 1: Must modify for your own module. It is same with “LOCAL_MODULE_DEPENDS +=” in cust_src/cust_module_list.mk.

Note 2: Add include path for GCC complier.

Note 3: Add MACRO defining for GCC complier.

Note 4: Source code list, default it will search files in path “src/”.

Note 5: Never modify this line.

2.3 Modify the parameters of platform

Platform control parameters are located in project/xxx /target/target.def. Currently, the parameters which can be configured are chip information and FLASH type.

The corresponding parameter for A6300A module is: 8809/flsh_spi32m。

The corresponding parameter for A6300V is: 8809/flsh_spi16m。

Note:xxx is the project name of customers, the default project is DemoProject

4.3 Start compilation

Double click project/xxx /build/cmd.exe, and run cust_build.bat.

Errors during compilation will be output to cmd window, and meanwhile, to log (log is located at build/xxx_build.log).

After compilation, download program for customer is generated, located in the directory “hex/xxx”.

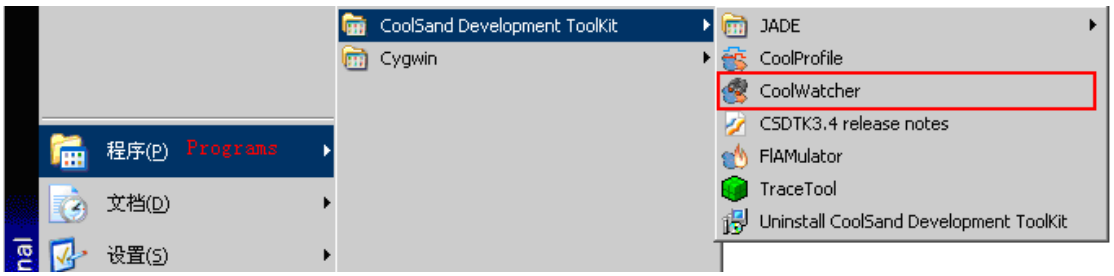
5. Download of customer program

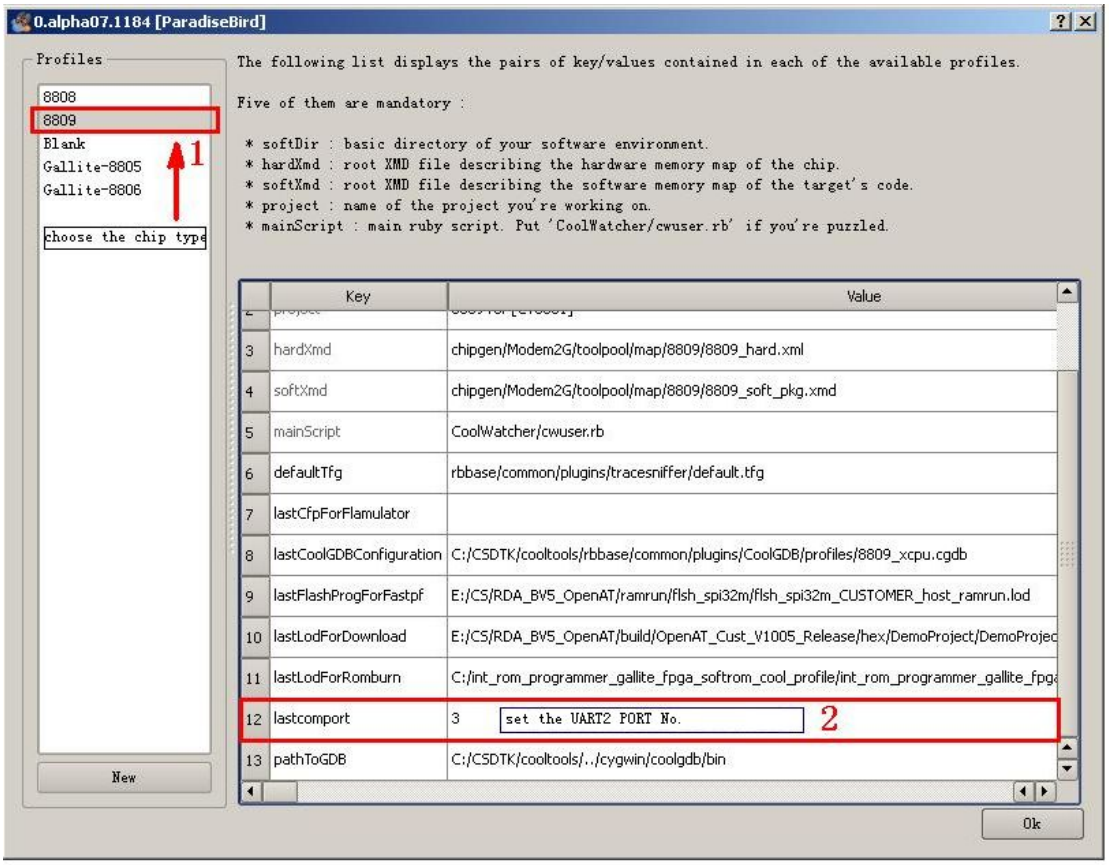
Please pay more attention! Before download of customer program, please make sure that the corresponding version of the platform program has been downloaded to module or phone, otherwise, customer program may be failed or questions that are difficult to be located may be caused.

Two types of download:

5.1 coolwatcher

1.1 Start coolwatcher





Note: if coolwatcher is started for the first time, ruby environment will be installed automatically, please wait for a while.

1.2 Download setting

Select the customer program to be downloaded, located in the directory “hex/xxx (project name of customer)”. For the Demo project, the download file is “hex/DemoProject/DemoProject_flash.lod”.


Select download driver “lod file”, which has relations with FLASH type of platform and is located in “ramrun/” directory. Now there are two driver lod files, flsh_spi16m and flsh_spi32m.

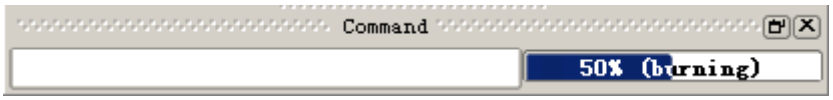
Module A6300A uses the lod file in directory “flsh_spi32m”.

Module A6300V uses the lod file in directory “flsh_spi16m”.


Start download. Before download, module or phone should be powered on and switched on. After download is started successfully, the table window “Ruby Script” of coolwatcher will show the following output:

```
> fastpfGo()
Fastpf V2 over Host
Loading the lod files:
DemoProject_flash.lod
Using flash programmer:
flsh_spi32m_CUSTOMER_host_ramrun.lod
Ramrunning the flash programmer...
Entered Host Monitor mode.

Configuring EBC RAM ...
Done.

0.000000 0.000000 0.000000 ( 0.218000)

Verify enabled: true
Fastpf Protocol Version : 1.4
Fastpfing...
0.375000 0.438000 0.813000 ( 0.813000)

Verifying (2 blocks)...
Verify succeeded.
0.000000 0.031000 0.031000 ( 0.031000)
```

Meanwhile, “command” window will indicate the progress:



5.2 Download tools for customer

Not applicable.

6. Debug for customer program

Customers may check the current running state or data for trouble

shooting by log output or by interrupting program run.

6.1 log output

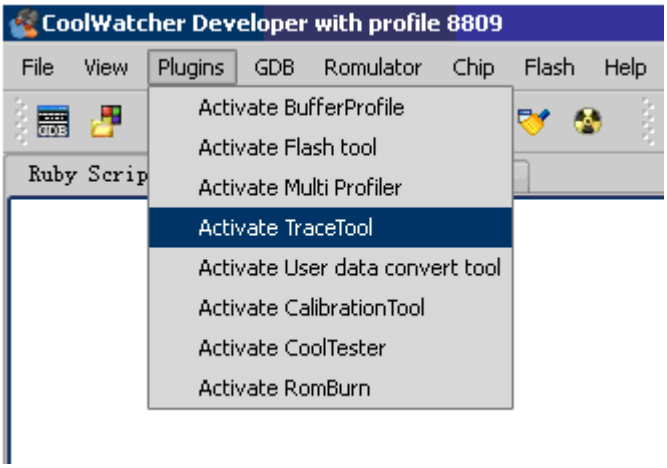
The customer program can make output log by calling the “print” function of interface functions.

Log output is cached in buffer area of module. When the system is at the status of IDLE, data in buffer area will be output by DEBUG_HOST (i.e. download port of program). If the buffer area is full with log for output, the log output by program afterwards will be lost. Since this buffer area is shared by platform program and customer program, the log losing problems can be very serious during module booting. So log output is not a reliable debugging method.

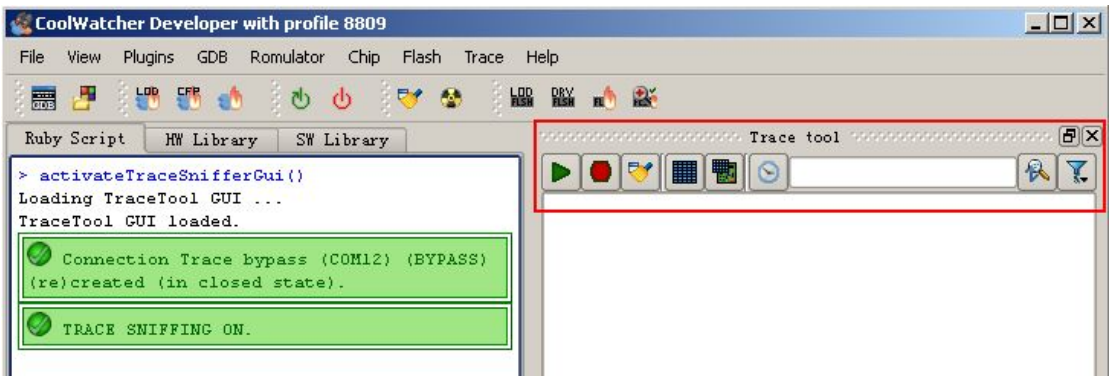
If customer wants to make reliable log output, UART interface can be used. The development for this function can be realized by customers themselves.

Logs output by DEBUG_HOST can be received by the plug-in unit “TraceTool” of coolwatcher.

Open TraceTool:



After opening TraceTool, TraceTool window will be displayed on the right side of coolwatcher.



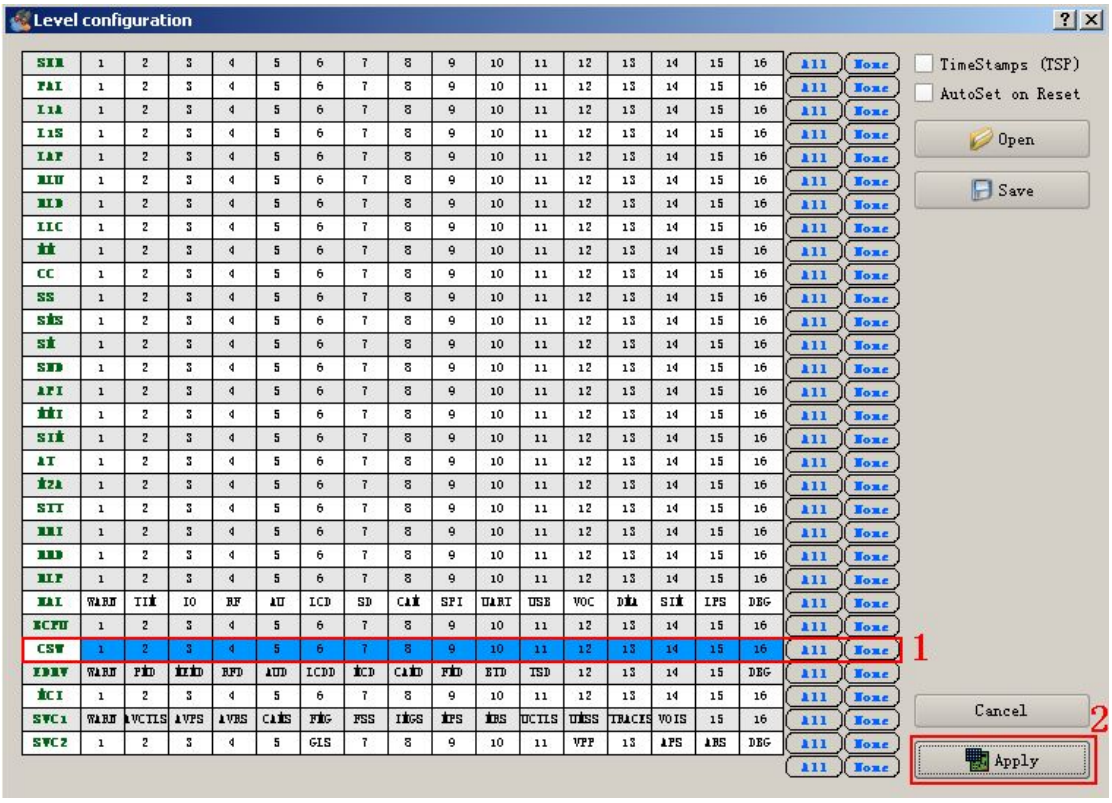
Function control for TraceTool:

 Start Trace function. After module receives the command, data in buffer area will be output. Closing this function is default, so every time after switching on, this button should be clicked again.

 Stop Trace function. After module receives the command, Trace output will stop.

 Clear the currently received Trace information.

 Configure Trace output module. Log output by print interface belongs to CSW module. If Trace output of other modules is not needed, and only customer's Trace needs checking, the following configuration can be referred to:



 Save the configuration set in  , which is set automatically every time when press  .

 Add timestamp before log information.

 log search function

6.2 Program interruption

Execution of program interruption might have many causes, for example:

1. Actively call assert interfac of platform

When customer program figures out some an illegal condition, it can call assert interface actively to interrupt the program.

In this case, module will output assert information, the figure below will be displayed in coolwatcher:

```
Assert received!! Reading detail...
ASSERT DETAIL :
Func:cust_at_message @line97
Detected CPU fatal error (0x9db00010), it went through GDB stub and will reboot.
```

The above displayed program means assert interface is called at Line 97 of the file in cust_at_message function.

By the time, coolgdb  can be used for debug, the use of coolgdb will be introduced in detail in subsequent chapters.

2. Illegal address access

When there is null pointer or other illegal address access in customer program, the chip will come across an exception and the program will be interrupted. This condition is a bit different from the first condition. Normally, the note below will be displayed in coolwatcher:

```
Detected CPU fatal error (0x9db00010), it went through GDB stub and will reboot.
```

The two conditions can be distinguished by inquiring the current system register through the register viewer in coolgdb.

3. Caused by others.

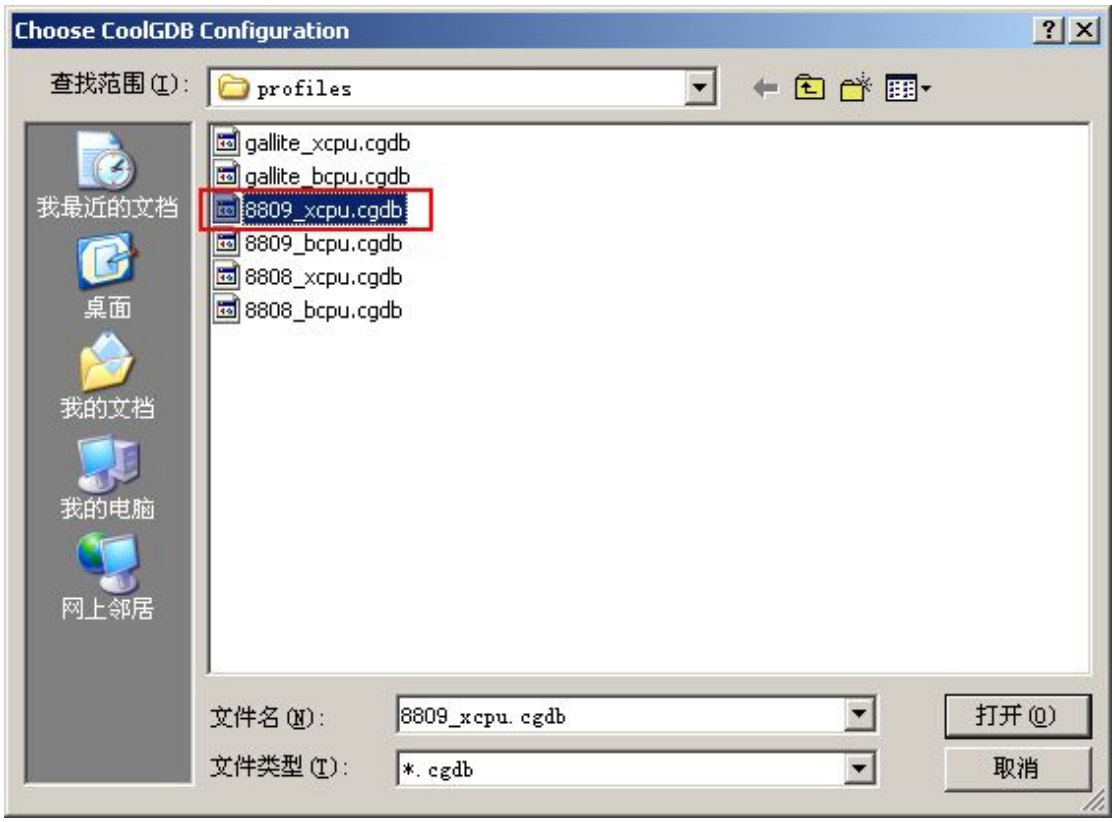
6.3 coolgdb debug tool

When program is interrupted, coolgdb can be used for checking the operation state of program.

6.3.1 Configure coolgdb

Coolgdb is configured according to the chip types of module. This configuration only needs to be made once, afterwards, the last configuration will be default.

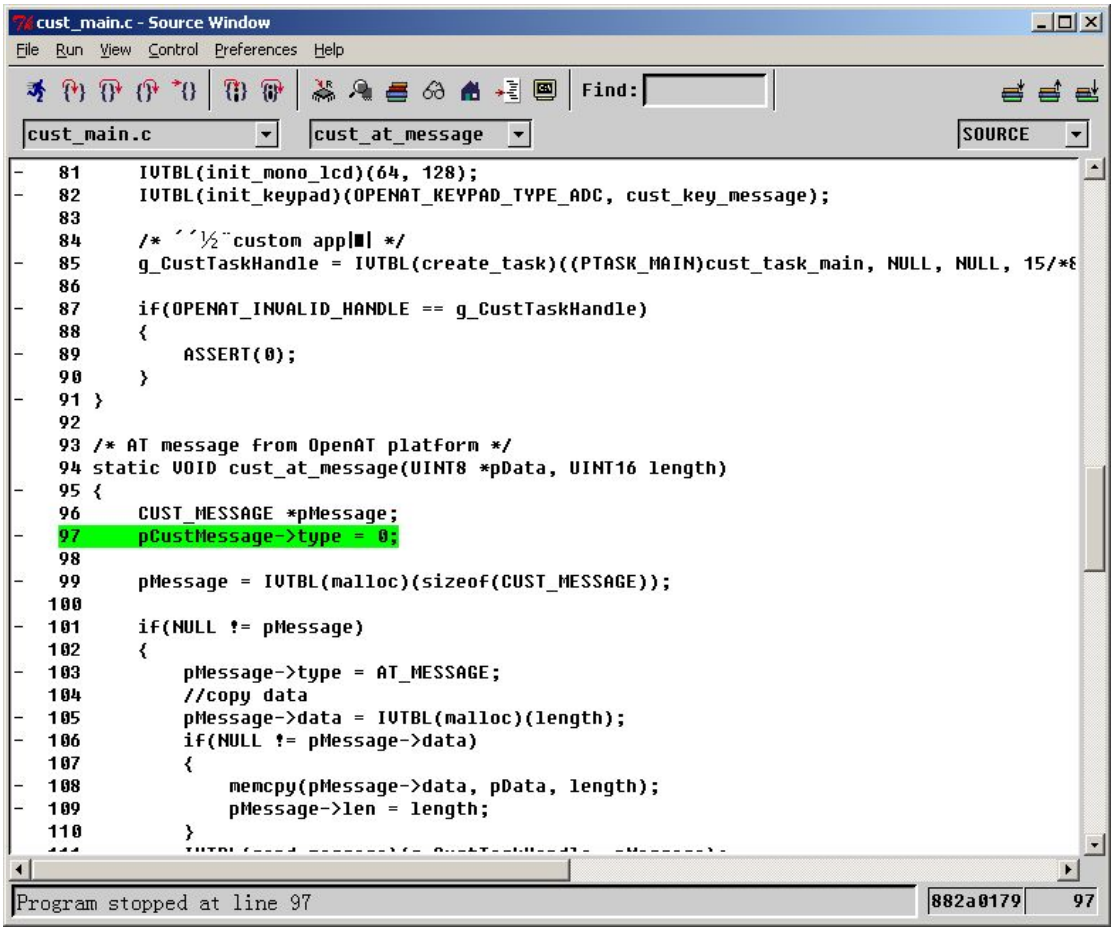
Select the corresponding configuration for chip. For A6300A or A6300V, please select 8809_xcpu.cgdb file.



6.3.2 Start coolgdb

After configuration, click  , then start coolgdb.

After starting coolgdb, main window will display. If coolgdb can be associated to source codes, main window will directly display the last row of code when interrupted, see as below:



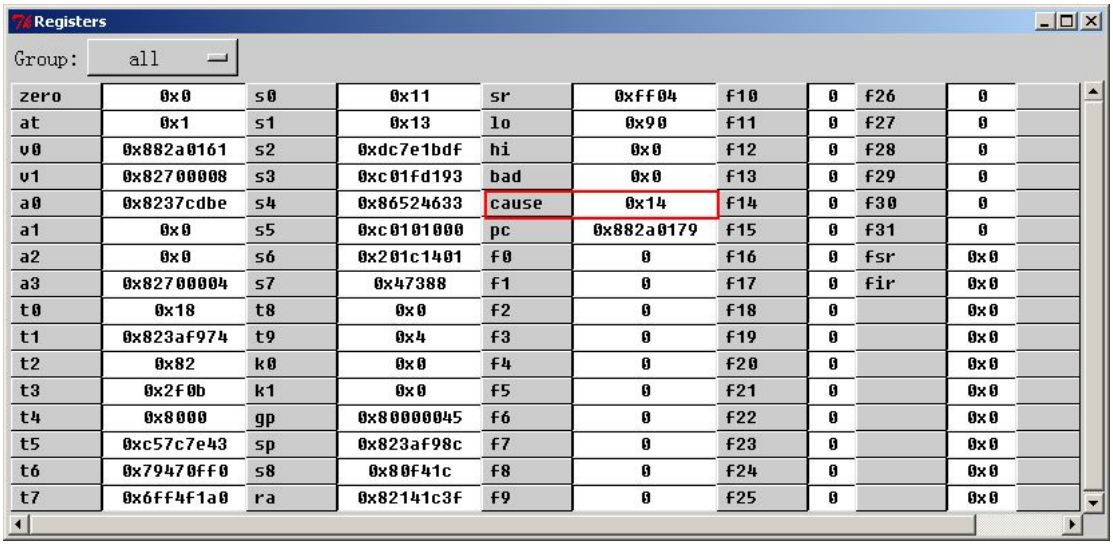
Meanwhile, status bar of windows will display the icon of CoolGDB, see as follows:

6.3.3 Use coolgdb

Different tools can be started by the function buttons on coolgdb main interface, for checking the running state of program, and such information as call stack, local variables (not optimized) and global variables.

Buttons are as below:

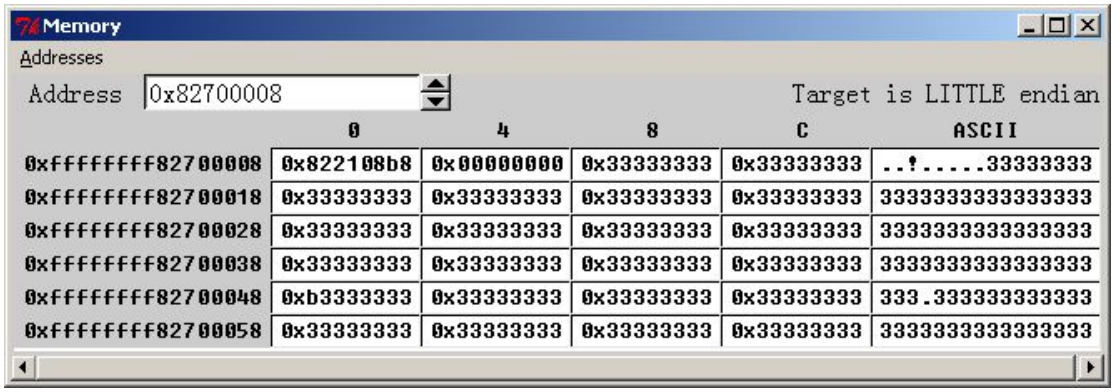
 Check the current system register. See as follows:



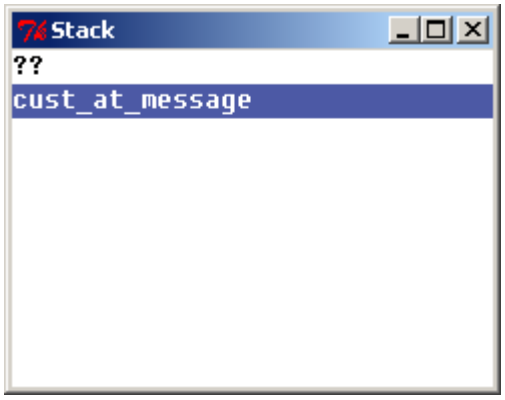
Casue register saves the cause of cpu exception, this value divided by 4 is the ExcCode indicated in the following table:

ExcCode	助记符	描述
0	Int	中断
1	Mod	存储操作时，该页在 TLB 中被标记为只读。
2	TLBL	没有 TLB 转换（读写分别）。也就是 TLB 中没有和程序地址匹配的有效入口。
3	TLBS	当根本没有匹配项（连无效的匹配项都没有）并且 CPU 尚未处于异常模式— SR(EXL) 置位—即 TLB 失效时，为高效平滑处理这种常见事件而采用的特殊异常入口点。
4	AdEL	（取数、取指或者存数时）地址错误：这要么是在用户态试图存取 kuseg 以外的空间，或者是试图从未对齐的地址读取一个双字、字或者半字。
5	AdES	
6	IBE	总线错误（取指或者读取数据）：外部硬件发出了某种出错信号；具体该怎么做与系统有关。因存储而导致的总线错，只能作为一个为了获取要写入的高速缓存行而执行的高速缓存读操作的结果间接出现。
7	DBE	
8	Syscall	执行了一条 syscall 指令。
9	Bp	执行了一条 break 断点指令，由调试程序使用。
10	RI	不认识的（或者非法的）指令码。
11	CpU	试图运行一条协处理器指令，但是在 SR(CU3–0) 中并没有使能相应的协处理器。 具体说，就是当 FPU 可用位 SR(CU1) 没有置位时从浮点操作得到的异常；因而是浮点仿真开始的地方。
12	Ov	自陷形式的整数算术指令（比如说 add 但 addu 不会）导致的溢出。C 语言程序不使用溢出-自陷指令。
13	TRAP	符合了 teq 等条件自陷指令的某一条。
14		目前未用。在有些拥有 L2 高速缓存的老式的 CPU 上，当硬件探测到可能的高速缓存重影时使用这位，在 4.12 节对此有解释。
15	FPE	浮点异常。（在某些很老的 CPU 上，浮点异常以中断形式出现。）
16–17	-	定制的异常类型，与具体实现相关。
18	C2E	来自协处理器 2 的异常（如果有的话，就是对指令集的一种定制的扩展）。
19–21	-	保留给未来扩展使用
22	MDMX	试图运行 MDMX 指令，但是 SR(MX) 位没有置位（很可能该 CPU 没有实现 MDMX）。
23	Watch	load/store 的物理地址匹配了使能的 WatchLo/WatchHi 寄存器。

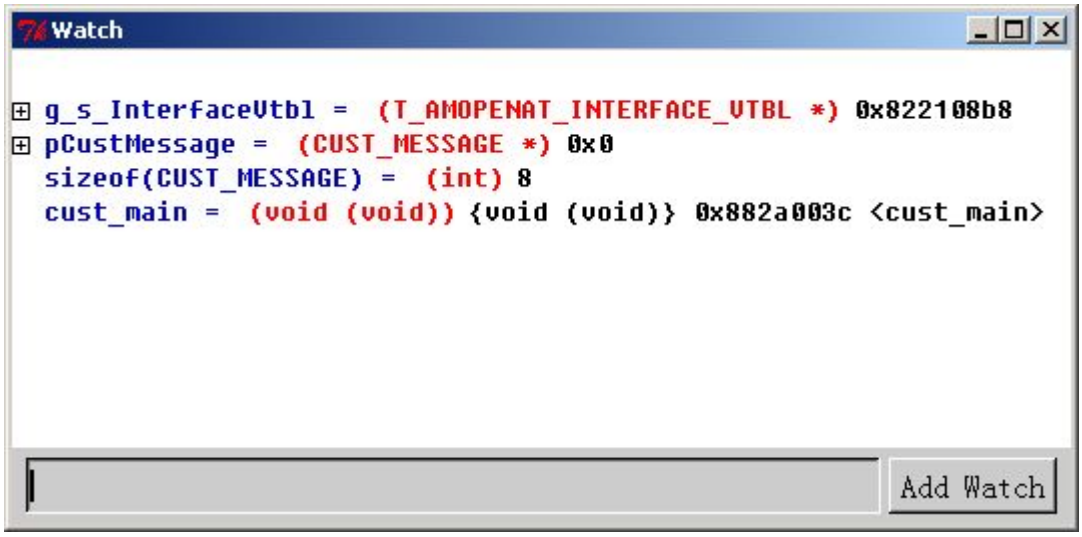
 Memory viewer, through which data in ROM and RAM can be checked. By Map file (located in hex/xxx directory) generated after compiling, the address of variables and functions in RAM or ROM can be found out. Input the address into memory viewer, press Enter, the data for corresponding address can be checked.



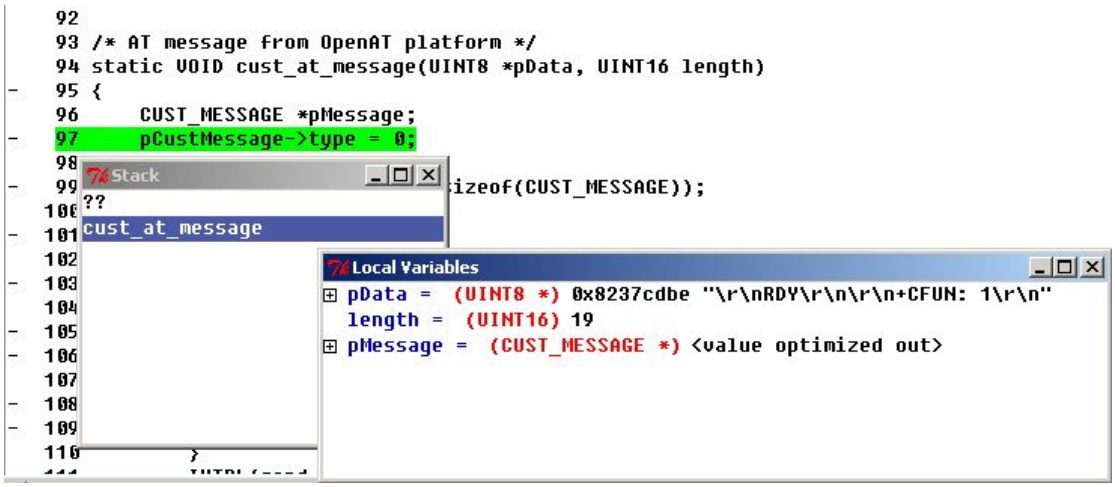
 Program call stack viewer, which can check the call stack of current program. If the function called finally is located in platform, the callstack will only shows “??”, and detailed information can’t be checked. This is because coolgdb hasn’t loaded the elf file of platform program, so it cannot analyze the call stack of functions.



 Viewer: With powerful function, it can check global variables, value of the expressions, and etc. See as below:



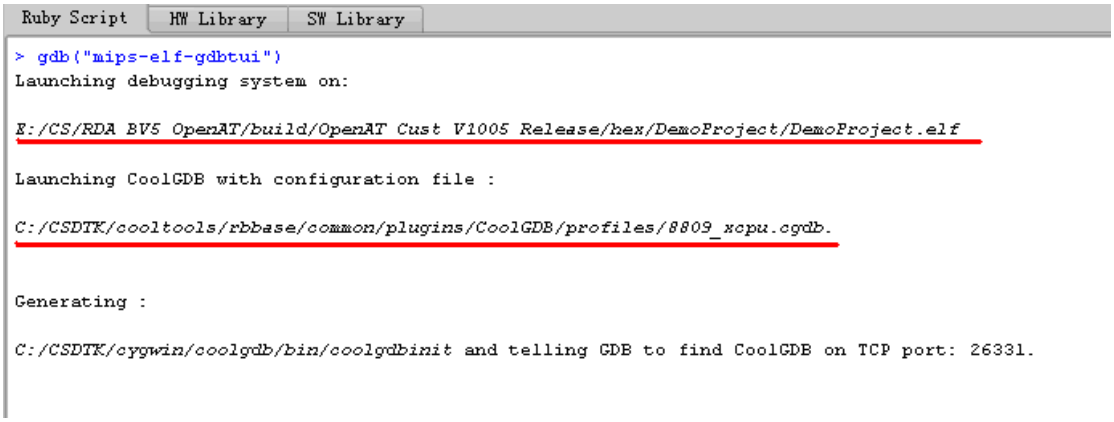
 Local variable viewer: It can check the local variable of current positioned function which hasn't been optimized.



The above figure shows the local variable of cust_at_messge function, pMessage of which has been optimized.

At the same time, please pay attention to the message output by "Ruby Script" window when coolgdb is started, which points out the loaded elf file by coolgdb. If this elf file is not that one corresponding to executive program in module or phone, all the above messages may be wrong. If the elf file loaded by coolgdb is found wrong, the correct elf file can be selected and loaded through "File->Open" menu of coolgdb main interface. Or quit coolgdb, select the lod files corresponding to module or phone's program by  button (Precondition for this is elf file and this lod file are located in the same directory, then coolgdb will load elf file from the directory where lod file is), then restart coolgdb.

Normally, the latter method is recommended.



If coolgdb is closed accidentally during debug, as long as module or phone is not restarted, coolgdb can be run again. Whereas, before restarting coolgdb, if the icon CoolGDB is still in status bar of windows, please select “Quit” from right key menu of mouse for quitting CoolGDB, otherwise coolgdb maybe work abnormally after restarted.

7. Application Programming Interfaces

Interfaces provided by platform are these header files in “platform/OpenAT_inc” directory. Specific functional modules include:

7.1 System module

This module mainly provides the thread interface, message queue interface of thread, timer interface, system interrupting close and open interface, semaphore interface, memory interface and others.

7.1.1 Thread interfaces

7.1.1.1 create_task

Description

Create a new thread.

Function prototype

```
HANDLE (*create_task)(
```

```
                        PTASK_MAIN pTaskEntry,
                        PVOID pParameter,
                        PVOID pStackAddr,
                        UINT16 nStackSize,
                        UINT8 nPriority,
                        UINT16 nCreationFlags,
                        UINT16 nTimeSlice,
                        PCHAR pTaskName
                      );
```

Parameters

1) PTASK_MAIN pTaskEntry: [IN] Thread main entry routine.

2) PVOID pParameter: [IN] Parameter passed to thread main routine.

3) PVOID pStackAddr: [IN] Start address for thread stack (NOT support yet, please set to NULL).

4) UINT16 nStackSize: [IN] Thread's stack size.

5) UINT8 nPriority: [IN] Thread priority. 0 is highest priority, and 255 is lowest priority. OPENAT_CUST_TASKS_PRIORITY_BASE is used as base priority for customer program.

6) UINT16 nCreationFlags: [IN] Set attributes for the thread. See E_AMOPENAT_OS_CREATION_FLAG for more information.

7) UINT16 nTimeSlice: [IN] NOT support yet, will be ignored.

8) PCHAR pTaskName: [IN] ANSIC string as thread name.

Return

If thread created successfully, return the handle of the thread, otherwise, return 0.

Note

7.1.1.2 start_task

Description

Start the thread.

Function prototype

```
VOID (*start_task)(
                        HANDLE hTask,
                        PVOID pParameter
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

2) PVOID pParameter: [IN] Parameter passed to thread main routine, which same with **create_task** interface's parameter "PVOID pParameter".

Return

Void.

Note

7.1.1.3 stop_task

Description

Stop the thread.

Function prototype

```
VOID (*stop_task)(
                        HANDLE hTask
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

Return

Void.

Note

7.1.1.4 delete_task

Description

Delete the thread.

Function prototype

```
BOOL (*delete_task)(
                        HANDLE hTask
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.1.5 suspend_task

Description

Suspend the thread.

Function prototype

```
BOOL (*suspend_task)(
                        HANDLE hTask
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.1.6 resume_task

Description

Resume the thread.

Function prototype

```
BOOL (*resume_task)(
                        HANDLE hTask
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.1.7 current_task

Description

Get the current thread handle.

Function prototype

```
HANDLE (*current_task)(
                        VOID
                      );
```

Parameters

Void.

Return

Handle of the current thread.

Note

7.1.1.8 get_task_info

Description

Get the thread creation information.

Function prototype

```
BOOL (*get_task_info)(
                        HANDLE hTask,
                        T_AMOPENAT_TASK_INFO *pTaskInfo
                      );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

2) T_AMOPENAT_TASK_INFO *pTaskInfo: [OUT] The information of the thread.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.2 Message queue interfaces

7.1.2.1 wait_message

Description

Waiting for a message blocking.

Function prototype

```
BOOL (*wait_message)(
                        HANDLE hTask,
                        PVOID* ppMessage,
                        UINT32 nTimeOut
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

2) PVOID* ppMessage: [OUT] The message data sending to this thread.

3) UINT32 nTimeOut: [IN] NOT support now.

Return

TRUE for successfully received one message, otherwise return FASLE.

Note

If you need a non-block interface, please use **available_message** interface for testing if there already exists a message in the queue first.

7.1.2.2 send_message

Description

Send a message to the thread message queue.

Function prototype

```
BOOL (*send_message)(
                        HANDLE hTask,
                        PVOID pMessage
                    );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

2) PVOID pMessage: [IN] The message data sending to thread.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.2.3 send_high_priority_message

Description

Send a message to the thread message queue header.

Function prototype

```
BOOL (*send_high_priority_message)(
                        HANDLE hTask,
                        PVOID pMessage
                                );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

2) PVOID pMessage: [IN] The message data sending to thread.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.2.4 available_message

Description

Check if there already exists message in the queue.

Function prototype

```
BOOL (*available_message)(
                        HANDLE hTask
                         );
```

Parameters

1) HANDLE hTask: [IN] The handle of thread, which returned by **create_task** interface.

Return

TRUE for existing message, otherwise return FASLE.

Note

7.1.3 Timer & time interfaces

7.1.3.1 create_timer

Description

Create a new timer.

Function prototype

```
HANDLE (*create_timer)(
                        PTIMER_EXPFUNC pFunc,
                        PVOID pParameter
                       );
```

Parameters

1) PTIMER_EXPFUNC pFunc: [IN] Timer expiry callback routine.
2) PVOID pParameter: [IN] Parameter passed to timer expiry callback

routine.

Return

If timer created successfully, return the handle of the timer, otherwise, return 0.

Note

7.1.3.2 start_timer

Description

Start the timer.

Function prototype

```
BOOL (*start_timer)(
                        HANDLE hTimer,
                        UINT32 nMillisecondes
                    );
```

Parameters

1) HANDLE hTimer: [IN] The handle of timer, which returned by **create_timer** interface.

2) UINT32 nMillisecondes: [IN] Timer expiry duration, in millisecond.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.3.3 stop_timer

Description

Stop the timer.

Function prototype

```
BOOL (*stop_timer)(
                        HANDLE hTimer
                    );
```

Parameters

1) HANDLE hTimer: [IN] The handle of timer, which returned by **create_timer** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.3.4 delete_timer

Description

Delete the timer.

Function prototype

```
BOOL (*delete_timer)(
                        HANDLE hTimer
                    );
```

Parameters

1) HANDLE hTimer: [IN] The handle of timer, which returned by **create_timer** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.3.5 available_timer

Description

Check the timer started or not.

Function prototype

```
BOOL (*available_timer)(
                        HANDLE hTimer
                       );
```

Parameters

1) HANDLE hTimer: [IN] The handle of timer, which returned by **create_timer** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.3.6 get_system_datetime

Description

Get date and time information form system.

Function prototype

```
BOOL (*get_system_datetime)(
                        T_AMOPENAT_SYSTEM_DATETIME* pDatetime
                       );
```

Parameters

1) T_AMOPENAT_SYSTEM_DATETIME* pDatetime: [OUT] Date and time from the system.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.3.7 set_system_datetime

Description

Set date and time information of the system.

Function prototype

```
BOOL (*set_system_datetime)(
                        T_AMOPENAT_SYSTEM_DATETIME* pDatetime
                       );
```

Parameters

1) T_AMOPENAT_SYSTEM_DATETIME* pDatetime: [IN] Date and time information.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.4 Critical section interfaces

7.1.4.1 enter_critical_section

Description

Enter critical section, and all interrupts will be ignored.

Function prototype

```
HANDLE (*enter_critical_section)(
                        VOID
                              );
```

Parameters

Void.

Return

Critical section handle.

Note

This interface can be called nested, but you must call exit_critical_section interface correspondingly.

7.1.4.2 exit_critical_section

Description

Exit critical section.

Function prototype

```
VOID (*exit_critical_section)(
```

```
                    HANDLE hSection
                        );
```

Parameters

1) HANDLE hSection: [IN] The handle of critical section, which returned by the correspondingly **enter_critical_section** interface.

Return

Void.

Note

7.1.5 Semaphore interfaces

7.1.5.1 create_semaphore

Description

Create new semamphore.

Function prototype

```
HANDLE (*create_semaphore)(
                        UINT32 nInitCount
                          );
```

Parameters

1) UINT32 nInitCount: [IN] The count of the semaphore.

Return

If semaphore created successfully, return the handle of the semaphore, otherwise, return 0.

Note

7.1.5.2 delete_semaphore

Description

Delete the semaphore.

Function prototype

```
BOOL (*delete_semaphore)(
                        HANDLE hSem
                        );
```

Parameters

1) HANDLE hSem: [IN] The handle of semaphore, which returned by **create_semaphore** interface.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.5.3 wait_semaphore

Description

Wait the semaphore.

Function prototype

```
BOOL (*wait_semaphore)(
                        HANDLE hSem,
                        UINT32 nTimeOut
                       );
```

Parameters

1) HANDLE hSem: [IN] The handle of semaphore, which returned by **create_semaphore** interface.
2) UINT32 nTimeOut: [IN] NOT support now.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.6 Memory interfaces

7.1.6.1 malloc

Description

Memory malloc interface.

Function prototype

```
PVOID (*malloc)(
                        UINT32 nSize
                );
```

Parameters

1) UINT32 nSize: [IN] The size you want.

Return

The address of memory if success, otherwise, return NULL.

Note

7.1.6.2 realloc

Description

Malloc a new memory area with data.

Function prototype

```
PVOID (*realloc)(
                        PVOID pMemory,
                        UINT32 nSize
                 );
```

Parameters

1) PVOID pMemory: [IN] The address of memory, which returned by **malloc** interface.

2) UINT32 nSize: [IN] New size of memory you want.

Return

The address of memory if success, otherwise, return NULL.

Note

7.1.6.3 free

Description

Free memory block.

Function prototype

```
VOID (*free)(                          PVOID pMemory
            );
```

Parameters

1) PVOID pMemory: [IN] The address of memory, which returned by **malloc** interface.

Return

Void.

Note

7.1.7 Miscellaneous interfaces

7.1.7.1 sleep

Description

Make a thread sleeping for a while.

Function prototype

```
BOOL (*sleep)(
                          UINT32 nMillisecondes
              );
```

Parameters

1) UINT32 nMillisecondes: [IN] Duration for thread sleepy, in millisecond.

Return

TRUE for success, otherwise return FASLE.

Note

7.1.7.2 get_system_tick

Description

Get current system tick count.

Function prototype

```
UINT32 (*get_system_tick)(
                        VOID
                         );
```

Parameters

Void.

Return

Tick count from the system powered up. 1 Tick = (1/16384)s.

Note

7.1.7.3 rand

Description

Get a rand integer.

Function prototype

```
UINT32 (*rand)(
                        VOID
                );
```

Parameters

Void.

Return

An integer is between 0 and 32767.

Note

7.1.7.4 srand

Description

Set rand integer seed.

Function prototype

```
VOID (*srand)(
                        UINT32 seed
              );
```

Parameters

1) UINT32 seed: [IN] Random integer seed.

Return

Void.

Note

7.1.7.5 shut_down

Description

System shutdown interface, which will be powered off directly without doing network deregister.

Function prototype

```
VOID (*shut_down)(
                        VOID
                );
```

Parameters

Void.

Return

Void.

Note

7.1.7.6 restart

Description

System restart interface.

Function prototype

```
VOID (*restart)(
                        VOID
                );
```

Parameters

Void.

Return

Void.

Note

7.2 File System module

In our system, there are two partitions:
1) Flash partition: use “/” as root directory.
2) TF(MicroSD) card partition: use “/TFLASH” as root directory.

7.2.1 Interfaces

7.2.1.1 open_file

Description

Open a file.

Function prototype

```
INT32 (*open_file)(
                        PCHAR pszFileName,
                        UINT32 iFlag,
                        UINT32 iAttr
                    );
```

Parameters

1) PCHAR pszFileName: [IN] File name include directory information.

2) UINT32 iFlag: [IN] Open flags, see **E_AMOPENAT_FILE_OPEN_FLAG** for more information.

3) UINT32 iAttr: [IN] NOT support now.

Return

File handle which be not a negative integer if success, otherwise be a negative integer.

If failed, please see **E_AMOPENAT_FS_ERR_CODE** for more information.

Note

7.2.1.2 close_file

Description

Function prototype

Parameters

Return

Note

7.2.1.3 read_file

Description

Function prototype

Parameters

Return

Note

7.2.1.4 write_file

Description

Function prototype

Parameters

Return

Note

7.2.1.5 seek_file

Description

Function prototype

Parameters

Return

Note

7.2.1.6 create_file

Description

Function prototype

Parameters

Return

Note

7.2.1.7 delete_file

Description

Function prototype

Parameters

Return

Note

7.2.1.8 change_dir

Description

Function prototype

Parameters

Return

Note

7.2.1.9 make_dir

Description

Function prototype

Parameters

Return

Note

7.2.1.10 remove_dir

Description

Function prototype

Parameters

Return

Note

7.2.1.11 remove_dir_rec

Description

Function prototype

Parameters

Return

Note

7.2.1.12 get_current_dir

Description

Function prototype

Parameters

Return

Note

7.2.1.13 find_first_file

Description

Function prototype

Parameters

Return

Note

7.2.1.14 find_next_file

Description

Function prototype

Parameters

Return

Note

7.2.1.15 find_close

Description

Function prototype

Parameters

Return

Note

7.2.1.16 init_tflash

Description

Function prototype

Parameters

Return

Note

7.3 Debug module

Provide log output and assert program interruption interface.

7.3.1 Interfaces

7.3.1.1 print

Description

Print log information to HOST UART.

Function prototype

```
VOID (*print)(
                        CHAR * fmt, ...
              );
```

Parameters

1) CHAR * fmt: [IN] Format string.
2) ...: [IN] Data to print.

Return

Void.

Note

s

7.3.1.2 assert

Description

Assert program interruption.

Function prototype

```
VOID (*assert)(
                        BOOL condition,
                        CHAR *func,
                        UINT32 line
               );
```

Parameters

1) BOOL condition: [IN] Condition.
2) CHAR *func: [IN] Function name which will display.
3) UINT32 line: [IN] Source file line number which assert occur.

Return

Void.

Note

7.4 VAT module

Virtual AT Command path interface, which sends AT command, receives the response for AT command and URC information.

Notes for **send_at_command** interface:

1. More than one AT command can be sent together
   E.g.: AT command can be ”ATE0;AT+CMGF=1\r\n”.
2. One piece of AT command can be separated and sent by many times.
   E.g.: Send ”AT+CMGF” first, then send ”=1\r\n”.

Specific use of interface can refer to notes in am_openat.h.

7.5 Driver module

Driver module includes the following sub-module interfaces:

7.5.1 GPIO interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.2 UART interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.3 Mono-LCD interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.4 Color-LCD interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.5 Keypad interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.6 Touchscreen interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.7 T-Card interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.7.1 init_tflash

Description

Function prototype

Parameters

Return

Note

7.5.8 Camera interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.9 PowerManage interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.10 ADC interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.11 Bluetooth interfaces

Use of specific interface refers to the notes in am_openat.h.

7.5.12 PSAM interfaces

7.5.13 SPI interfaces

7.5.14 Audio interfaces

8. FAQ

8.1 How to create auto restart timer?

Example:

```
/* Timer out routine */
static  VOID  cust_timerout_handler(T_AMOPENAT_TIMER_PARAMETER
*pCbParam)
{
    //post timer expire message to cust main task
```

```
        ...

        /* restart timer */
        IVTBL(start_timer)(pCbParam->hTimer, pCbParam->period);
    }

    /* main */
    HANDLE  hTimer  =  IVTBL(create_timer)(cust_timerout_handler,
NULL);
    IVTBL(start_timer)(hTimer, 1000);
    ...
```