AC109N-E SDK

User Guide

目录

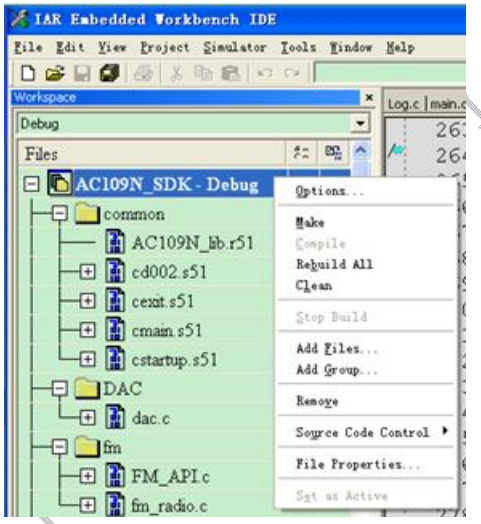
一、IAR 集成开发环境介绍

本工程（AC109N-E SDK）使用的是 IAR Embedded Workbench IDE（集成开发环境）7.20H 版本，用户需安装相应的版本进行开发。

（注：更高版本可能存在编译问题）

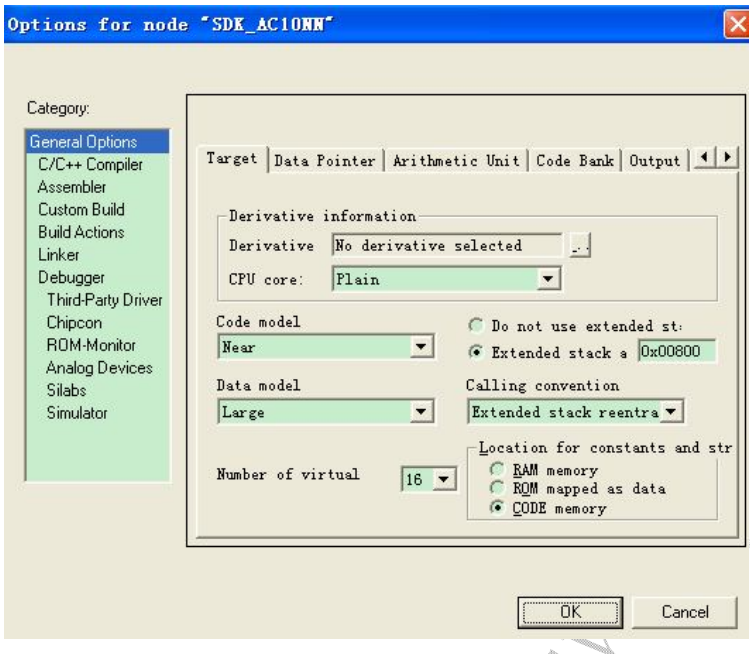
1.1 编译器设置

AC109N-E 开发包内 AC109N-E_SDK.eww 为对应的工程文件，打开后便可以进行工程设置，右键工程名->Options…，如图 1-1 所示：



（图 1-1）

点击 Options 后进入工程设置，里面包括 General Options（常规设置选项）、C/C++ Complier（C/C++ 编译器选项）、Assembler（汇编器选项）和 Linker（连接器选项）等用户常用的工程设置选项，如图 1-2：



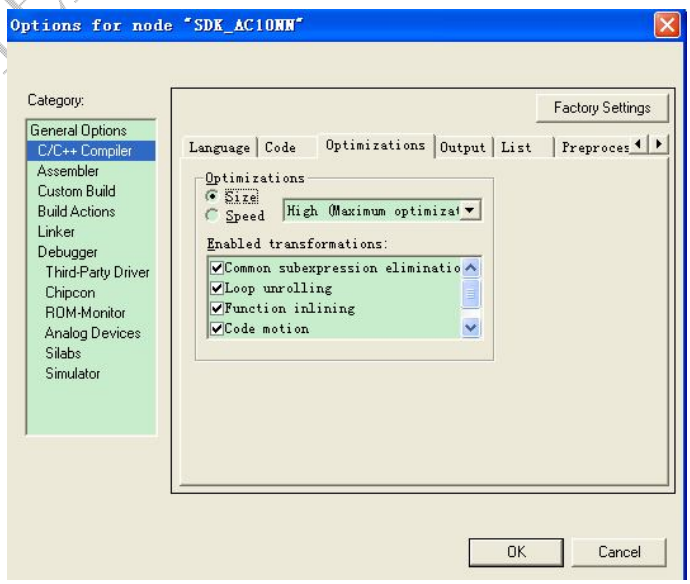
（图 1-2）

1.1.1 General Options

在 General Options（常规设置）的 Target 页内包括了几个常用的工程设置，堆栈的起始地址、函数调用的重入规则、变量、常量和字符串的默认存储类型

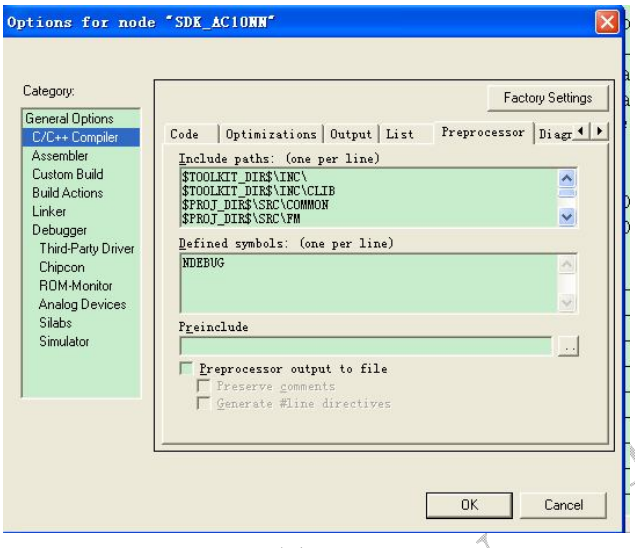
1.1.2 C/C++ Complier

在 C/C++ Complier（C/C++编译器选项）的 Optimizations 页里面，包含了代码优化设置，本工程默认使用空间最高级优化方式。



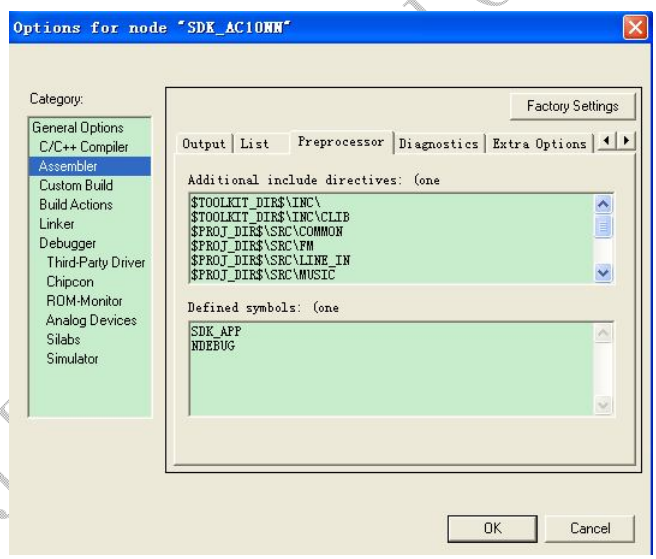
（图 1-3）

另外在 Preprocessor 页内包含了 C/C++工程所包含的文件路径设置，用户新增加的 C/C++文件夹路径需添加到 Include Path（包含路径）内，如图 1-4：



（图 1-4）

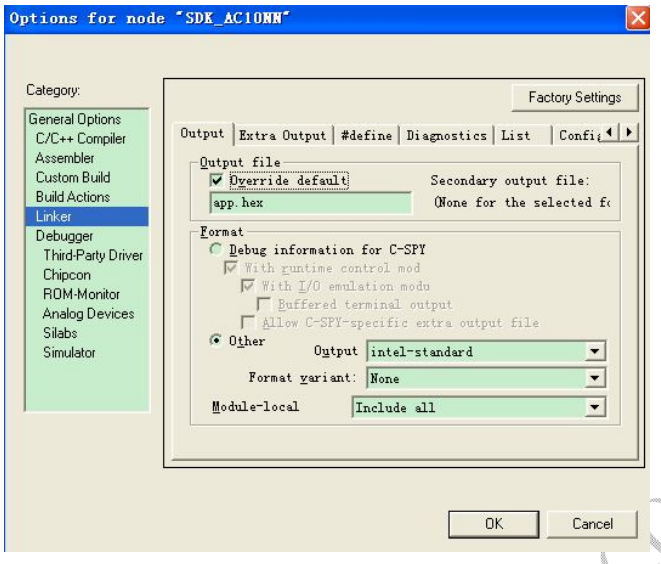
其中$PROJ_DIR$\为工程所在目录的路径。同样地，在 Assembler\选项内包含了汇编工程所包含的文件路径设置，同样需要添加文件夹路径到 Include Path（包含路径）内，如图 1-5



（图 1-5）

1.1.3 Linker

在 Linker 选项内包含了连接过程的设置，其中 Output 页包含输出文件格式设置，如图 1-6：



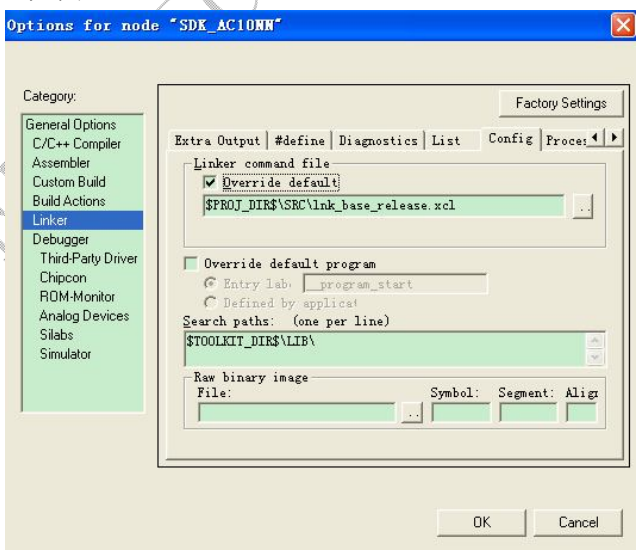
（图 1-6）

Override default 可以修改输出 hex 格式的文件名，Output 为 intel-standard 标准 51 Hex 格式。

注：详细介绍可参考《EW8051_UserGuide.pdf》。

1.2 资源分配

IAR 可以通过设置.xcl 文件配置工程的资源分配，首先需要在 Linker 选项的 Config 里添加用户自定义的.xcl 文件，如图 1-7：



（图 1-7）

勾选 Override default，lnk_base_release.xcl 为工程所用的资源配置文件，路径为工程目录下。

打开 lnk_base_release.xcl 文件，里面由两部分组成，分别为范围指定和资源分配，如图 1-8：

```
28//     DATA
29//
30-D_DATA_START=0x30
31-D_DATA_END=0x7F
```

（图 1-8）

-D 为宏定义_DATA_START/_DATA_END 的命令，分别定义了 Data/IData/PData/XData/Code/Near_Code 的范围；

在宏定义了范围后，实际控制资源分配的命令如图 1-9：

```
125
126 -Z(DATA)DOVERLAY=_DATA_START-_DATA_END
127 -Z(DATA)DATA_I,DATA_Z,DATA_N=_DATA_START-_DATA_END
```

（图 1-9）

-Z 为定义指定的段到指定的区域的命令，分别指定了 DATA_I,DATA_Z,DATA_N 的段到_DATA_START/_DATA_END 的区域内，即 0x30-0x7F；用户可以通过设置.xcl 控制资源的分配。

注：详细介绍可参考《xlink.pdf》

1.3 常用关键字介绍

- __root：函数不会因为未被调用而被优化删除；
- __no_init：不需要初始化的变量声明，本工程全局变量统一使用此声明方式；
- void fun(void) AT (*segment name*)：指定函数到特定的段 *segment name* 为段名；
- #pragma location = *addr*
  __no_init unsigned char *var* ;绝对定位变量 *var* 到 *addr* 地址；
- #pragma data_alignment = *n*
  __no_init u16 *array*[5];数组变量 *array* 在分配地址的时候 *n* 对齐；

注：详细介绍可参考《8051 IAR Embedded Workbench Help》

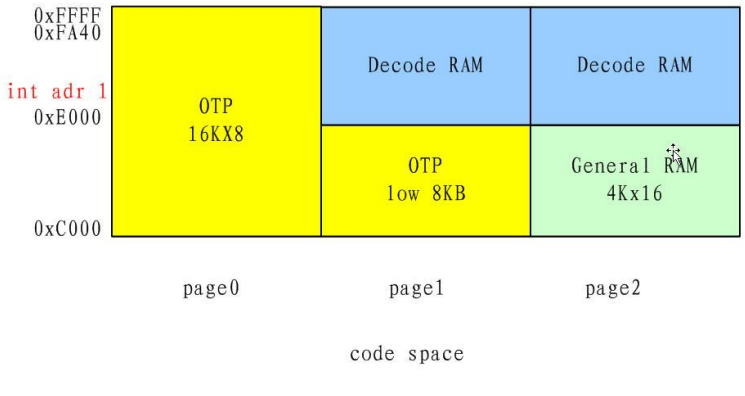
二、系统结构介绍

本方案为针对音频类电子消费类产品的软件开发包，系统集成了 USB OTG/SD 驱动、多款 FM 模块驱动、显示屏驱动（LED/LCD 点阵屏/LCD 段码屏）、IRTC 时钟驱动、红外遥控、内置 IRTC RAM；详细芯片资源介绍请参考《AC109N-E_SDK_v110.chm》。

同时具有特色包括：音乐断点记忆功能、语音提示功能、设备记忆、频谱显示功能、支持双 SD 卡。

2.1 CODE 空间介绍

本系统有 64KB 的程序空间，分别由 48KB Maskrom 和 16KB OTP 组成，如图 2-1：

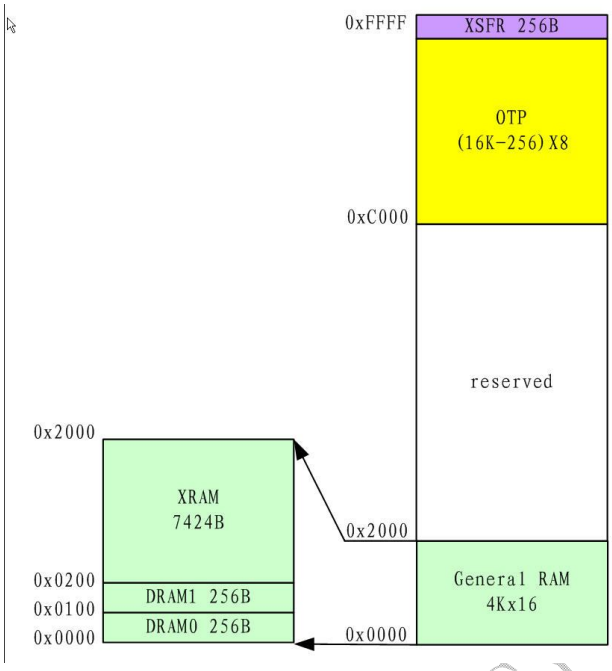


（如图 2-1）

注意在 Page1 的映射关系中 RAM 空间映射到原 OTP 0xE000-10xFA40；在 Page 2 的映射关系中 RAM 空间映射到原 OTP 0xC000-0xFA40。

2.2 RAM 空间介绍

本系统有 8KB 的数据空间，其中部分数据空间被 Maskrom 工程占用，如图 2-2：



（图 2-2）

2.3 OTP 系统资源分配

类型	起始地址	结束地址
Data	0x30	0x7F
IData	0x80	0xFF
Extend Stack	0x800	0x97F
XData	0x980	0x144F
Decode XData	0x1850	0x1FFF

注意：OTP 工程能够使用的 XData 空间仅有 0x980~0x144F

2.4 OTP 开发注意事项

2.4.1 系统配置文件

Config.h 文件为系统配置文件，通过开关里面的宏定义可以方便用户定制自定义方案，该配置文件组织形式为以下几部分：

- 时钟配置，可配置输入晶振类型、系统时钟运行频率，如图 2-3：

```
47
48 /*---------Clock Configuration*/
49 #define OSC_32768           32768L
50 #define OSC_12M             12000000L
51
52 #define OSC_CLK             OSC_32768
53
54 //SYSTEM CLOCK
55 #define CLK_256K            256000L
56 #define CLK_512K            512000L
57 #define CLK_12M             12000000L
58 #define CLK_24M             24000000L
59 #define CLK_48M             48000000L
60
61 #define SYSTEM_CLK          CLK_24M
62 //#define CLK_USE_HTC
63 //#define CLK_USE_32K_WITH_HTC
64 #define CLK_USE_32K_NO_HTC
65 //#define CLK_USE_12M_NO_HTC
66 //#define CLK_USE_12M_WITH_HTC
67
68 //#define SHARE_32K_TO_FM
69
```

（图 2-3）

- 显示配置，可选择不同的显示屏驱动，如图 2-4：

```
94
95 //-------------------------------------------------------------------
96 /*---------UI Configuration*/
97 //#define LCD_96X32_SERIAL
98 //#define LCD_128X32_SERIAL
99 //#define LCD_128X64_SERIAL
100 //#define LCD_128X64_PARALLEL
101 //#define LCD_SEG_4X8
102 //#define LCD_SEG_3X9
103 //#define LCD_SEG_5X9
104 #define LED_5X7
```

（图 2-4）

- FM 模块配置，可选择对应型号的 FM 驱动，如图 2-5：

```
89 /*---------FM Configuration*/
90 #define RDA5807
91 //#define BK1080
92 //#define KT0830EG
93 //#define QN8035
94 //#define AR1019
95
```

（图 2-5）

- 系统特色功能配置，如图 2-6：

```
148
149 /*---------System Charateristic Configuration v100---------------*/
150 #define     IR_REMOTE_EN                    //是否需要红外遥控
151 ///<音乐播放功能选择
152 #define     LAST_MEM_FILE_PLAY_EN           //是否允许记忆文件序号播放功
153 #ifdef      LAST_MEM_FILE_PLAY_EN
154 #define     BREAK_POINT_PLAY_EN             //是否允许断点播放功能
155 #endif
156
157 //MP3频谱存放于xdata 0x2cd4~0x2cdd中，共10段，每段16bit
158 //#define    MP3_SPECTRUM                    //MP3频谱显示
159 //#define    FF_FR_MUSIC_EN                  //在快进快退时，是否需要听
160
161 //#define    KEY_VOICE_EN                    //按键音使能，如果使用按键
162 #ifndef     KEY_VOICE_EN                    //没有按键音时，方可以使用
163 #define     MUSIC_ENERGY_DETECT             //用于检测解码后的数据大小
164 #define     MAX_WAVEFORM        1           //检测解码后数据的限制范围
165 //门槛值必须为非0值(1~255)，1:完全静音，2: <-90.3dB，3: < -84.29dB，
166 #endif
167
168 //#define      GET_MUSIC_TOTAL_TIME           //是否获取音乐文件总时间
169 //#define    RANDOM_PLAY_EN                  //是否支持随机播放功能
170 //#define    FOLDER_PLAY_EN                  //是否支持文件夹切换和播放
171
172 #define     USB_DISK_EN                     //是否可以读U盘
173 #define     SDMMC_IDLE_EN                   //可以使SDMMC卡进入省电模式
174 #define     UDISK_IDLE_EN                   //有些(较少)U盘在此模式下，
175
176
177 //#define      USE_EEPROM_MEMORY             //使用EEPROM 作为存储器记忆
178 #ifndef     USE_EEPROM_MEMORY                //使用内部RTC RAM作记忆
179 #define     USE_IRTC_MEMORY
180 #endif
181
182 #ifdef      USE_EEPROM_MEMORY
183 #define     CHECK_EEPROM_ON_POWER_ON        //是否在上电时，校验eeprom
184 #endif
```

（图 2-6）

- 工作模式配置，如图 2-7：

```
 96 #if defined RDA5807 || defined BK1080 || defined KT0830EG ||
 97 #define FM_ENABLE              //FM 模式开关
 98 #endif
 99
100 /*---------Work Mode Configuration*/
101 #define USB_DEVICE_EN         //Enable USB SLAVE MODE
102
186
187 /*---------System Charateristic Configuration v101---------------*/
188 #define RTC_EN                  //<RTC 模式使能控制位
189 #define RTC_ALARM_EN            //<RTC 闹钟使能控制位
190 #if (!defined RTC_EN) && (defined RTC_ALARM_EN)
191 #error("RTC selectd err!")
192 #endif
193
194
```

（图 2-7）

2.4.2 中断服务程序

本方案的中断服务程序没有使用 IAR 编译器定义的响应方式，用户新增加中断服务程序需要按照以下步骤添加：

- Cd002.s51 文件中将相应的中断入口开放，如 Timer0 中断入口为 0x03+0xE000，将 int_config TIMER0_INT 打开；
- 将相应的中断服务程序函数指针(pIsrfun)赋值给 int_enter_pro[*Vector*] = pIsrfun，*Vector* 为相应中断号；

按照上述操作便能完成中断服务程序的添加。

2.4.3 消息处理机制

在本方案中为了实现实时操作的目的，采用了消息处理机制来处理系统消息和用户操作消息这两大类消息，其中使用的消息池同时具有**先进先出**和**后进先出**的属性，分别定义为高优先级消息和低优先级消息，其实体为一个 32 Byte 大小的数组，一个消息占 1 Byte，消息处理机制实现接口如下：

- u8 app_get_msg(void)，消息统一获取的接口，包括系统消息、用户操作消息（按键消息和红外遥控消息）；
- void put_msg_fifo(u8 msg)，低优先级消息发送接口，该消息属性为先进先出；
- void put_msg_lifo(u8 msg)，高优先级消息发送接口，该消息属性为后进先出；
- void flush_all_msg(void)，消息清除接口，清空消息池；

详细的消息定义见 msg.h 文件，里面详细列举出系统所用的消息。

系统的自身的运作依靠的是系统消息触发，例如音乐播放流程，系统消息的运作过程如下：

1. 设备驱动触发“设备插入”系统消息 *MSG_SDMMCA_IN/MSG_USB_DISK_IN*；
2. 响应“设备插入”消息后，系统触发“新设备插入”消息 put_msg_lifo(MSG_MUSIC_NEW_DEVICE_IN);
3. 响应“新设备插入”消息后，系统执行模式跳转操作；从非音乐模式切换到音乐模式；
4. 进入音乐模式后，系统触发“查找设备”消息，put_msg_lifo(MSG_MUSIC_SELECT_NEW_DEVICE);
5. 在查找设备并初始化设备成功的情况下，继续响应“查找文件”消息 MSG_MUSIC_SELECT_NEW_FILE；
6. 查找文件有效后，响应“播放文件”消息 MSG_MUSIC_PLAY_NEW_FILE，解码开始。

注：流程过程中可以进行容错处理，在某一个消息响应中，可以通过触发不同的消息让系统执行不同的行为，此为消息处理机制的特点。

2.4.4 事件与消息转换

本系统除了有消息池外还有 32Bit 的事件容器，事件的属性是不会被覆盖，在未转换为消息之前不会因为溢出而导致丢失事件，同时事件的容器属性是低位事件优先被处理，高位的后处理，同时**事件的优先级要高于消息**，对于必须响应的操作可以使用事件。事件的实现接口如下：

- void put_event(u8 event)，发送指定事件接口；
- bool check_event(u8 event)，检查事件接口，从低位开始检查事件是否存在于事件容器；
- void clear_one_event(u8 event)，清除指定事件接口；
- void clear_all_event(void)，清空事件容器接口；

在使用事件时用户只需要完成指定事件的发送操作，事件与消息的转换已经提供了实现方法，如设备插入操作使用的是事件触发，添加用户定义事件的步骤如下：

1. 在 msg.h 添加自定义事件和转换后的消息，其中 13 个已被定义；
2. 在 key.c 文件 event_msg_table[]数组添加对应消息；

按照以上步骤便完成事件的添加，然后用户只需要调用发送指定事件接口 void put_event(u8 event)就可以完成事件的触发。

三、调试模式介绍

AC109N-E 工程分别支持两种调试模式，6 线调试模式和 2 线调试模式，分别对应不同的工程选项，用户使用调试模式可以在不烧写 OTP 的情况下进行开发，开发后的程序通过 IDE 与实际芯片进行通讯，运行效果能基本接近实际烧写芯片的运行效果。

3.1 6 线调试模式介绍

若使用 6 线调试模式进行开发，只能支持开发板与 IDE 的组合使用的调试方式，编译模式作如下选择，如图 3-1：

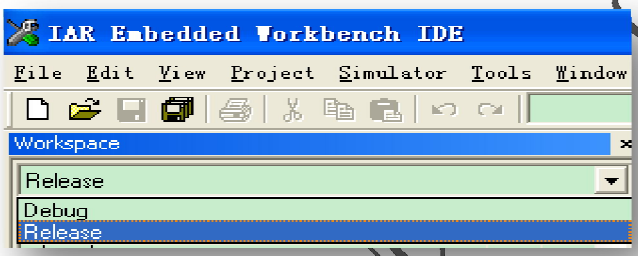


图 3-1

在 workspace 选择 Release 进行编译，工具链会自动下载程序到 IDE，芯片进入调试模式，便完成了 6 线工程的下载操作；另外在此模式下编译会生成最终烧写代码，烧写文件 myoutotp.bin.fw 生成在工程目录下的 link_hex6L 目录下，如图 3-2：

图 3-2

3.2 2 线调试介绍

3.2.1 编译器设置

不同于 6 线调试模式，2 线模式分别占用了 P00&P01/P24&P25 两组 I/O 口之一，***在进行该调试模式开发时需要注意不能使用被占用的 I/O，编译后不会生成烧写代码，必须切换会 Release 模式重新编译才能生成烧写代码，***另外 2 线模式是基于 Code Bank 的原理运行于芯片的指定的 RAM 区域，因此工程需要进行 CodeBanking 处理，首先编译器选项需选择为 Debug 编译模式，如图 3-3

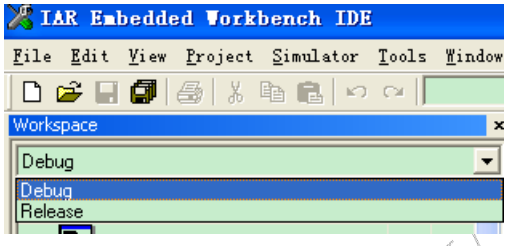


图 3-3

3.2.2 .xcl 文件介绍

在选择了 Debug 编译模式后，代码就需要进行分 Bank 处理，此时需要修改 lnk_base_debug.xcl 文件，该文件主要提供以下修改，如 BankCode 的大小，如图 3-4：

```
55//
56//
57//    BANKED CODE
58//
59-D_BANKED_CODE_START=0xF240
60-D_BANKED_CODE_END=0xF9BF
61-D_BANKED1_CODE_END=0xFA3F        // Last address for near code. tot
62//
63//    NEAR CODE
64//
65-D_NEAR_CODE_START=0x00DA40
66-D_NEAR_CODE_END=0x00F23F        // Last address for near code.
67//
68//
```

图 3-4

本工程定义了 6K 的公共区（NEAR CODE）和 1.92K 的 Bank 区（BANKED CODE），代码运行于芯片的 RAM 区域，范围见图 3-4；

在分配好代码空间大小后，需要对代码段进行 Bank 分配，同样可以在.xcl 文件里完成设置，如图 3-5 定义了 Common Code 的代码段

```
145
146/*----Common Code------*/
147-Z(CODE)CODE_C=_NEAR_CODE_START-_NEAR_CODE_END
148-Z(CODE)NEAR_CODE=_NEAR_CODE_START-_NEAR_CODE_END
149-Z(CODE)RCODE,MY_UART=_NEAR_CODE_START-_NEAR_CODE_END
150-Z(CODE)DIFUNCT=_NEAR_CODE_START-_NEAR_CODE_END
151
152/*-------2012/09/13 Bingquan added------------*/
153-Z(CODE)COMMON_CODE,TABLE_CODE=_NEAR_CODE_START-_NEAR_CODE_END
154/*--Key*/
155-Z(CODE)KEY_CODE=_NEAR_CODE_START-_NEAR_CODE_END
156
```

图 3-5

图 3-6 定义了 Bank Code 的代码段

```
170 //CSTART必须在bank0(常驻Bank)
171 /*----Bank0 Code------*/
172 -P(CODE)CSTART,BANK0_CODE,BANK0_TABLE_CODE=_BANKED_CODE_START-_BANKED_CODE_END
173 -P(CODE)DAC_CODE,DAC_TABLE_CODE=_BANKED_CODE_START-_BANKED_CODE_END
174
175 /*----Bank1 Code------*/
176 -P(CODE)MUSIC_PLAY,BANK1_CODE,BANK1_TABLE_CODE=(_BANKED_CODE_START+0x10000)-(_BANKED_CODE_EN
177
178 /*----Bank2 Code------*/
179 -P(CODE)GET_MUSIC_FILE,GET_DEVICE,FS_CODE,BANK2_CODE,BANK2_TABLE_CODE=(_BANKED_CODE_START+0x
180
181 /*----Bank3 Code------*/
182 -P(CODE)USB_DEVICE_CODE,USB_HOST_CODE,BANKED_CODE,BANK3_CODE,BANK3_TABLE_CODE=(_BANKED_CODE_
183
184 /*----Bank4 Code------*/
185 -P(CODE)RTC_CODE,BANK4_CODE,BANK4_TABLE_CODE=(_BANKED_CODE_START+0x40000)-(_BANKED_CODE_END+
186
187 /*----Bank5 Code------*/
188 -P(CODE)LINE_IN_CODE,BANK5_CODE,BANK5_TABLE_CODE=(_BANKED_CODE_START+0x50000)-(_BANKED_CODE_
189 -P(CODE)LED_5X7_CODE,LED_5X7_TABLE_CODE=(_BANKED_CODE_START+0x50000)-(_BANKED_CODE_END+0x500
190 -P(CODE)LCD_SEG_CODE,LCD_SEG_TABLE_CODE=(_BANKED_CODE_START+0x50000)-(_BANKED_CODE_END+0x500
191 -P(CODE)LCD_CODE=(_BANKED_CODE_START+0x50000)-(_BANKED_CODE_END+0x50000)
192 -P(CODE)UI_COMMON_CODE,UI_TABLE_CODE=(_BANKED_CODE_START+0x50000)-(_BANKED_CODE_END+0x50000)
193
194
195 /*----Bank6 Code------*/
196 -P(CODE)BK1080_CODE,BK1080_TABLE_CODE=(_BANKED_CODE_START+0x60000)-(_BANKED_CODE_END+0x60000
197 -P(CODE)KT0830_CODE,KT0830_TABLE_CODE=(_BANKED_CODE_START+0x60000)-(_BANKED_CODE_END+0x60000
198 -P(CODE)QN0835_CODE,QN0835_TABLE_CODE=(_BANKED_CODE_START+0x60000)-(_BANKED_CODE_END+0x60000
199 -P(CODE)RDA5807_CODE,RDA5807_TABLE_CODE=(_BANKED_CODE_START+0x60000)-(_BANKED_CODE_END+0x600
200 -P(CODE)AR1019_CODE,AR1019_TABLE_CODE,BANK6_CODE,BANK6_TABLE_CODE=(_BANKED_CODE_START+0x6000
201
202 /*----Bank7 Code------*/
203 -P(CODE)IIC_CODE,IIC_TABLE_CODE,FM_CODE,FM_TABLE_CODE,BANK7_CODE,BANK7_TABLE_CODE=(_BANKED_C
204
```

图 3-6

在进行 CodeBanking 的过程中需要尽量遵守以下原则：

1. 被中断服务函数调用的函数需要放置于 Common Code；
2. 同一模式下的函数或相互调用频繁的函数应该定义在相同 Bank Code；
3. Bank 函数使用的 const 应与 Bank 函数放置于同一个 Bank；

Code Bank 的工程在运行时应尽量以少 Bank 切换操作为优先考虑因素。

工程默认分 8 个 Bank，用户需要增加 Bank Code 可以修改[_BANKED_CODE_START-_BANKED1_CODE_END]*8（默认 Bank 数）

```
205 -M(CODE)_NEAR_CODE_START-_NEAR_CODE_END=0-17FF
206 -M(CODE)[_BANKED_CODE_START-_BANKED1_CODE_END]*8+10000=1800
207
```

3.2.3 硬件连接

使用 2 线调试模式，*调试方式不仅限于开发板与 IDE 的组合，还支持在实际样机上调试*，硬件连接方式如下：备注：主控 LDO5V 需供电

IDE 接口	AC109N-E IO 口
GND	GND
IDE_TME	VCOM
IDE_CLK	P00/P24
IDE_TDI	P01/P25

版本信息

日期	版本	备注	作者
2013-6-13	AC109N-E SDK v110		Bingquan Cai