User Manual

M22A-F/N/U20

MiniARM 嵌入式工控模块

UM03050129 V1.00 Date: 2008/03/05

产品用户手册

类别	内容
关键词	MiniARM、嵌入式工控模块
摘　要	讲解核心板上资源以及内部固件的使用方法

广州致远电子有限公司

ZHIYUAN ELECTRONICS

修订历史

版本	日期	原因
V1.00	2008/03/05	创建文档

销售与服务网络（一）

广州周立功单片机发展有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4 邮编：510630
电话：(020)38730916 38730917 38730972 38730976 38730977
传真：(020)38730925
网址：www.zlgmcu.com

广州专卖店

地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634 87569917
传真：(020)87578842

南京周立功

地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221 83613271 83603500
传真：(025)83613271

北京周立功

地址：北京市海淀区知春路 113 号银网中心 A 座 1207-1208 室（中发电子市场斜对面）
电话：(010)62536178 62536179 82628073
传真：(010)82614433

重庆周立功

地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室
电话：(023)68796438 68796439
传真：(023)68796439

杭州周立功

地址：杭州市登云路 428 号浙江时代电子市场 205 号
电话：(0571)88009205 88009932 88009933
传真：(0571)88009204

成都周立功

地址：成都市一环路南二段 1 号数码同人港 401 室（磨子桥立交西北角）
电话：(028)85439836 85437446
传真：(028)85437896

深圳周立功

地址：深圳市深南中路 2070 号电子科技大厦 A 座 24 楼 2403 室
电话：(0755)83781788（5 线）
传真：(0755)83793285

武汉周立功

地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497 87168297 87168397
传真：(027)87163755

上海周立功

地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452 53083453 53083496
传真：(021)53083491

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296 83063000 87881295
传真：(029)87880865

销售与服务网络（二）

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼
邮编：510660
传真：(020)38601859
网址：www.embedtools.com （嵌入式系统事业部）
www.embedcontrol.com （工控网络事业部）
www.ecardsys.com （楼宇自动化事业部）

技术支持：

CAN-bus：

电话：(020)22644381 22644382 22644253
邮箱：can.support@embedcontrol.com

iCAN 及模块：

电话：(020)28872344 22644373
邮箱：ican@embedcontrol.com

MiniARM：

电话：(020)28872684
邮箱：miniarm.support@embedtools.com

以太网及无线：

电话：(02 0)22644385 22644386
邮箱：wireless@embedcontrol.com
ethernet.support@embedcontrol.com

编程器：

电话：(020)38681856 28872449
邮箱：programmer@embedtools.com

分析仪器：

电话：(020)22644375 28872624 28872345
邮箱：tools@embedtools.com

ARM 嵌入式系统：

电话：(020)28872347 28872377 22644383 22644384
邮箱：arm.support@zlgmcu.com

楼宇自动化：

电话：(020)22644376 22644389
邮箱：mjs.support@ecardsys.com
mifare.support@zlgmcu.com

销售：

电话：(020)22644249 22644399 28872524 28872342
28872349 28872569 28872573

维修：

电话：(020)22644245

目 录

1. 功能简介

M22A 系列工控模块集成 LPC2220/LPC2290 微控制器最小系统、它最大、最突出的特点是集成 NAND Flash 电子盘、10M 以太网通信控制器以及 USB 主机控制器，其内存容量为 2Mbytes/8Mbytes。产品提供保护型总线设计，使该系列模块在 EMC 性能及稳定性方面具有良好的表现。

MiniARM 系列产品内置有商业化 TCP/IP 协议栈固件、USB 控制器驱动固件和文件管理系统固件，用户只需调用系统提供的 API 函数即可实现相应的功能。产品提供在线升级功能接口，可以使产品更快投入市场，而且现场升级简单可靠，明显增强产品的市场竞争力。

MiniARM 嵌入式工控模块各产品型号及对应功能列表如表 1.1 所示。

表 1.1 MiniARM 嵌入式工控模块型号及功能表

产品型号	程序存储器容量	内存容量	电子盘	以太网	CAN-bus	U 盘	CF 卡	Modbus	温度范围
M2020-FNU20	2MB	2MB	256MB	10M	--	√	√	√	I/C
M2080-FNU20	2MB	8MB	256MB	10M	--	√	√	√	I/C
M9020-FNU20	2MB	2MB	256MB	10M	2 路	√	√	√	I/C
M9080-FNU20	2MB	8MB	256MB	10M	2 路	√	√	√	I/C
M2020-FN20	2MB	2MB	256MB	10M	--	--	√	√	I/C
M2080-FN20	2MB	8MB	256MB	10M	--	--	√	√	I/C
M9020-FN20	2MB	2MB	256MB	10M	2 路	--	√	√	I/C
M9080-FN20	2MB	8MB	256MB	10M	2 路	--	√	√	I/C
M2020-FU20	2MB	2MB	256MB	--	--	√	√	√	I/C
M2080-FU20	2MB	8MB	256MB	--	--	√	√	√	I/C
M9020-FU20	2MB	2MB	256MB	--	2 路	√	√	√	I/C
M9080-FU20	2MB	8MB	256MB	--	2 路	√	√	√	I/C
M2020-NU20	2MB	2MB	--	10M	--	√	√	√	I/C
M2080-NU20	2MB	8MB	--	10M	--	√	√	√	I/C
M9020-NU20	2MB	2MB	--	10M	2 路	√	√	√	I/C
M9080-NU20	2MB	8MB	--	10M	2 路	√	√	√	I/C
M2020-F20	2MB	2MB	256MB	--	--	--	√	√	I/C
M2080-F20	2MB	8MB	256MB	--	--	--	√	√	I/C
M9020-F20	2MB	2MB	256MB	--	2 路	--	√	√	I/C
M9080-F20	2MB	8MB	256MB	--	2 路	--	√	√	I/C
M2020-N20	2MB	2MB	--	10M	--	--	√	√	I/C
M2080-N20	2MB	8MB	--	10M	--	--	√	√	I/C
M9020-N20	2MB	2MB	--	10M	2 路	--	√	√	I/C
M9080-N20	2MB	8MB	--	10M	2 路	--	√	√	I/C
M2020-U20	2MB	2MB	--	--	--	√	√	√	I/C
M2080-U20	2MB	8MB	--	--	--	√	√	√	I/C
M9020-U20	2MB	2MB	--	--	2 路	√	√	√	I/C
M9080-U20	2MB	8MB	--	--	2 路	√	√	√	I/C

1.1 硬件资源介绍

MiniARM 嵌入式工控模块集成大量硬件资源：

1. 2MB NOR Flash；
2. 2MB/8MB PSRAM（板上集成）；
3. 10M 以太网接口；
4. 2 路 USB-Host 控制器；
5. 集成电子盘（可选配置 16MB - 1GB）；
6. 2 路 CAN 控制器；
7. E^2 PROM 以及 RTC 等。

用户需要为模块提供系统电源和 RTC 电源，系统电源：3.3V±5%、5.0V±5%，RTC 电源：1.8V～5.5V。

MiniARM 嵌入式工控模块上总线器件的地址分配如表 1.2 所示，其它的地址空间属于保留空间，用户不可操作。

表 1.2 模块总线地址分配

总线器件	地址分配
NOR Flash	0x8000 0000 ～ 0x801F FFFF（2MB）
板上 PSRAM	0x8100 0000 ～ 0x811F FFFF（2MB）或 0x8100 0000 ～ 0x817F FFFF（8MB）
以太网	0x8300 0000 ～ 0x831F FFFF
USB 控制器	0x8320 0000 ～ 0x833F FFFF
NAND Flash 电子盘	0x8340 0000 ～ 0x835F FFFF
I^2C	0x80， 0xA0， 0xA2

由于 MiniARM 嵌入式工控模块均使用开放的总线设计，所以可使用外部总线扩展器件（如扩展 I/O、LCD 液晶屏、高速 ADC 等），可供用户使用的外部总线地址空间为：0x8200 0000 ～ 0x82FF FFFF。

1.2 外部接口

MiniARM 嵌入式工控模块上的 CPU 为 LPC2220/LPC2290，同时板上集成有以太网控制器、USB 主机控制器以及 NAND Flash 电子盘等。核心板提供的外设接口有：UART0、UART1、SPI、I2C、USB-Host1、USB-Host2、CAN1、CAN2、以太网接口以及总线接口等。表 1.3-表 1.5 列出了 MiniARM 嵌入式工控模块默认引脚功能对照表，用户可根据实际情况配置相应引脚功能。

注：I^2C 引脚功能不能更改。

表 1.3　MiniARM 嵌入式工控模块引脚 P0 口功能规划对照表

LPC2220/LPC2290 名称	配置功能	MiniARM 名称	用户用途
P0.0/TXD0/PWM1	TXD0	**TXD0**	串口0
P0.1/RXD0/PWM3/EINT0	RXD0	**RXD0**	
P0.2/SCL/CAP0.0	SCL	**SCL0**	I2C总线
P0.3/SDA/MAT0.0/EINT1	SDA	**SDA0**	
P0.4/SCK0/CAP0.1	SCK0	**SCK0**	SPI0 总线
P0.5/MISO0/MAT0.1	MISO0	**MISO0**	
P0.6/MOSI0/CAP0.2	MOSI0	**MOSI0**	
P0.7/SSEL0/PWM2/EINT2	SSEL0	**SSEL0**	
P0.8/TXD1/PWM4	TXD1	**TXD1**	串口1
P0.9/RXD1/PWM6/EINT3	RXD1	**RXD1**	
P0.10/RTS1/CAP1.0/RD5	RTS1	**RTS1**	
P0.11/CTS1/CAP1.1/TD5	CTS1	**CTS1**	
P0.12/DSR1/MAT1.0/RD4	P0.12	**P0_12**	用户GPIO
P0.13/DTR1/MAT1.1/TD4	P0.13	**P0_13**	用户GPIO
P0.14/DCD1/EINT1	EINT1	**USRINT0**	用户中断0
P0.15/RI1/EINT2	EINT2	**N.C.**	无
P0.16/EINT0/MAT0.2/CAP0.2	EINT0	**N.C.**	无
P0.17/CAP1.2/SCK1/MAT1.2	CAP1.2	**CAP0**	脉冲捕获
P0.18/CAP1.3/MISO1/MAT1.3	MAT1.3	**MAT0**	匹配输出
P0.19/MAT1.2/MOSI1/CAP1.2	P0.19	**P0_19**	用户GPIO
P0.20/MAT1.3/SSEL1/EINT3	EINT3	**USRINT1**	用户中断1
P0.21/PWM5/RD3/CAP1.3	PWM5	**PWM0**	PWM输出
P0.22/TD3/CAP0.0/MAT0.0	P0.22	**P0_22**	用户GPIO
P0.23/RD2	RD2	**RD2**	CAN2（仅LPC2290）
P0.24/TD2	TD2	**TD2**	
P0.25/RD1	RD1	RD1	CAN1（仅LPC2290）
TD1	TD1	**TD1**	
P0.27/AIN0/CAP0.1/MAT0.1	AIN0	**AIN0**	模拟输入
P0.28/AIN1/CAP0.2/MAT0.2	AIN1	**AIN1**	

P0.29/AIN2/CAP0.3/MAT0.3	AIN2	**AIN2**	
P0.30/AIN3/EINT3/CAP0.0	AIN3	**AIN3**	

注：CAN 功能只有 LPC2290 支持，LPC2220 不支持。

表 1.4　MiniARM 嵌入式工控模块引脚 P1 口功能规划对照表

LPC2220/LPC2290 名称	配置功能	MiniARM 名称	用户用途
P1.16	P1.16	**N.C.**	无
P1.17	P1.17	**N.C.**	无
P1.18	P1.18	**N.C.**	无
P1.19	P1.19	**N.C.**	无
P1.20	P1.20	**N.C.**	无
P1.21	P1.21	**N.C.**	无
P1.22	P1.22	**N.C.**	无
P1.23	P1.23	**N.C.**	无
P1.24	P1.24	**N.C.**	无
P1.25	P1.25	**N.C.**	无

表 1.5　MiniARM 嵌入式工控模块引脚 P2、P3 口功能规划对照表

LPC2220/LPC2290 名称	配置功能	MiniARM 名称	用户用途
P2.16	P2.16	**P2_16**	用户GPIO
P2.17	P2.17	**P2_17**	用户GPIO
P2.18	P2.18	**P2_18**	用户GPIO
P2.19	P2.19	**P2_19**	用户GPIO
P2.20	P2.20	**P2_20**	用户GPIO
P2.21	P2.21	**P2_21**	用户GPIO
P2.22	P2.22	**P2_22**	用户GPIO
P2.23	P2.23	**P2_23**	用户GPIO
P2.24	P2.24	**P2_24**	用户GPIO
P2.25	P2.25	**P2_25**	用户GPIO
P2.26/BOOT0	BOOT0	**N.C.**	N.C.
P2.27/BOOT1	BOOT1	**N.C.**	N.C.
P2.28/RD6	P2.28	**P2_28**	用户GPIO

P2.29/TD6	P2.29	**P2_29**	用户GPIO
P2.30/AIN4	P2.30	**P2_30**	用户GPIO
P2.31/AIN5	P2.31	**P2_31**	用户GPIO
LPC2220/LPC2290 名称	**配置功能**	**MiniARM 名称**	**用户用途**
P3.28/nBLS3/AIN7	P3.28	**P3_28**	用户GPIO
P3.29/nBLS2/AIN6	P3.29	**P3_29**	用户GPIO

为了方便用户调试，表 1.6 用列表方式标示引脚的位置和默认功能；若用户自行配置各管脚功能，则请参考表 1.7。

表 1.6 MiniARM 嵌入式工控模块引脚默认功能表（MiniARM 标准）

TXD0	RXD0	SCL0	**3**	**1**	A0_B	A1_B	A2_B
SDA0	SCK0	MISO0	**6**	**4**	A3_B	A4_B	A5_B
MOSI0	SSEL0	TXD1	**9**	**7**	A6_B	A7_B	A8_B
RXD1	RTS1	CTS1	**12**	**10**	A9_B	A10_B	A11_B
P0_12	P0_13	USRINT0	**15**	**13**	A12_B	A13_B	A14_B
N.C.	N.C.	CAP0	**18**	**16**	A15_B	A16_B	A17_B
MAT0	P0_19	USRINT1	**21**	**19**	A18_B	A19_B	A20_B
PWM0	P0_22	RD2	**24**	**22**	A21_B	A22_B	A23
TD2	RD1	TD1	**27**	**25**	D0_B	D1_B	D2_B
AIN0	AIN1	AIN2	**30**	**28**	D3_B	D4_B	D5_B
AIN3	AGND	AGND	**33**	**31**	D6_B	D7_B	D8_B
DGND	P2_16	P2_17	**36**	**34**	D9_B	D10_B	D11_B
P2_18	P2_19	P2_20	**39**	**37**	D12_B	D13_B	D14_B
P2_21	P2_22	P2_23	**42**	**40**	D15_B	nBLS0_B	nBLS1_B
P2_24	P2_25	N.C.	**45**	**43**	nOE_B	nWE_B	nCS2
N.C.	P2_28	P2_29	**48**	**46**	N.C.	P3_28	P3_29
GND	P2_30	P2_31	**51**	**49**	N.C.	N.C.	N.C.
GND	DP3V3	DP3V3	**54**	**52**	N.C.	N.C.	N.C.
nH_OC2[1]	nH_OC1[1]	H_DP2[1]	**57**	**55**	N.C.	N.C.	N.C.
H_DM2[1]	H_DP1[1]	H_DM1[1]	**60**	**58**	N.C.	nRST	nTRST
D_DP[1]	D_DM[1]	nH_PSW2[1]	**63**	**61**	TDI	TMS	TCK
nH_PSW1[1]	D_VBUS[1]	nGL[1]	**66**	**64**	RTCK	TDO	LED_N1[2]
GND	DP5V0	DP5V0	**69**	**67**	RX+[2]	RX-[2]	LED_N2[2]
N.C.	TIMEVCC	N.C.	**72**	**70**	TX+[2]	TX-[2]	LED_N3[2]

注：[1] 产品功能码不含 U，则此部分无连接（N.C.）;

[2] 产品功能码不含 N，则此部分无连接（N.C.）;

表 1.7 同样遵循此约定。

表 1.7　MiniARM 嵌入式工控模块引脚功能原型对照表

P0.0/TXD0/ PWM1	P0.1/RXD0/ PWM3/EINT0	P0.2/SCL/ CAP0.0	3	1	A0_B	A1_B	A2_B
P0.3/SDA/ MAT0.0/EINT1	P0.4/SCK0 /CAP0.1	P0.5/MISO0 /MAT0.1	6	4	A3_B	A4_B	A5_B
P0.6/MOSI0/ CAP0.2	P0.7/SSEL0/ PWM2/EINT2	P0.8/TXD1/ PWM4	9	7	A6_B	A7_B	A8_B
P0.9/RXD1/ PWM6/EINT3	P0.10/RTS1/ CAP1.0/RD5	P0.11/CTS1/ CAP1.1/TD5	12	10	A9_B	A10_B	A11_B
P0.12/DSR1/ MAT1.0/RD4	P0.13/DTR1/ MAT1.1/TD4	P0.14/DCD1/ EINT1	15	13	A12_B	A13_B	A14_B
P0.15/RI1/ EINT2	N.C.	P0.17/CAP1.2/ SCK1/MAT1.2	18	16	A15_B	A16_B	A17_B
P0.18/CAP1.3/ MISO1/MAT1.3	P0.19/MAT1.2/ MOSI1/CAP1.2	P0.20/MAT1.3/ SSEL1/EINT3	21	19	A18_B	A19_B	A20_B
P0.21/PWM5/ RD3/CAP1.3	P0.22/TD3/ CAP0.0/MAT0.0	P0.23/RD2	24	22	A21_B	A22_B	A23/ XCLK_B
P0.24/TD2	P0.25/RD1	TD1	27	25	D0_B	D1_B	D2_B
P0.27/AIN0/ CAP0.1/MAT0.1	P0.28/AIN1/ CAP0.2/MAT0.2	P0.29/AIN2/ CAP0.3/MAT0.3	30	28	D3_B	D4_B	D5_B
P0.30/AIN3/ EINT3/CAP0.0	AGND	AGND	33	31	D6_B	D7_B	D8_B
GND	P2.16	P2.17	36	34	D9_B	D10_B	D11_B
P2.18	P2.19	P2.20	39	37	D12_B	D13_B	D14_B
P2.21	P2.22	P2.23	42	40	D15_B	nBLS0_B	nBLS1_B
P2.24	P2.25	P2.26/BOOT0	45	43	nOE_B	nWE_B	nCS2/ RD6_B
P2.27/BOOT1	P2.28/RD6	P2.29/TD6	48	46	nCS3/ TD6	P3.28/ AIN7	P3.29/ AIN6
GND	P2.30/AIN4	P2.31/AIN5	51	49	N.C.	N.C.	N.C.
GND	DP3V3	DP3V3	54	52	N.C.	N.C.	N.C.
nH_OC2[1]	nH_OC1[1]	H_DP2[1]	57	55	N.C.	N.C.	N.C.
H_DM2[1]	H_DP1[1]	H_DM1[1]	60	58	N.C.	nRST	nTRST
D_DP[1]	D_DM[1]	nH_PSW2[1]	63	61	TDI	TMS	TCK
nH_PSW1[1]	D_VBUS[1]	nGL[1]	66	64	RTCK	TDO	LED_N1[2]
GND	DP5V0	DP5V0	69	67	RX+[2]	RX-[2]	LED_N2[2]
N.C.	TIMEVCC	N.C.	72	70	TX+[2]	TX-[2]	LED_N3[2]

1.3 系统接口框图

MiniARM 嵌入式工控模块（以 M9020-FNU20I 为例）的系统框图如图 1.1 所示，不同型号的产品系统结构框图有所不同。

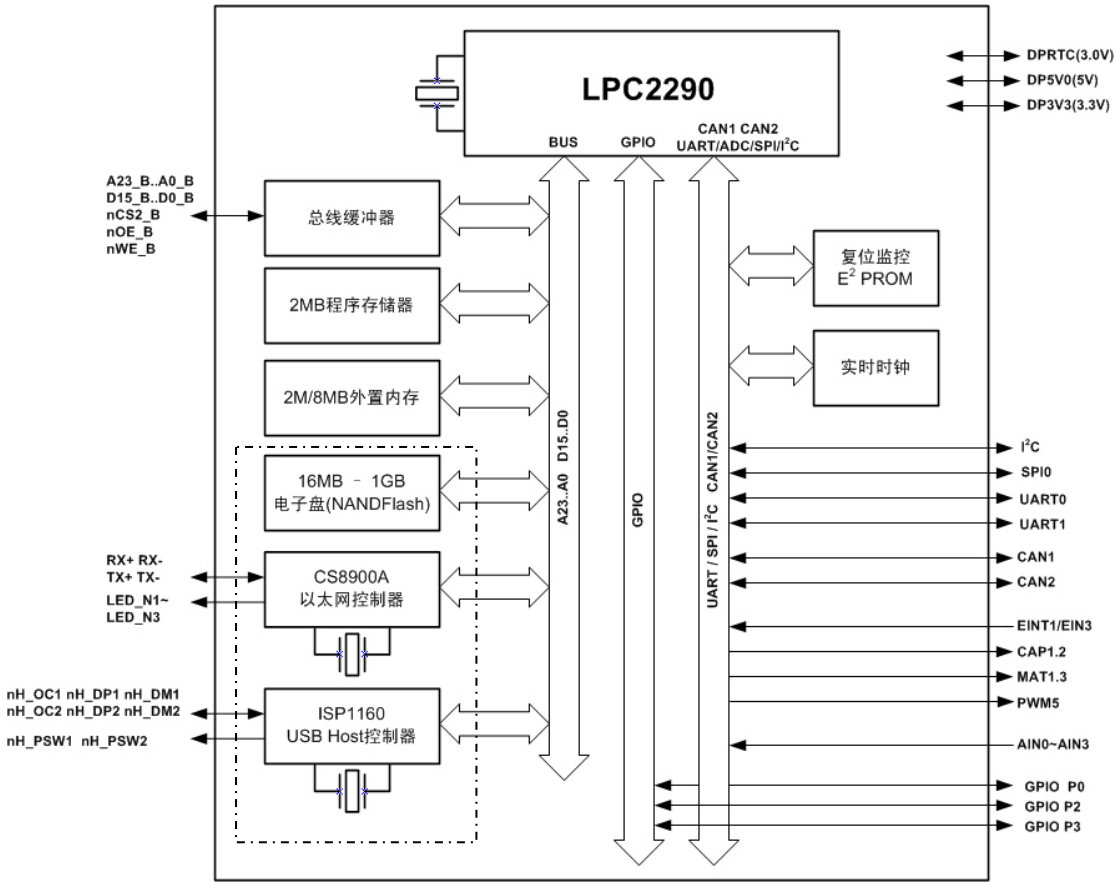

图 1.1 M9020-FNU20I 系统结构框图

注：不同型号的嵌入式工控模块，系统结构框图中虚线部分有所差别。

2. 开发环境

2.1 外围硬件设计

核心板含有大量的接口资源，必须设计可靠的外围电路与其配合。本手册给出部分外围电路的推荐设计方法。

注：对所有型号的嵌入式工控模块均适用。

2.1.1 电源电路设计

核心板需要同时供应 3.3V 电源和 5.0V 电源才能正常工作。对于 5.0V 电源，精度要求在±5%以内，典型电流消耗为 70mA。LM2575 具有输入电压范围宽，输出电流大（1A），耗散功率小的优点，可以满足核心板要求。对于 3.3V 电源，精度要求在±10%以内，典型电流消耗为 90mA，SPX1117 系列 LDO 芯片输出电流可达 800mA，输出电压的精度在＋1%以内，具有电流保护和过热保护功能，广泛应用于手持式仪表、数字家电和工业控制等领域。使用 SPX1117 为核心板供电时，输出端需接入≥10μF 钽电容用于改善瞬态响应和稳定性。

推荐电源电路设计如图 2.1 所示。

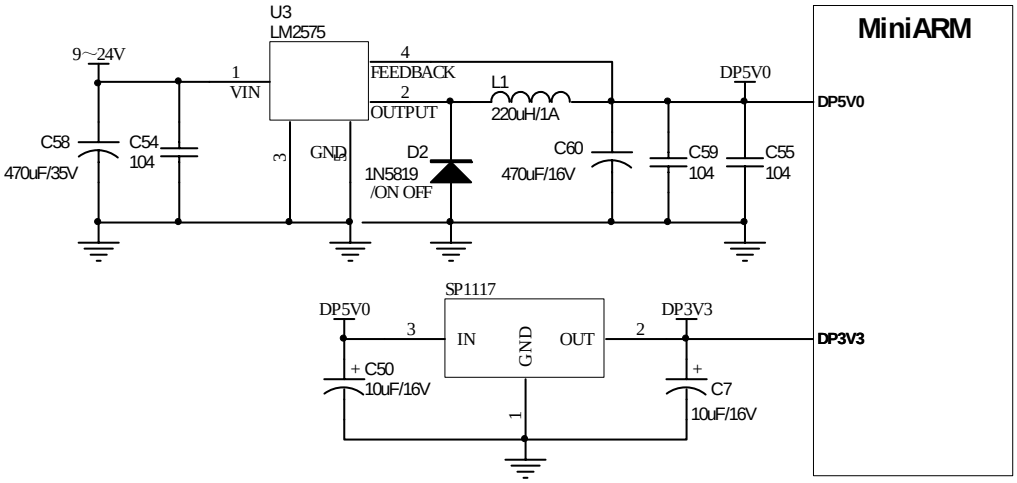

图 2.1 电源电路

2.1.2 复位电路设计

核心板上自带有复位电路，复位门槛电压为 2.93V。如果 DP3V3 电源电压低于 2.93V 时，CPU 处于复位状态，只有高于 2.93V 时，CPU 才能够脱离复位状态。如果用户需要手动复位，只需在核心板上复位引脚与 GND 之间连接一按键，如图 2.2 所示。按下按键后，会强制拉低 CPU 复位引脚，从而达到使系统复位的目的。

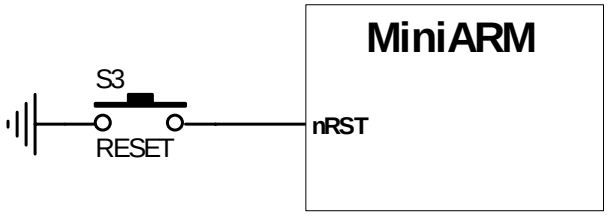

图 2.2 手动复位电路

2.1.3 ISP 电路设计

MiniARM 嵌入式工控模块含有 ISP 功能，因此需要设计 ISP 电路，如图 2.3 所示。如果在系统复位期间，JP1 跳线短接，那么模块将运行自身的 Boot 代码，而不会运行用户程序，因此，如果用户需要运行自己的程序，在复位期间，需要断开 JP1 跳线。

在使用 JTAG 调试程序期间，建议短接该跳线。程序调试完毕后脱机运行时，JP1 跳线必须要断开。

ISP 的引脚与 USRINT0 是复用的。

注：ISP 引脚在核心板上已经外接了上拉电阻。

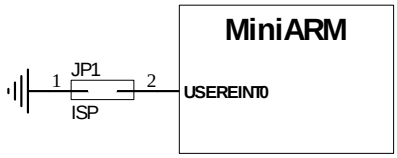

图 2.3 ISP 电路

2.1.4 调试端口设计

MiniARM 嵌入式工控模块采用 ARM 公司定义的标准 20 脚 JTAG 仿真调试接口，JTAG 信号定义及与模块的连接如图 2.4 所示。

模块在复位期间，如果引脚 RTCK 为低，将使能 JTAG 调试引脚。引脚 RTCK 内部含有上拉电阻，用户需外接一下拉电阻，推荐的阻值为 4.7K。

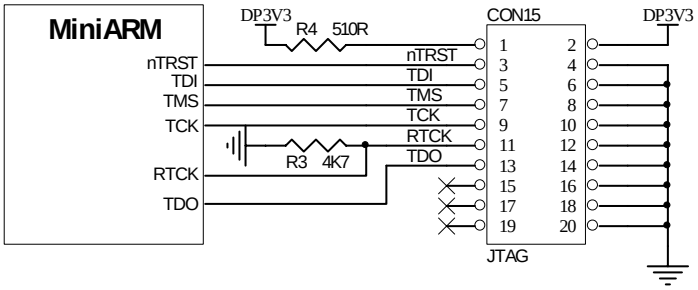

图 2.4 JTAG 接口电路

2.1.5 UART 接口设计

MiniARM 嵌入式工控模块提供 2 路通用异步串行接口—UART0、UART1，其中 UART1 符合 16C550 标准 Modem 接口，UART0 为三线制串行接口。UART 接口电平为 3.3V 逻辑，用户可以采用电平传输，也可以使用 SP3232E、SP3243EEA 等电压转换芯片来实现 RS-232C 传输。支持波特率最高达 200K。使用 UART0 实现三线制 RS-232C 接口通信时，推荐外围电路如图 2.5 所示。SP3232E 系列是一个 2 驱动器/2 接收器的低功耗器件，内部含有一个高效的电荷泵，工作电压为 3.3V 时只需 0.1μF 电容即可完成稳定的 RS-232C 通信。电荷泵允许 SP3232E 系列在 3.3V 到 5.0V 内的某个电压下发送符合 RS-232C 的信号。

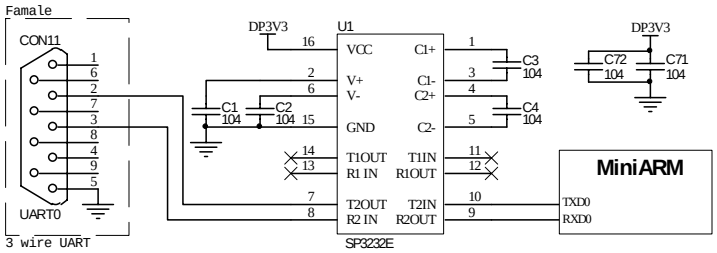

图 2.5 UART 接口电路

2.1.6 I²C 接口设计

MiniARM 嵌入式工控模块含有 1 路 I²C 接口，通信时固定为主机模式，速率 100KHz。所以，如果用户需要外接 I²C 接口的从设备，只需要将其挂接在 I²C 总线上即可，如图 2.6 所示。模块内部 I²C 总线已含有上拉电阻，外部不必增加上拉电阻。

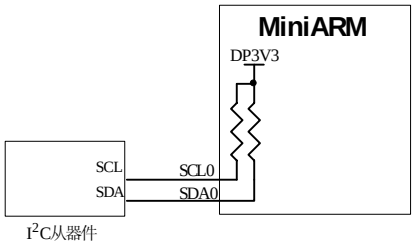

图 2.6 I²C 接口设计

需要注意一点：模块内部含有 3 个 I²C 总线器件，总线地址分别为 0x80、0xA0 和 0xA2，其中 0xA0 为 CAT1025 的器件地址，0xA2 为实时时钟 PCF8563 的器件地址，0x80 为系统占用，这个器件地址用户不能访问。

2.1.7 SPI 接口设计

MiniARM 嵌入式工控模块提供 1 个 SPI 接口，推荐设计为主机模式使用。与从机设备通信时，需要额外选择 GPIO 作为片选信号线，选择对应的设备。同时，作为主机时自身的片选信号线 SSEL0 需外接上拉电阻。

注： MiniARM 嵌入式工控模块为 3.3V 系统；SPI 接口速率最大值为：1.38MHz。

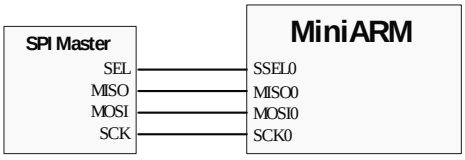

图 2.7 SPI 从模式接口设计

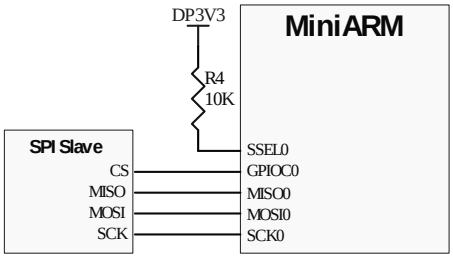

图 2.8 SPI 主模式接口设计

2.1.8 CAN-bus 接口设计

完整的 CAN 总线设备由 CAN 控制器、总线收发器以及相应的隔离电路组成。推荐 MiniARM 嵌入式工控模块使用如图 2.9 所示 CAN 接口电路（图中只有 1 路），图中 U1 为隔离 CAN 收发器模块 CTM8251，能确保 CAN 总线在遭受严重干扰时控制器仍正常工作。U2 为总线 ESD 保护器件，可以使用 NUP2105L 等。

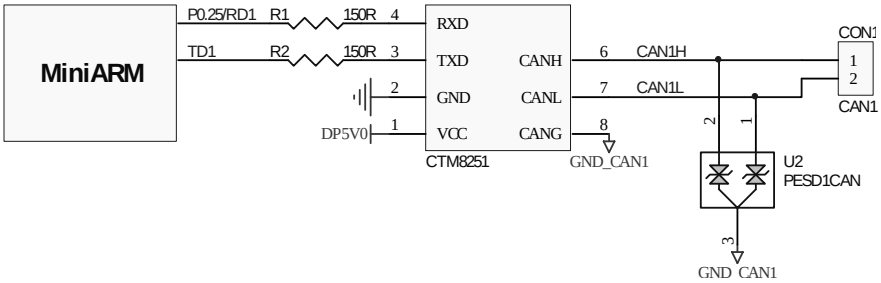

图 2.9 CAN 总线接口电路

致远 CTM 系列 CAN 隔离收发模块集电源隔离、电气隔离、CAN 收发器、CAN 总线保护于一体，模块 TXD、RXD 引脚兼容+3.3V、+5V 电平连接，毋需外接其他元器件，直接将 CAN 控制器收发引脚与 CTM 模块的收发引脚连接即可构成完整的 CAN 设备。

注：只有 M9020/9080 系列工控模块支持 CAN 通信功能
关于 CTM 模块更详细的信息请联系致远电子销售部门。

2.1.9 外部总线接口设计

MiniARM 嵌入式工控模块设计为开放型外部总线接口，外部总线可以连接外部存储器（支持 SRAM、ROM、FLASH）或者 ADC/DAC 等器件。具有 24 根地址总线 A0_B ～ A23_B（即，A[0：23]），16 根数据总线 D0_B ～ D15_B（即，D[0：15]），读使能信号 nOE_B（即，OE）、写使能信号 nWE_B（即，WE）、 字节定位选择信号 nBLS0_B 和 nBLS1_B，片选信号为 nCS2，地址范围为 0x8200 0000～0x82FF FFFF。

模块支持 16 位总线器件和 8 位总线器件，16 位总线器件的连接示意图如图 2.10 所示，其中，UB、LB 为器件的字节定位信号，并非所有的 16 位器件都含有这两个信号，如 16 位 SRAM 器件 IS61LV6416 含有字节定位信号，而 16 位 NOR FLASH 器件 SST39VF1601 就没有字节定位信号，如果没有字节定位信号，这两个信号线也就不必连接了。在扩展 16 位总线器件时，地址线 A0 无用。

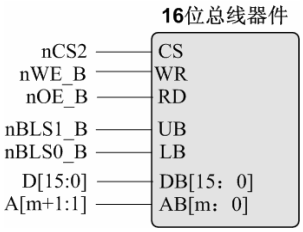

图 2.10 16 位总线器件连接示意图

在外部总线上也可以扩展 8 位总线器件，8 位总线器件的连接示意图如图 2.11 所示。对于 8 位总线器件来说，就没有字节定位信号了，此时，总线器件的写使能信号可以使用 nWE_B 也可以使用 nBLS0_B。

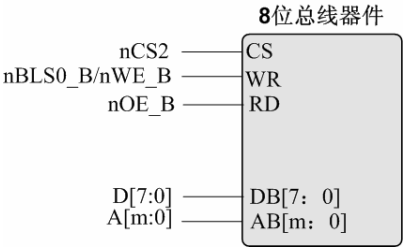

图 2.11 8 位总线器件连接示意图

对于 MiniARM 嵌入式工控模块来说，外部总线的访问速度是可以调整的，如图 2.12、图 2.13 所示，WST1 和 WST2 分别控制外部总线的读、写访问速度。有关外部总线的操作请参考 MiniARM 专用工程模板。

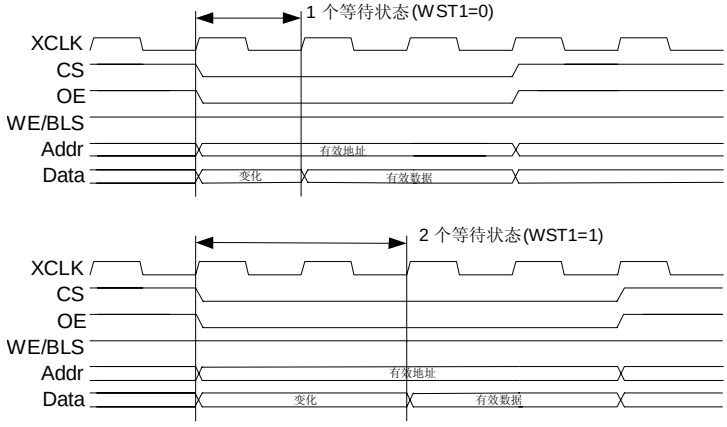

图 2.12 部存储器读访问（WST1=0 和 WST1=1 两种情况）

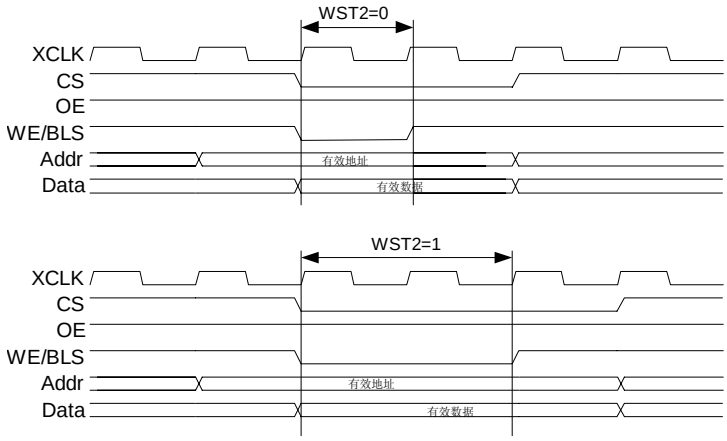

图 2.13 外部存储器写访问（WST2=0 和 WST2=1 两种情况）

2.1.10 以太网接口设计

MiniARM 嵌入式工控模块集成了 10M 工业级以太网控制器，用户只需外接状态指示灯、网络变压器、匹配电阻、高压电容和 RJ45 插座。

特别注意：DM9000E 和 CS8900A 网络变压器型号不同，连接方式也稍有差异。用户在设计底板的时候必须以本手册为准，不建议使用 MiniARM M22A Series EV Board 评估板的

10M/100M 通用接口电路方案。如果用户采用内部不带 LED 的 RJ45 插座，可以参考图 2.14，如果使用内部集成 LED 的 RJ45 插座，请参考图 2.15。

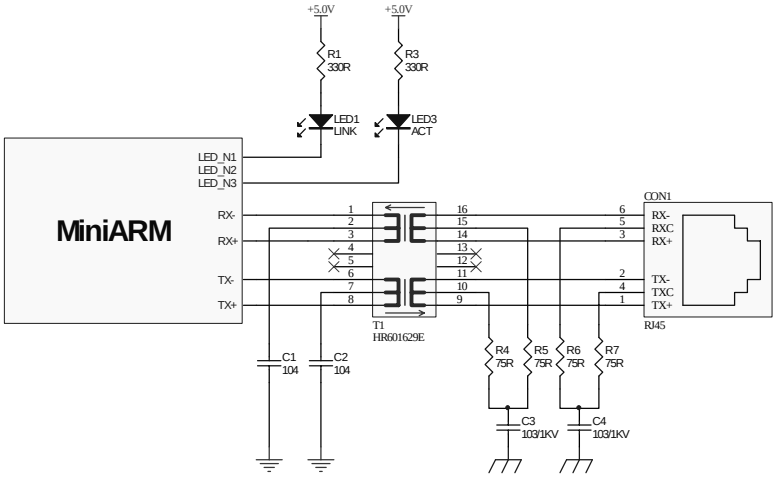

图 2.14 以太网接口电路（一）

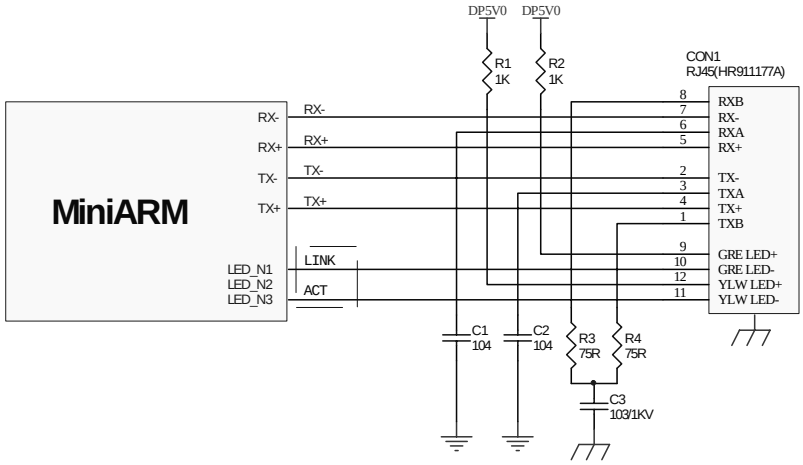

图 2.15 以太网接口电路（二）

注：以太网通信具体操作见《以太网通信函数参考手册》。

2.1.11 USB Host 接口设计

MiniARM 嵌入式工控模块提供 2 路 USB Host，接口电路采用 5V 供电，具备电源开关、电流限制保护功能。设计电路时须注意以下几点：

- U 盘最大电流消耗要求不大于 500mA（与+5V 电源负载有关）；
- EMC 测试中建议电容 C11 耐压为 1.5KV~2.0KV；
- PCB 铺地设计中必须考虑不同地线的最小间距以及板材介电常数，以免地线之间产生放电现象。

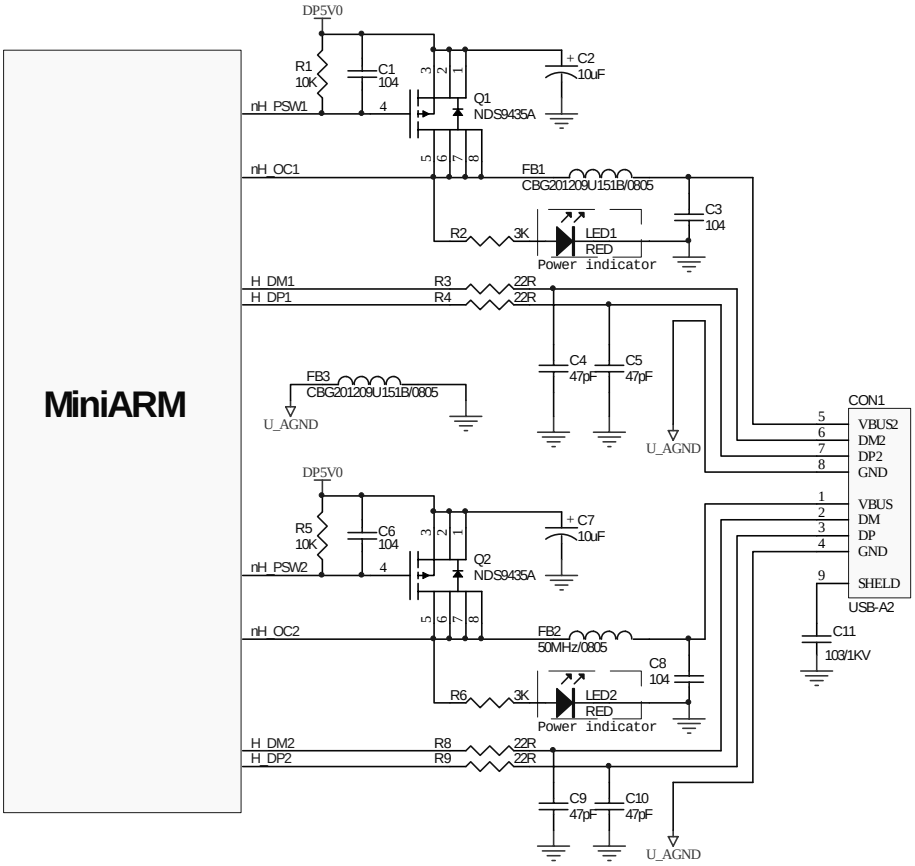

图 2.16　USB Host 接口电路

注：USB 具体操作见《文件管理系统及相关存储设备函数参考手册》。

2.1.12　用户资源列表

可供用户使用的资源列表如表 2.1 所示。

表 2.1 用户资源列表

资源分类	标配系统资源	备注
IRQ 中断优先级	6～15	0～5 系统占用
CPU 硬件接口	I^2C 接口	● 只能够用作主模式，通信速率默认为 100KHz ● 用户不可对 I2C 速率进行更改 ● I2C 引脚功能不可更改 ● 用户不可以访问总线地址 0x80
	SPI 接口	主、从模式都可以（推荐使用主机模式）
	UART0	波特率最低不得低于 200，最高不得高于 200K
	UART1	
	定时器 1	● 定时器 1 捕获引脚：CAP0 （CPU 引脚：P0.17/CAP1.2/SCK1/MAT1.2） ● 定时器 1 匹配输出引脚：MAT0 （CPU 引脚：P0.18/CAP1.3/MISO1/MAT1.3）
	CAN1	引脚：TD1、RD1（仅适用 LPC2290）
	CAN2	引脚：TD2、RD2（仅适用 LPC2290）
	PWM0	CPU 引脚：P0.21/PWM5/RD3/CAP1.3

	外部中断 1	CPU 引脚：P0.14/DCD1/EINT1
	外部中断 3	CPU 引脚：P0.20/MAT1.3/SSEL1/EINT3
	模拟电压输入 AIN0～AIN3	● 参考电压：3.3V ● 输入测量电压最大值 3.3V
用户 GPIO	P2_16	P2.16
	P2_17	P2.17
	P2_18	P2.18
	P2_19	P2.19
	P2_20	P2.20
	P2_21	P2.21
	P2_22	P2.22
	P2_23	P2.23
	P2_24	P2.24
	P2_25	P2.25
	P2_28	P2.28
	P2_29	P2.29
	P2_30	P2.30
	P2_31	P2.31
	P3_28	P3.28
	P3_29	P3.29
	P0_12	P0.12/DSR1/MAT1.0/RD4
	P0_13	P0.13/DTR1/MAT1.1/TD4
	P0_19	P0.19/MAT1.2/MOSI1/CAP1.2
	P0_22	P0.22/TD3/CAP0.0/MAT0.0
FLASH	NOR FLASH	0x8000 0000 ～ 0x801F FFFF（2MB）
RAM 空间	片外 PSRAM	0x8100 0000 ～ 0x811F FFFF（2MB）或 0x8100 0000 ～ 0x817F FFFF（8MB）

2.2 用户工程模板

在 MiniARM 嵌入式工控模块中已经固化了文件系统、TCP/IP 协议栈、USB 协议栈以及 μC/OS-II 源代码等，致远电子为用户提供专用的开发工程模板，模板已经完成操作所有的系统配置，以 M9020-FNU20I 为例，其中包含的文件如表 2.2 所示。

表 2.2 M9020-FNU20I 工程模板文件一览表

文件夹		文件	描述
ZY_CODE	ZY_CODE 为系统代码，一般不需用户更改		
	Head	config.h	模板配置头文件
		Includes.h	μC/OS-II（v2.52）的包含文件
		LPC2294.h	LPC2000 头文件
		OS_CPU.H	LPC2220 配置部分的移植代码
		target.h	系统初始化函数头文件
		arithmetic.h	工程常用算法接口函数
	uCOSII	uCOS_II.h	操作系统头文件
		OS_CFG . h	μC/OS-II（v2.52）的配置文件
	NET	CFG_NET.h	以太网配置文件
		rcf_socket.h	Socket API 函数头文件
		Connect_Test.c Connect_Test.h	网络配置函数
	USB	HostStack.h	海量存储设备头文件
		USBDriver.h	USB 驱动程序头文件
	NandFlash	NandFlashInfo.c、 NandFlashInfo.h	NandFlash 电子盘的芯片结构信息
	FileSys	OSFile.h	文件系统 API 函数头文件
	Startup	Startup.s	系统启动代码
		target.c	系统初始化函数
	Scf	RelOutChipa、 RelOutChipb、 RelOutChipc	分散加载文件
	M9020-FNU20I	system.a、 system.h	系统函数声明
	ZLGDriver2200_Lib	----	LPC2200 基础驱动库
USER_CODE	USER _CODE 为用户代码，建议用户的代码均放于此目录，以便升级和维护		
	PINCFG	LPC2200PinCfg.c	LPC2200 系列 CPU 引脚连接与模式配置文件
		LPC2200PinCfg.h	
		ISR.c	用户 FIQ 和 DefIRQ 中断响应程序
		main.h	用户头文件声明
		main.c	用户主文件

注：不同型号的工控模块对应的模板包含的文件有所差别。

工程模板的项目工程目录如图 2.17 所示，其中含有 3 个编译选项：DebugInLowAddr、DebugInHighAddr 和 Recovery。DebugInLowAddr 用于生成存储在低区的用户程序，DebugInHighAddr 用于生成存储在高区的用户程序， Recovery 用于工控模块固件恢复，如表 2.3 所示。

表 2.3 编译选项说明

编译选项	说明
DebugInLowAddr	用户程序空间：0x8002 0000 ～ 0x801F FFFF 只有使用 DebugInLowAddr 模式才能够进行 JTAG 调试，生成 LowAddr.bin
DebugInHighAddr	用户程序空间：0x8011 0000 ～ 0x801F FFFF 不可用于调试，仅用于生成 HighAddr.bin 以配合用户程序升级
Recovery	用于工控模块固件恢复，可用于调试，但不推荐

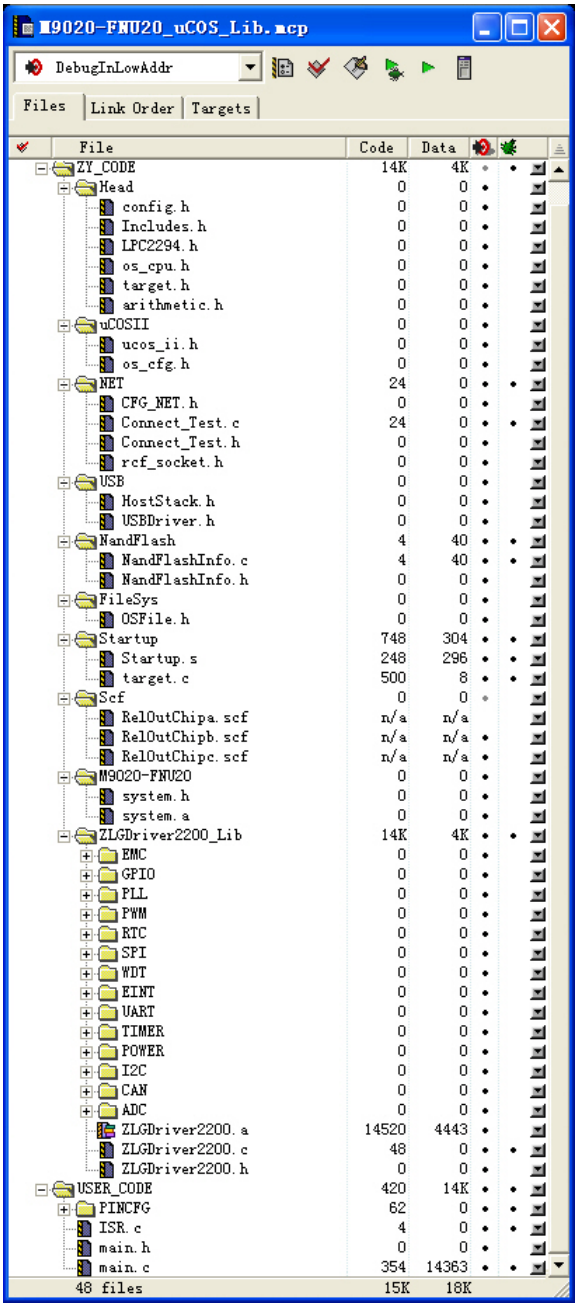

图 2.17 工程模板项目目录

注：使用 DebugInHighAddr 选项生成的代码不能进行调试。

2.2.1 模板参数设置

MiniARM 嵌入式工控模块对应的工程模板内部含有一些默认的配置信息，用户在使用时，可以根据自身的需要作一些适当的调整。

2.2.1.1 总线配置

MiniARM 嵌入式工控模块上的 CPU 含有 4 个外部存储器控制器，即总线控制器—Bank0、Bank1、Bank2 和 Bank3。在核心板上已经将 Bank 0 分配给 NOR Flash，Bank1 分配给 PSRAM，Bank3 分配给 NAND Flash 电子盘（功能代码含 F）、以太网控制器（功能代码含 N）和 USB 控制器（功能代码含 U）。对于用户来说，只能使用 Bank2 来控制外部总线接口。表 2.4 所示为每一个控制器所对应的总线地址。

表 2.4 总线控制器所对应的总线地址

Bank	地址范围
0	8000 0000－80FF FFFF
1	8100 0000－81FF FFFF
2	8200 0000－82FF FFFF
3	8300 0000－83FF FFFF

每一个控制器都含有一个配置寄存器，用来配置对应的总线，主要包括：总线宽度、两个操作之间空闲的 CPU 周期数（IDCY）、读操作所占用的 CPU 周期数（WST1）、写操作所占用的 CPU 周期数（WST2），如图 2.18 和图 2.19 所示。

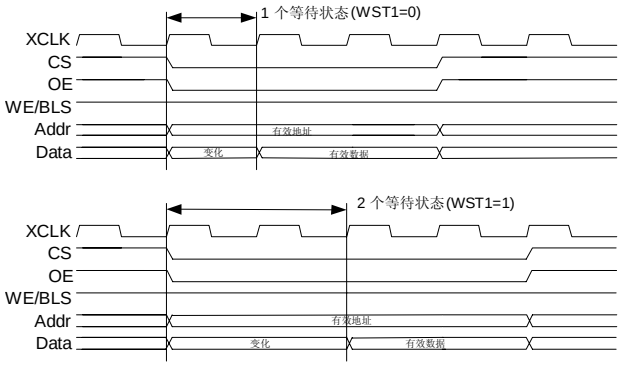

图 2.18 总线读访问（WST1=0 和 WST1=1 两种情况）

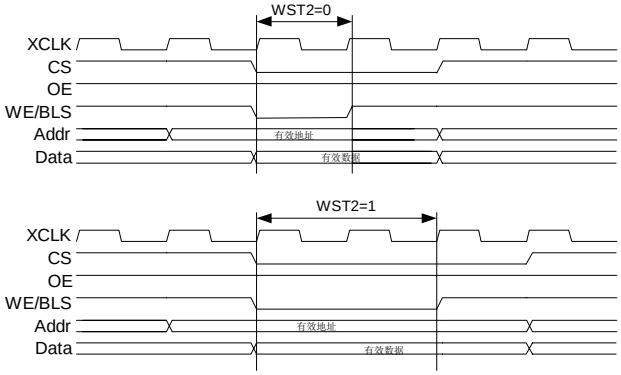

图 2.19 总线写访问（WST2=0 和 WST2=1 两种情况）

一次完整的读访问时间长度：WST1＋3（CPU 周期——cclk）
一次完整的写访问时间长度：WST2＋3（CPU 周期——cclk）

在文件 Startup.s 中已经设定了一个默认值，如程序清单 2.1 所示。

程序清单 2.1 系统总线配置

```
BCFG_08DEF   EQU      0x00000400        ;//  8Bit Bus，8 位总线宽度
BCFG_16DEF   EQU      0x10000400        ;// 16Bit Bus，16 位总线宽度
BCFG_32DEF   EQU      0x20000400        ;// 32Bit Bus，32 位总线宽度。
;//                                    |     IDCY      |     WST1      |     WST2
;//                                    | Idle width | Read width | Write width
;//                                    | 0x00 ~ 0x0f| 0x00 ~ 0x1f| 0x00~0x1f
;// Bank0 For Flash
BCFG_FLASH   EQU      (BCFG_16DEF | (0x00<<00) | (0x03<<05) | (0x03<<11))
;// Bank1 For SRAM
BCFG_PSRAM   EQU      (BCFG_16DEF | (0x00<<00) | (0x03<<05) | (0x03<<11))
;// Bank2
BCFG_CS2     EQU      (BCFG_16DEF | (0x0f<<00) | (0x1f<<05) | (0x1f<<11))
;// Bank3 For Peripheral Equipment
BCFG_CS3     EQU      (BCFG_16DEF | (0x08<<00) | (0x0f<<05) | (0x0f<<11))
```

从程序清单 2.1 可见，如果 CPU 时钟频率为 44MHz，那么对于 Bank0 来说，由于 WST1＝WST2＝3，则表示读写 Flash 的时间长度为 136nS，16 位总线宽度；Bank1 中，由于 WST1＝WST2＝3，表示读写 PSRAM 的时间长度也为 136nS，16 位总线宽度。

在 Bank2 中，默认的读、写访问时间是最慢的，为 772.7nS，16 位总线宽度。用户可以根据自己的实际情况来进行调整。

对于 Bank0、Bank1 和 Bank3，用户是绝对不能调整总线配置的，用户只能够调整 Bank2 所对应的总线。
同时，我们只支持 8 位、16 位总线，不支持 32 位总线。

2.2.1.2 config.h 文件配置信息

config.h 文件包含系统频率设置和应用设置两部分。其中系统频率设置关系到整个系统的主频，用户请勿自行调整，如程序清单 2.2 所示。

程序清单 2.2 工程模板系统频率设置

```
/* 系统设置, Fosc、Fcclk、Fcco、Fpclk 必须定义 */
#define    Fosc  11059200              // 晶振频率,10MHz~25MHz，应当与实际一致
#define    Fcclk (Fosc * 4)            // 系统频率，必须为 Fosc 的整数倍(1~32)，且<=60MHZ
#define    Fcco  (Fcclk * 4)           // CCO 频率，必须为 Fcclk 的 2、4、8、16 倍
#define    Fpclk (Fcclk / 4) * 1       // VPB 时钟频率，只能为(Fcclk / 4)的 1、2、4 倍
```

为了方便用户调试，以 M22A 评估板为例，在 main.h 中设置了 LED、蜂鸣器、KEY1 和 KEY2 的控制引脚。如程序清单 2.3 所示。

程序清单 2.3 工控模块 I/0 口设置

```
#define    LED9      P0_19      // P0.19
#define    BUZZER    P0_21      // P0.21
#define    KEY1      P0_20      // P0.20
#define    KEY2      P0_22      // P0.22
```

2.2.1.3 µC/OS-II 配置

虽然 µC/OS-II 是一个可裁剪的实时多任务内核，但是，我们已经将其全部优化配置并固化到 MiniARM 嵌入式工控模块中，因此用户不能对 µC/OS-II 内核进行配置，只能使用已设置好的配置信息，如表 2.5 所示，有关更详细的配置信息，用户可参考 OS_CFG.h 文件。同时，任务的优先级 0～15 用户不能够使用，这 16 个优先级是系统保留的，可供用户使用的优先级为 17～61。

表 2.5 µC/OS-II 部分配置信息

类别	配置选项	配置信息
事件控制块的最大数目	OS_MAX_EVENTS	200
事件标志的最大数目	OS_MAX_FLAGS	10
消息队列的最大数目	OS_MAX_QS	10
最多任务数	OS_MAX_TASKS	63
最低任务优先级	OS_LOWEST_PRIO	63
空闲任务的堆栈容量	OS_TASK_IDLE_STK_SIZE	128 字＝512 Bytes
统计任务使能设置	OS_TASK_STAT_EN	使能统计任务
统计任务的堆栈容量	OS_TASK_STAT_STK_SIZE	48 字＝192 Bytes
系统时钟频率	OS_TICKS_PER_SEC	200Hz，定时器 0 的定时时间为 5mS

2.2.2 system 函数

system 函数是 MiniARM 嵌入式工控模块的系统函数，表 2.6 为系统函数一览表

表 2.6 系统函数一览表

功能类型	函数名称	函数描述
快速中断	FIQDisable	全局禁止 FIQ 中断
	FIQEnable	全局允许 FIQ 中断
	FIQSave	全局禁止 FIQ 中断的同时返回 FIQ 的状态
	FIQRestore	与 FIQSave ()配对使用，恢复全局 FIQ 状态
	SetVICFIQ	使能所选中断通道的 FIQ 中断
向量 IRQ 中断	SetVICIRQ	设置所选中断通道的中断服务地址并使能中断
非向量 IRQ 中断	SetDefIRQAddr	设置非向量中断服务程序地址
	EnableDefIRQ	使能所选中断通道的非向量中断，必须在调用 SetDefIRQAddr 后使用
I^2C 相关操作	SysInit	固件程序初始化

	I2CWrite	向 I^2C 从器件写数据
	I2CRead	从 I^2C 从器件读数据
	I2CGetFlag	获取 I^2C 总线状态
	I2CGetWriteBytes	获得未写完的数据个数
	I2CGetReadBytes	获得未读完的数据个数
IP 地址操作——获取	GetIpSet	获得本机 IP 设置
	GetMarkSet	获得本机子网掩码设置
	GetGateWaySet	获得本机网关设置
	GetPortSet	获得本机端口设置
	GetServerIpSet	获得服务器 IP 设置
	GetServerPortSet	获得服务器端口设置
IP 地址操作—保存	SaveIpSet	保存本机 IP 设置
	SaveMarkSet	保存本机子网掩码设置
	SaveGateWaySet	保存本机网关设置
	SavePortSet	保存本机端口设置
	SaveServerIpSet	保存服务器 IP 设置
	SaveServerPortSet	保存服务器端口设置
IAP 相关操作	BeginUpgrade	开始升级
	EndUpgrade	升级结束
	WordProgram	升级程序字节编程
	WordUpProgram	
	WordCurProgram	当前程序字节编程
	WordUpRead	升级程序半字读
	WordCurRead	当前程序半字读
	GetUpSecIndex	获得指定升级程序偏移所在的扇区号
	GetCurSecIndex	获得指定程序偏移所在的扇区号
	SectorErase	擦除扇区
其它	SetDataAbortISR	设置数据中止异常服务地址
	SetPrefetchISR	设置指令中止异常服务地址
	SetUndefinedSR	设置未定义指令异常服务地址
	OSGetPrio	返回当前任务的优先级
	GetCodeVer	获取固件版本

2.2.2.1 快速中断

MiniARM 嵌入式工控专用工程模板已经将 FIQ 的设置整理成一个简单的接口函数，用户只需要调用这些函数即可。

表 2.7　FIQDisable 函数

函数原型	void　FIQDisable(void)
函数功能	全局禁止 FIQ 中断
入口参数	无
出口参数	无
调用示例	FIQDisable();　// 禁止 FIQ 中断

表 2.8 FIQEnable 函数

函数原型	void FIQEnable(void)
函数功能	全局允许 FIQ 中断
入口参数	无
出口参数	无
调用示例	FIQEnable (); // 允许 FIQ 中断

表 2.9 FIQSave 函数

函数原型	unsigned int FIQSave(void)
函数功能	全局禁止 FIQ 中断的同时返回 FIQ 的状态
入口参数	无
出口参数	FIQ 的状态
说明	与 FIQRestore()配对使用

调用示例	state = FIQSave(); // 禁止 FIQ 中断，同时返回程序状态字
	FIQRestore(state); // 允许 FIQ 中断，同时恢复程序状态字的其它位

表 2.10 FIQRestore 函数

函数原型	void FIQRestore(unsigned int psr)
函数功能	恢复全局 FIQ 状态
入口参数	psr FIQSave()返回的值
出口参数	无
说明	与 FIQSave ()配对使用

调用示例	state = FIQSave(); // 禁止 FIQ 中断，同时返回程序状态字
	FIQRestore(state); // 允许 FIQ 中断，同时恢复程序状态字的其它位

表 2.11 SetVICFIQ 函数

函数原型	void SetVICFIQ(uint32 channel)
函数功能	使能所选中断通道的 FIQ 中断
入口参数	channel 中断通道号
出口参数	无
说明	无
调用示例	SetVICFIQ(5); // 使能 TIMER1 为 FIQ 中断

用需在 ISR.c 文件中的 FIQ_Exception 函数内添加 FIQ 服务代码，用户按照我们提供的方法可以很方便的实现 FIQ 的功能。编写 FIQ 的步骤：

- 设置中断源，将中断源设置为 FIQ 中断
- 使能 FIQ——FIQEnable();（可选）
- 在 FIQ_Exception (void)函数中添加 FIQ 服务代码（FIQ_Exception 函数在 ISR.c 文件中定义）

下面介绍一个 FIQ 的例子：使用 ZLGDriver2200 基础驱动库，设置定时器 1 定时 0.5 秒发生 FIQ 中断，控制蜂鸣器的鸣叫。核心板的底板使用 M22A 评估板 M22 Series EV Board。如程序清单 2.4 所示。

程序清单 2.4 定时器 1 设置操作

文件 1：“LPC2200PinCfg.h”

```
#define   P0_07_FNUC                P0_07_GPIO     // P0.07 引脚设置为 GPIO
```

文件 2：“main.c”

```
……
PinInit();                                          // 引脚初始化
……
void TASK1(void *pdata)
{
    pdata = pdata;
    TimerInit(TIMER1,"Mode=0",NULL);                // 定时器 1 初始化
    TimerSetMode(TIMER1,SET_TIMERMODE,"Time=500000 ActCtrl=3");
                                                    // 设置定时器 1 的工作模式为定时模式：
                                                    // Time=500000：定时时间为 500ms
                                                    // ActCtrl=3：匹配后将 T1TC 复位,并产生中断
    SetVICFIQ (TIMER1_IRQ_CHN);                     // 中断设置为 FIQ
    TimerStart(TIMER1,NULL);                        // 启动定时器
    while (1) {
        OSTimeDly(OS_TICKS_PER_SEC);
    }
}
```

至此，有关 FIQ 的设置工作都已经完成，接下来需要编写 FIQ 服务代码，FIQ 的服务代码需要在函数 FIQ_Exception 中添加，如程序清单 2.5 所示。

程序清单 2.5 FIQ_Exception 函数

```
void FIQ_Exception(void)
{
    /* 1.添加用户 FIQ 中断服务代码 */
    // 2.清除中断标志位
    GpioCpl(P0_07);
    TimerISR(TIMER1);                                   // 清除中断标志
}
```

2.2.2.2 向量 IRQ 中断

同 FIQ 一样，模板也将 IRQ 的设置整理成一个非常简单的接口，用户可以像编写普通函数一样来编写向量 IRQ 服务程序。需要注意的一点是，IRQ 含有 16 个优先级（0～15），但是，系统函数已经占据了前 6 个中断优先级（0～5），所以，可供用户使用的中断优先级只有 6～15。如果用户使用前 6 个优先级（0～5），那么系统可能会出现意想不到的后果。

中断优先级的数值越大，优先级越低。因此，最高的优先级是 0＃优先级，最低的优先级是 15＃优先级。

表 2.12 SetVICIRQ 函数

函数原型	void SetVICIRQ(uint32 channel,uint32 PRI,uint32 ISRFuction)
函数功能	设置向量中断服务程序
入口参数	Channel 中断号 PRI 中断优先级 ISRFuction 中断服务程序（普通 C 语言函数）
出口参数	无

调用示例	__inline void UART0_ISR(void)
	{
	UartISR(UART0);
	//-----添加用户代码------//
	//----------------------//
	VICVectAddr = 0x00; // 中断处理结束
	}
	// UART0 中断设置为 IRQ 模式，中断优先级为 7#
	SetVICIRQ (UART0_IRQ_CHN , 7, (uint32) UART0_ISR);

注：对于 MiniARM 嵌入式工控模块来说，系统已经占用了 IRQ 的前 6 个优先级（0～5），用户能够使用的优先级为：6～15;

向量 IRQ 中断服务程序的编写与普通函数一样，只是增加了中断结束标志。

2.2.2.3 非向量 IRQ 中断

非向量 IRQ 的设置只需要调用 2 个接口函数即可，如表 2.13 所示。用户可以像编写普通函数一样来编写非向量 IRQ 服务程序。

表 2.13 SetDefIRQAddr 函数

函数原型	void SetDefIRQAddr(void *Function)
函数功能	设置非向量中断服务程序
入口参数	Function 中断服务程序（普通 C 语言函数）
出口参数	无

调用示例	void DefIRQ_Exception(void)
	{
	……
	}
	// 非向量 IRQ 的中断服务程序为 DefIRQ_Exception，在 ISR.c 中定义
	SetDefIRQAddr ((void *)DefIRQ_Exception);

设置好非向量中断服务地址后，必须调用表 2.14 所示函数，使能非向量中断。

表 2.14 EnableDefIRQ 函数

函数原型	void EnableDefIRQ(uint32 channel)
函数功能	使能所选中断通道号的非向量中断
入口参数	channel ：外设对应的中断通道号
出口参数	无
调用示例	EnableDefIRQ (5); // 设置定时器 1 为非向量中断

2.2.2.4 IP 地址操作

在功能代码含 N 的嵌入式工控模块中，将本机的 IP 地址、网关地址、子网掩码、本地端口、服务器 IP 地址和服务器端口信息全部保存在了非易失存储器中，用户可以通过相关的接口函数来访问这些信息。

表 2.15 GetIpSet 函数

函数原型	uint8 GetIpSet(uint8 *ip)
函数功能	获得本机 IP 设置
入口参数	ip 用来保存本机 IP 地址的缓冲区
出口参数	TRUE 成功 FALSE 失败

调用示例	uint8 MCU_IP[4];
	// 获得本机的 IP 地址，并将 IP 地址保存到 MCU_IP 数组中
	GetIpSet(MCU_IP);

表 2.16 GetMarkSet 函数

函数原型	uint8 GetMarkSet(uint8 *mark)
函数功能	获得本机子网掩码设置
入口参数	mark 用来保存本机子网掩码的缓冲区
出口参数	TRUE 成功 FALSE 失败

调用示例	uint8 MCU_Mark[4];
	// 获得本机的子网掩码，并将子网掩码保存到 MCU_ Mark 数组中
	GetMarkSet(MCU_Mark)

表 2.17 GetGateWaySet 函数

函数原型	uint8 GetGateWaySet(uint8 *GateWaySet)
函数功能	获得本机网关设置
入口参数	GateWaySet 用来保存本机网关地址的缓冲区
出口参数	TRUE 成功 FALSE 失败

调用示例	uint8 MCU_GateWay[4];
	// 获得本机的网关地址，并将网关地址保存到 MCU_ GateWay 数组中
	GetGateWaySet(MCU_GateWay);

表 2.18 GetPortSet 函数

函数原型	uint16 GetPortSet(void)
函数功能	获得本机端口设置
入口参数	无
出口参数	本机端口

调用示例	uint16 MCU_Port;
	// 获得本机端口，保存到 MCU_Port 中
	MCU_Port = GetPortSet();

表 2.19 GetServerIpSet 函数

函数原型	uint8 GetServerIpSet(uint8 *ip)
函数功能	获得服务器 IP 设置
入口参数	ip 用来保存服务器 IP 地址的缓冲区
出口参数	TRUE 成功 FALSE 失败

调用示例	uint8 PC_IP [4];
	// 获得本机服务器 IP 地址，并保存到 PC_IP 中
	GetServerIpSet(PC_IP);

表 2.20 GetServerPortSet 函数

函数原型	uint16 GetServerPortSet(void)
函数功能	获得服务器端口设置
入口参数	无
出口参数	服务器端口

调用示例	uint16 PC_Port;
	// 获得服务器端口，保存到 PC_Port 中
	PC_Port = GetServerPortSet();

表 2.21 SaveIpSet 函数

函数原型	uint8 SaveIpSet(uint8 ip0, uint8 ip1, uint8 ip2, uint8 ip3)
函数功能	保存本机 IP 设置
入口参数	ip0~ip3 本机 ip 值
出口参数	TRUE 成功 FALSE 失败

调用示例	// 设置本机 IP 地址为：192.168.0.179
	SaveIpSet(192,168,0,179);

表 2.22 SaveMarkSet 函数

函数原型	uint8 SaveMarkSet(uint8 mark0, uint8 mark1, uint8 makrk2, uint8 mark3)
函数功能	保存本机子网掩码设置
入口参数	mark0~mark3 本机子网掩码
出口参数	TRUE 成功 FASLE 失败

调用示例	// 设置本机子网掩码为：255.255.255.0
	SaveMarkSet(255,255,255,0);

表 2.23 SaveGateWaySet 函数

函数原型	uint8 SaveGateWaySet(uint8 GateWaySet0, uint8 GateWaySet1, uint8 GateWaySet2, uint8 GateWaySet3)
函数功能	保存本机网关设置
入口参数	GateWaySet0~GateWaySet3 本机网关 IP 地址
出口参数	TRUE 成功

	FALSE 失败
调用示例	// 设置本机的网关地址为：192.168.0.1 SaveGateWaySet (192,168,0,1);

表 2.24 SavePortSet 函数

函数原型	uint8 SavePortSet(uint16 Port)
函数功能	保存本机端口设置
入口参数	Port 本机端口号
出口参数	TRUE 成功 FALSE 失败

调用示例	// 设置本机端口号为：1200
	SavePortSet(1200);

表 2.25 SaveServerIpSet 函数

函数原型	uint8 SaveServerIpSet(uint8 ip0, uint8 ip1, uint8 ip2, uint8 ip3)
函数功能	保存服务器 IP 设置
入口参数	ip0~ip3 服务器 ip 值
出口参数	TRUE 成功 FALSE 失败
调用示例	SaveServerIpSet(192,168,0,65); // 设置服务器 IP 地址为：192.168.0.65

表 2.26 SaveServerPortSet 函数

函数原型	uint8 SaveServerPortSet(uint16 Port)
函数功能	保存服务器端口设置
入口参数	Port 服务器端口号
出口参数	TRUE 成功 FALSE 失败
调用示例	SaveServerPortSet(2200); // 设置服务器端口号为：2200

注：该部分函数只存在于功能代码含 N 的 MiniARM 嵌入式工控模块中。

2.2.2.5 I²C 相关操作

MiniARM 嵌入式工控模块内部含有一个 I²C 接口，可以直接用来操作外部 I²C 从器件。模板将有关 I²C 的操作全部封装起来，使用时，用户只需要调用相关的接口函数即可。

表 2.27 I2CGetFlag 函数

函数原型	uint8 I2CGetFlag(void)
函数功能	获取 I²C 总线状态
入口参数	无
出口参数	I2C_WRITE_END 写完成 I2C_READ_END 读完成 I2C_NOT_GET_BUS 丢失仲裁 I2C_ACK_ERR 接收 ACK 错误 I2C_FINISH 操作完成
调用示例	Flag = I2CGetFlag( ); // 获取 I²C 总线状态

表 2.28 I2CWrite 函数

函数原型	uint16 I2CWrite(uint8 Addr, uint8 *Data, int16 NByte)
函数功能	向 I^2C 从器件写数据
入口参数	Addr 从机地址 Data 指向将要写的数据的指针 NByte 写的数据数目
出口参数	无意义
说明	向 I^2C 从器件写数据时，需要与 I2CGetFlag 函数配合完成。

表 2.29 I2CRead 函数

函数原型	int16 I2CRead(uint8 Addr , uint8 *Ret , uint8 *Eaddr , int16 EaddrNByte , int16 ReadNbyte)
函数功能	从 I^2C 从器件读数据
入口参数	Addr 从机地址 Ret 指向返回数据存储位置的指针 Eaddr 扩展地址存储位置 EaddrNByte 扩展地址字节数，0 为无 ReadNbyte 将要读取的字节数目
出口参数	无意义
说明	从 I^2C 从器件读数据时，需要与 I2CGetFlag 函数配合完成。

表 2.30 I2CGetWriteBytes 函数

函数原型	uint16 I2CGetWriteBytes(void)
函数功能	获得未写完的数据个数
入口参数	无
出口参数	未写完的数据个数
说明	该函数在实际使用过程中只对最后一次操作有效。

表 2.31 I2CGetReadBytes 函数

函数原型	int16 I2CGetReadBytes(void)
函数功能	获得未读完的数据个数
入口参数	无
出口参数	未读完的数据个数
调用示例	该函数在实际使用过程中只对最后一次操作有效。

注：工程模板中，已经初始化了 I^2C 接口，I^2C 总线速率为 100KHz，用户在使用过程中不可再重新初始化 I^2C 接口，此处也未介绍 I^2C 初始化函数。

向 I^2C 从器件写入数据时，只有一个函数 I2CWrite(uint8 Addr, uint8 *Data, int16 NByte)。对于无子地址的器件来说，该函数直接应用即可。使用该函数操作有子地址的器件时，我们将器件的子地址当作普通的数据来处理，子地址要最先发送，即，发送数据缓冲区的开始应该为器件的子地址，如程序清单 2.6、程序清单 2.7 所示。

程序清单 2.6 向单字节子地址器件中写入数据

```
#define          CAT24C02      0xA0                    // CAT24C02 的器件地址
……
for(i = 0;i < 10;i++) {
    DataBuf[i + 1] = i + '0';
}
DataBuf[0] = 0;                                        // 子地址为 0
// 向 24C02 中写入 10 个字节的数据，保存在地址 0x00～0x09 处
I2CWrite(CAT24C02, DataBuf, 11);
for(i=0; i<OS_TICKS_PER_SEC; i++) {                    // 等待写操作完成，超时时间为 1S
    temp = I2CGetFlag();
    if (temp == I2C_WRITE_END || temp == I2C_FINISH) {// 等待写操作完成
        break;
    }
    OSTimeDly(1);
}
……
```

程序清单 2.7 向双字节子地址器件中写入数据

```
#define     Addr24C128     0xA0                        // 24C128 的地址为 A0H
……
for(i = 0;i < 10;i++) {
    DataBuf[i + 2] = i + '0';
}
DataBuf[0] = 0x00;                                     // 子地址为 0x0040
DataBuf[1] = 0x40;
// 向 24C128 中写入 10 个字节的数据，保存在地址 0x0040～0x0049 处
I2CWrite(Addr24C128, DataBuf, 12);
for(i=0; i<OS_TICKS_PER_SEC; i++) {                    // 等待写操作完成，超时时间为 1S
    temp = I2CGetFlag();
    if (temp == I2C_WRITE_END || temp == I2C_FINISH) {// 等待写操作完成
        break;
    }
    OSTimeDly(1);
}
……
```

注：在操作 24C128 时，先发送子地址的高字节，后发送子地址的低字节。

当从 I^2C 器件读取数据时，需要单独提供从器件的子地址（即，扩展地址）。例，从器件的器件地址为 0xA0，器件子地址是单字节形式，将器件内部地址 0x90 开始的 12 个字节数据读取出来，如程序清单 2.8 所示。

程序清单 2.8 读取从器件内部的数据

```
……
Data[0] = 0x90;
```

```
I2CRead(0xA0, Data, Data, 1, 12);                    // 读取器件内的数据信息
while (1) {                                          // 等待读操作完成
    Temp = I2CGetFlag();
    if((Temp == I2C_READ_END) || (Temp == I2C_FINISH)) {
        break;
    }
    if((Temp == I2C_NOT_GET_BUS) || (Temp == I2C_ACK_ERR)) {
                                                     // 如果总线出现错误，重新读取数据
        Data[0] = 0x90;
        I2CRead(0xA0, Data, Data, 1, 12);            // 重新读取器件内的数据信息
    }
}
……
```

函数 I2CGetWriteBytes 和 I2CGetReadBytes 没有什么实际意义。在 MiniARM 系列工程模板中，允许用户在多个任务中操作 I^2C 总线，而函数 I2CGetWriteBytes 和 I2CGetReadBytes 只对最后一次的操作有效。如果系统中只有一个任务操作 I^2C 总线，则函数 I2CGetWriteBytes 和 I2CGetReadBytes 返回的值是有意义的，用户可以通过这两个函数来得知对 I^2C 从器件的操作进度。

注：用户不能访问器件地址 0x80。

2.2.2.6 在应用中编程——IAP

在应用中编程——IAP，顾名思义，就是系统在运行的过程中可以对自身的 Flash 进行再编程。MiniARM 嵌入式工控模块支持 IAP 操作，并提供了 IAP 的底层驱动函数。用户可以利用 IAP 底层函数向 Flash 中写入数据，还可以实现在线升级，在模板中也有专门的升级接口。IAP 底层驱动函数按照功能划分如所示。

表 2.32　IAP 功能函数划分

功能	函数	功能说明
在线升级接口	BeginUpgrade	开始升级
	WordProgram	升级程序区半字(16 位)编程
	EndUpgrade	升级结束
扇区擦除接口	SectorErase	擦除扇区
获取扇区号	GetUpSecIndex	获得升级程序区指定偏移地址所在的扇区号
	GetCurSecIndex	获得当前程序区指定偏移地址所在的扇区号
读取数据	WordUpRead	在升级程序区，读半字(16 位)
	WordCurRead	在当前程序区，读半字(16 位)
半字编程	WordUpProgram	在升级程序区，写入半字(16 位)
	WordCurProgram	在当前程序区，写入半字(16 位)

表 2.33　BeginUpgrade 函数

函数原型	uint8　BeginUpgrade(void)
函数功能	开始升级
入口参数	无

出口参数	0　当前程序执行在 Low 区 1　当前程序执行在 High 区

表 2.34　WordProgram 函数

函数原型	uint8　WordProgram(uint32　Addr, uint16　Data)
函数功能	升级程序区半字(16 位)编程
入口参数	Addr　　所编程的 2 个字节在升级程序区中的相对偏移 Data　　要编成的数据
出口参数	0　失败 1　成功

函数 WordUpProgram(a, b)的功能与 WordProgram(uint32　Addr, uint16　Data)的功能是一样的，参数也是相同的：a 表示在升级程序区中的相对偏移地址（uint32 Addr），b 表示所要编程的数据（uint16 Data）。

表 2.35　EndUpgrade 函数

函数原型	void　EndUpgrade(void)
函数功能	升级结束
入口参数	无
出口参数	无

表 2.36　SectorErase 函数

函数原型	uint8　SectorErase(int32　Index)
函数功能	擦除扇区
入口参数	Index　　相对当前程序区起始地址的扇区索引
出口参数	0　失败 1　成功

表 2.37　WordCurProgram 函数

函数原型	uint8　WordCurProgram(uint32　Addr, uint16　Data)
函数功能	当前程序区半字(16 位)编程
入口参数	Addr　　所编程的 2 个字节在当前程序区中的相对偏移地址 Data　　要编成的数据
出口参数	0　失败 1　成功

表 2.38　WordUpRead 函数

函数原型	uint32　WordUpRead(uint32　Offset)
函数功能	升级程序区，读半字(16 位)
入口参数	Addr　　所读取的 2 个字节在升级程序区中的相对偏移地址
出口参数	读到的结果

表 2.39 WordCurRead 函数

函数原型	uint32 WordCurRead(uint32 Offset)
函数功能	当前程序区，读半字(16 位)
入口参数	Addr 所读取的 2 个字节在当前程序区中的相对偏移
出口参数	读到的结果

表 2.40 GetUpSecIndex 函数

函数原型	int32 GetUpSecIndex(int32 Offset)
函数功能	获得升级程序区指定偏移地址所在的扇区号
入口参数	Offset 相对升级程序区起始地址的偏移地址
出口参数	相对当前程序区的扇区偏移

表 2.41 GetCurSecIndex 函数

函数原型	int32 GetCurSecIndex(int32 Offset)
函数功能	获得当前程序区指定偏移地址所在的扇区号
入口参数	Offset 相对当前程序区起始地址的偏移地址
出口参数	相对当前程序的扇区偏移

核心板中提供的 IAP 功能主要有两个方面的应用：数据存储和在线升级。

当执行在线升级时，MiniARM 产品有 2 个用户程序区，分别称为 HIGH 程序区和 LOW 程序区，如图 2.20 所示。如果用户程序运行于 HIGH 程序区，那么升级将对 LOW 程序区操作；同理，如果用户程序运行于 LOW 程序区，那么升级将对 HIGH 程序区操作。

为了方便用户编译，模板编译模式有 3 个选项，分别是 DebugInLowAddr、DebugInHighAddr、recovery。其中，DebugInLowAddr 选项用于生成存储在低位的用户程序，DebugInHighAddr 选项用于生成存储在高位的用户程序。

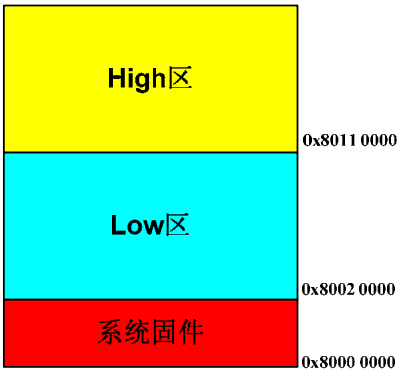

图 2.20 MiniARM 嵌入式工控模块执行在线升级时用户程序空间分配

升级程序区半字(16 位)编程函数 WordProgram(uint32 Addr, uint16 Data)中，Addr 是一个地址相对偏移量。如果对 Low 区间进行编程，则基准地址为：0x8002 0000；如果对 High 区间进行编程，则基准地址为：0x8011 0000。

MiniARM 嵌入式工控模块在线升级方案如图 2.21 所示。固件中仅仅提供了一种在底层操作 Flash 的方法，在线升级要求数据要绝对准确，所以，用户需要在应用层使用一些协议来确保数据的准确性，只有确保数据的可靠，才能够进行升级，否则会出现意想不到的后果。

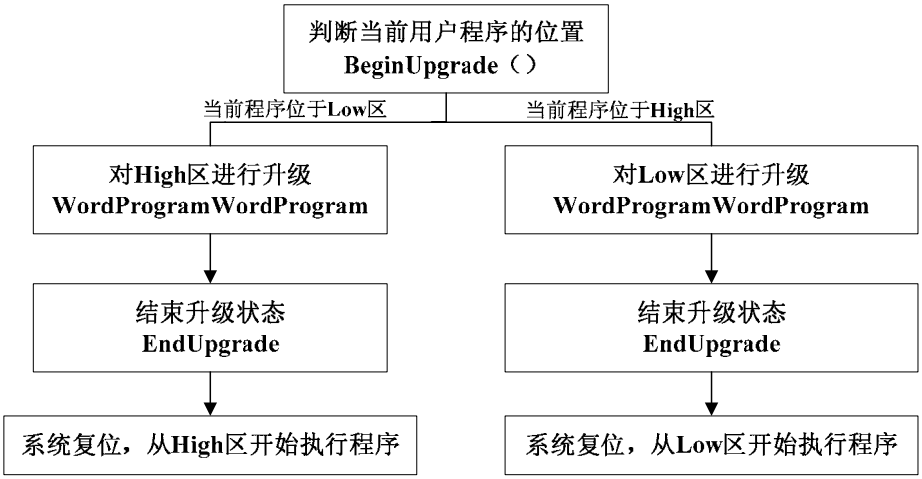

图 2.21　在线升级方案

当作为数据存储功能时，首先要找到目标地址所在的扇区，然后调用扇区擦除函数，擦除该扇区，最后调用半字编程函数，向该扇区写入指定数据。

注：请慎用程序存储器作为数据存储功能使用，因为不正确的操作将会破坏工控模块固件。

2.2.2.7 其它

在系统函数中除了上述一些函数外，还提供了一些特殊函数，用来完成一些特定的操作。

表 2.42　CheckVersion 函数

函数原型	uint8 GetCodeVer ()
函数功能	检查固件版本与工程模板是否兼容
入口参数	无
出口参数	固件版本号

注：仅适用 M22A-N20 系列 MiniARM

表 2.43　CheckVersion 函数

函数原型	uint8 CheckVersion(uint32 Ver, const char *Name)
函数功能	检查固件版本与工程模板是否兼容
入口参数	Ver　：固件版本， Name ：产品型号，
出口参数	TRUE ：固件版本号、产品型号完全匹配 FLASE：固件版本号、产品型号不匹配但兼容 不返回：固件版本号或产品型号不兼容，函数不返回,停止运行

表 2.44　SetDataAbortISR 函数

函数原型	void　SetDataAbortISR(uint32　addr)
函数功能	设置数据中止异常服务地址
入口参数	addr　　数据中止异常服务地址
出口参数	无
说明	数据中止异常服务程序需要使用 ARM 汇编语言编写

表 2.45 SetPrefetchISR 函数

函数原型	void SetPrefetchISR(uint32 addr)
函数功能	设置指令中止异常服务地址
入口参数	addr 指令中止异常服务地址
出口参数	无
说明	指令中止异常服务程序需要使用 ARM 汇编语言编写

表 2.46 SetUndefinedSR 函数

函数原型	void SetUndefinedSR(uint32 addr)
函数功能	设置未定义指令异常服务地址
入口参数	addr 未定义指令异常服务地址
出口参数	无
说明	未定义指令异常服务程序需要使用 ARM 汇编语言编写

表 2.47 OSGetPrio 函数

函数原型	uint8 OSGetPrio(void)
函数功能	获取当前任务优先级
入口参数	无
出口参数	返回当前任务的优先级

注：指令中止异常、数据中止异常和未定义指令中止异常的服务程序需要使用汇编语言来编写，用户可以参考 FIQ 服务程序的编写方法。

3. 操作说明

上文介绍了用户工程模板，下面演示一下该模板的使用方法。核心板底板使用 M22A 评估板 M22 Series EV Board，程序控制评估板上的蜂鸣器鸣叫。

步骤 1：添加专用工程模板（以 M9020-FNU20I 为例，如果已经添加，则该步骤可以省略）

将工程模板所在文件夹 M9020-FNU20_uCOS_Lib_Rxxxxxx 拷贝到 ADS 安装目录下的 Stationery 文件夹内，如图 3.1 所示（Rxxxxxx 表示发布日期）。

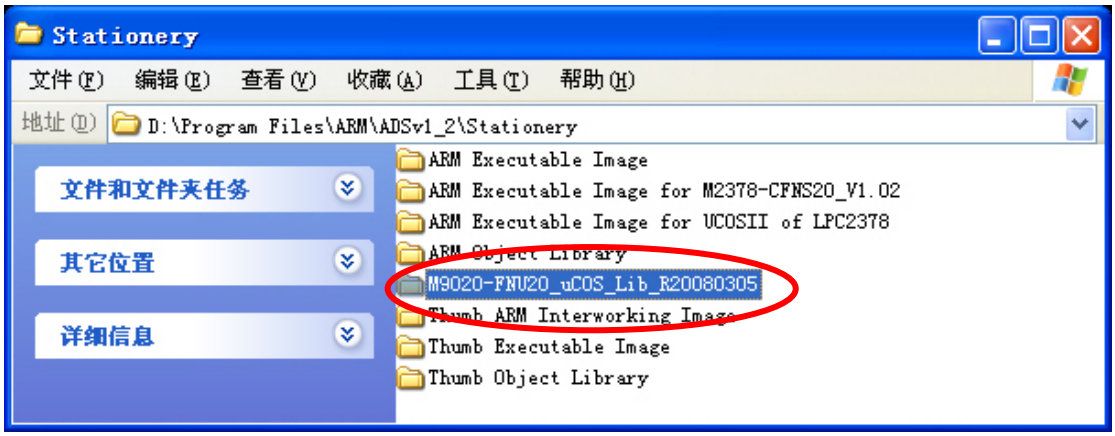

图 3.1 添加专用工程模板

步骤 2：建立工程

打开 ADS（ARM_Developer Suite v1.2->CodeWarrior for ARM Developer Suite）开发环境，使用模板 M9020-FNU20_uCOS_Rxxxxxx 建立工程——BEEP_CON，如图 3.2 所示。

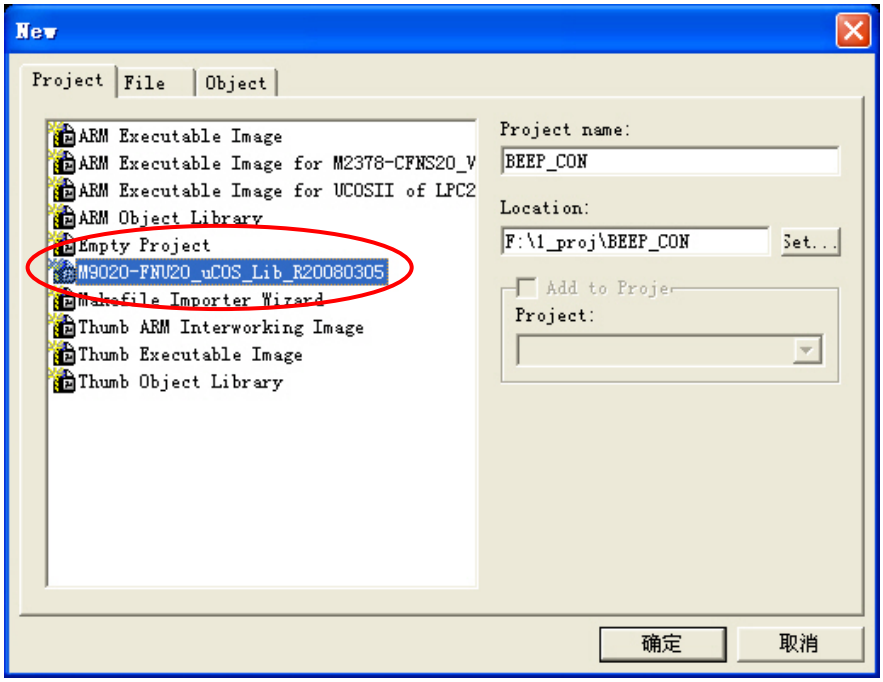

图 3.2 使用 M9020-FNU20 专用工程模板建立工程——BEEP_CON

步骤 3：编写程序

为了方便用户编程，模板中主文件为用户编写了 6 个任务：Task0～Task5，用户只需要向其中填写用户代码即可，Task5 用于控制一个 LED 闪烁。任务 Task0～Task5 占用的优先级为：17～22。每个任务的堆栈大小都为 512 个字（OS_STK），用户可以根据自己的实际使用来调整堆栈的大小，如程序清单 3.1 所示。

程序清单 3.1 任务函数规划

```
#define TASK0_PRIO              17              // 任务的优先级
#define TASK0_ID                TASK0_PRIO      // 任务的 ID
#define TASK0_STACK_SIZE        512             // 定义用户堆栈长度

#define TASK1_PRIO              18              // 任务的优先级
#define TASK1_ID                TASK1_PRIO      // 任务的 ID
#define TASK1_STACK_SIZE        512             // 定义用户堆栈长度

#define TASK2_PRIO              19              // 任务的优先级
#define TASK2_ID                TASK2_PRIO      // 任务的 ID
#define TASK2_STACK_SIZE        512             // 定义用户堆栈长度

#define TASK3_PRIO              20              // 任务的优先级
#define TASK3_ID                TASK3_PRIO      // 任务的 ID
#define TASK3_STACK_SIZE        512             // 定义用户堆栈长度

#define TASK4_PRIO              21              // 任务的优先级
#define TASK4_ID                TASK4_PRIO      // 任务的 ID
#define TASK4_STACK_SIZE        512             // 定义用户堆栈长度

#define TASK5_PRIO              22              // 任务的优先级
#define TASK5_ID                TASK5_PRIO      // 任务的 ID
#define TASK5_STACK_SIZE        512             // 定义用户堆栈长度
```

注：如果用户需要的任务数多余 6 个，那么用户也可以在 main 函数中再设置一些任务，但是要注意，用户禁止使用 0~15 之中的优先级。

我们在工程模板的任务 Task1 中编写应用程序来控制蜂鸣器的鸣叫，代码如程序清单 3.2 所示，蜂鸣器每隔 1S 鸣叫一次。有关蜂鸣器控制引脚的初始化工作都在 main 函数中完成了，所以，此处不必再对蜂鸣器重新初始化了。

程序清单 3.2 任务 Task1

```
/*********************************************************************
** 任务名称:    TASK1
** 功能描述:    无
** 入口参数:    无
** 出口参数:    无
*********************************************************************/
```

```
void   TASK1(void   *pdata)
{
      pdata = pdata;
      while (1) {
            GpioSet(BUZZER);                          // 蜂鸣器鸣叫
            OSTimeDly(OS_TICKS_PER_SEC/10);
            GpioClr(BUZZER);                          // 蜂鸣器停止鸣叫
            OSTimeDly(OS_TICKS_PER_SEC);
      }
}
```

步骤 4：编译、调试

使用 DebugInLowAddr 选项编译工程，如果有错误，则修改错误，直到编译通过为止，如图 3.3 所示，然后便可以进行工程调试。

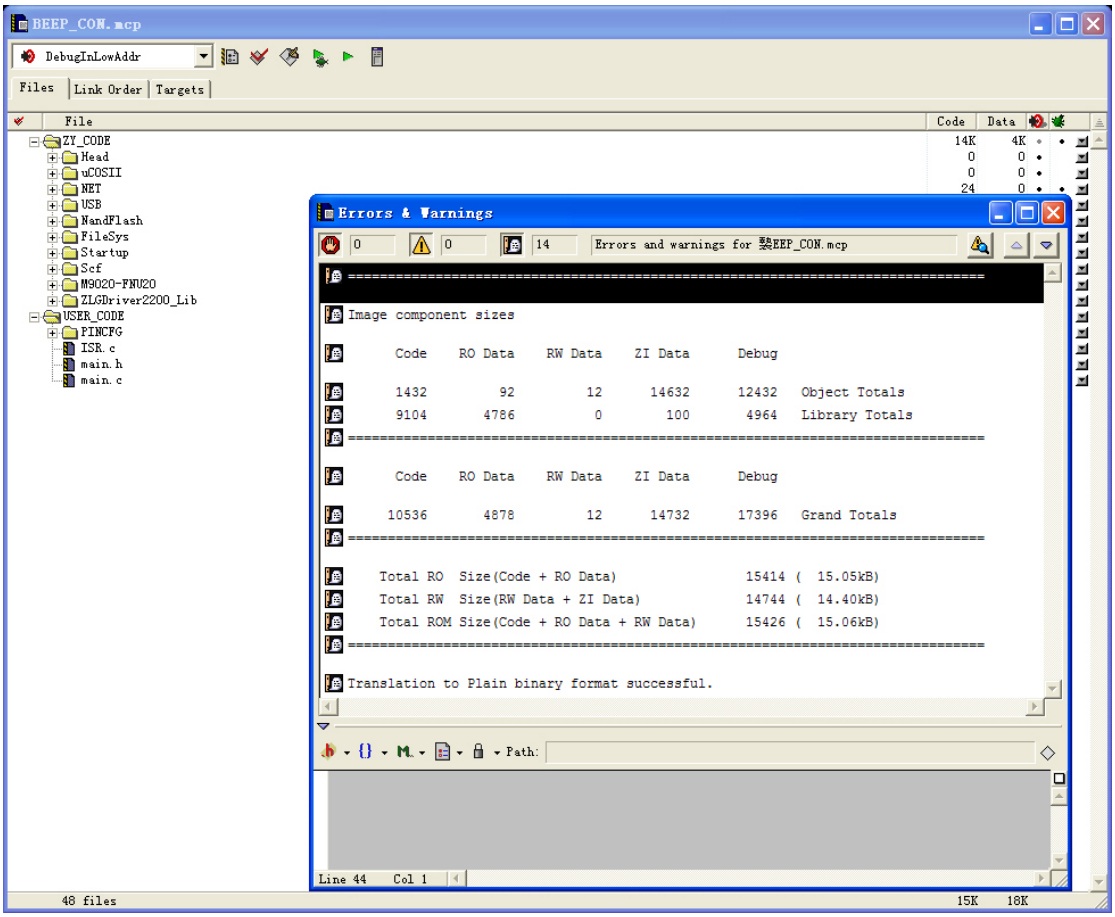

图 3.3 工程编译 OK

4. 故障及其解决

下面列举一些比较常见的故障情况，更详细的情况请拨打 MiniARM 技术支持电话。

故障 1、使用 ADS1.2 不能下载用户程序，提示“RDI Severe Error 00500:Could not find h-jtag server. Please make sure that h-jtag server has been started properly”。

解决：1. 查看 MiniARM 嵌入式工控模块是否正常供电；

2. 查看 PC 机 H-JTAG Server 是否启动，是否能检测 MCU;

3. 确保 H-Flasher 正确配置，查看能否检测 Flash（工业级产品 Flash Selection 项芯片选择 36VF1601C、商业级产品该项选择 39VF1601，光盘配置文件默认设置为工业级 36VF1601C）；

故障 2、程序下载后不能运行到 main 函数。

解决：使用与该产品型号对应的工程模板恢复固件，编译目标栏选择 recovery，而非 DebugInLowAddr（MiniARM 嵌入式工控模块用户不能使用 H-Flasher 整片擦除功能）。

故障 3、CF 卡不能操作。

解决：1. 确保 CF 卡未损坏；

2. CF 卡不能带电插拔，确保在断电情况下 CF 卡正确插入卡槽；

3. 查看 M22A EV Board JP6、JP12 是否短接；

4. 查看程序代码是否成功加载了 CF 卡驱动且初始化了文件系统；

5. 查看程序代码访问的 CF 卡盘符是否正确（4、5 点同样适用于 U 盘和电子盘）。

故障 4、工控模块不能与 PC 机 TCP/UDP 调试软件通信。

解决：1. 确保使用的工控模块含网络功能（功能码含 N）；

2. 查看工控模块端 IP 相关设置信息与 TCP/UDP 调试软件的相关信息是否对应。

5. 免责声明

开发预备知识

MiniARM® M22A 系列产品将提供尽可能全面的开发模板、驱动程序及其应用说明文档以方便用户使用，但 MiniARM® M22A 系列产品不是教学开发平台。对于需要熟悉 ARM7 体系结构，LPC2200 系列微控制器特性及其 ADS 开发环境的用户，建议同时购买我公司 SmartARM2200 或 EasyARM2200 教学开发平台。

LPC2000 系列微控制器

建议用户在 NXP 半导体主页（http://www.nxp.com）上获取最新勘误表并仔细阅读。广州致远电子有限公司对 LPC2200 系列微控制器无论是已知的还是潜在的设计缺陷不负任何责任。

修改文档的权利

广州致远电子有限公司保留任何时候在不事先声明的情况下对 MiniARM® M22A 系列产品相关文档的修改的权力。

ESD 静电放电保护

MiniARM® M22A 系列产品部分元器件内置 ESD 保护电路，但依然建议用户在设计底板时提供 ESD 保护措施，特别是电源与 I/O 设计，以保证产品的稳定运行。安装 MiniARM® M22A 系列产品时，请先将积累在身体上的静电释放，例如佩戴可靠接地的静电环，触摸接入大地的自来水管等。

公　　司：广州致远电子有限公司 嵌入式系统事业部
地　　址：广州市天河区车陂路黄洲工业区二栋四楼（研发部）
邮　　编：510660
网　　址：www.embedtools.com
销售电话：+86 (020) 2264-4249
技术支持：+86 (020) 2887-2684
传　　真：+86 (020) 3860-1859