ACT-416RL 系列用户手册

版本号：V2.0

瑞强科技

版本修订

版本号	修订日期	描述	审核
V1.0	20160907	创建文档	
V1.1	20171218	修改文档	
V2.0	20180103	修改文档	

特别说明

本公司保留在未通知用户的情况下，对产品、文档、服务等内容进行修改、更正等其他一切变更权利。

目录

一、产品概述 4
二、产品特性 4
1. 硬件特性 4
2. 软件特性 6
2.1 系统特性 6
2.2 环境配置 6
2.3 管理机登录 6
三、接口定义 7
1. 电源接口 7
2. RS485 接口 7
3. RS232、CAN 接口 8
4. 开关量输出输入 9
5. 网络接口 9
6. 调试接口 10
7. RS485/RS232 驱动接口 11
四、驱动实例 11

一、产品概述

ACT-416RL 是一款基于精简指令集（RISC）架构高性能的 32 位 MPU 的嵌入式计算机。该 CPU 是以 ARM Cortex-A8 为核心的系统级单芯片，内置 NEON 单指令流多数流（SIMD）协处理，带有错误校正码（ECC）的 256KB L2 缓存，最高支持 1GHz 的频率。系统提供 RS458/RS232 通讯，有线网络通讯，CAN 总线，同时也提供可选的无线 GPRS 通讯，具有体积小、功耗低、效率高等特点，适用于电力集中器、HMI、工业控制、网关等场合。

二、产品特性

1. 硬件特性

- AM355x CPU：
  - 32bit ARM Corte-A8 架构，主频 800MHz，1.6MIPS/MHz，最高主频 1GHz
  - 32KB I-cache，32KB D-cache，NeonSIMD 协处理器
- 内存：
  - 512Mbyte DDR3、64KB 专用 RAM
- FLASH：
  - 256Mbyte NANDFlash，最大支持 8Gbyte
  - 支持 NAND、NOR、SRAM 等 FLASH
- 加密：
  - 支持 PRNG/DES/3DES/AES/SHA/HMAC 加密，最高 256 位加密模式
- 看门狗：
  - 内置 WDT，溢出时间小于 60 秒，支持空闲唤醒和掉电唤醒
- RTC:
  - 高精度实时时钟，内置供电电池
- 调试口：
  - 1 路串口为系统 console 口。波特率：115200，数据位：8，停止位：1，

校验位：none，流控：无

- RS485/RS232：
  - 8 路独立 RS485 通讯，内部全隔离保护设计
  - 8 路 RS485/RS232 分时复用通讯，可根据实际选择使用，内部全隔离保护设计
- B 码对时：
  - 1 路 RS485 接收，专用于 B 码对时，连接后自动对时
- CAN：
  - 1 路 CAN 通讯，内置隔离保护设计
- 开关量输出输入：
  - 2 路双刀双掷继电器输出
  - 2 路开关量输入
- 网络：
  - 4 路 10M/100M 自适应工业以太网，标准 RJ45 接口
  - 15KV TVS 保护，内部全隔离保护设计
- 无线功能（可选）：
  - 射频波段 800/900/1800/1900MHz（可选 2/3/4G）
  - 可选 WIFI：可连接 AP，也可做 AP
  - 1 个 SIM 卡接口，1 个天线接口
  - 传输速度：达到相应功能的标准速度
- SD CARD：
  - 内置一个 SD/MMC 卡接口
- 电源：
  - 输入电压：220V，支持交流、直流
  - 单机功耗：< 12W
- 机械特性
  - 外壳金属材质
  - 尺寸：1U
  - 防护等级：IP63

- 工作环境
  - 工作温度：-40℃~+85℃
  - 工作湿度：5% ~ 95%

2. 软件特性

2.1 系统特性

ACT-416RL 预装基于 TI AM335x 的 Linux 操作系统，版本为 3.2.0。满足 POSIX 标准或类 UNIX 平台的应用程序。针对系统特有的硬件设备，内核提供了简单、易用的驱动接口，可加速用户的应用程序开发。

ACT-416RL 系统的软件系统共分为 3 部分，分别为 Bootloader、linux 内核和 rootfs。Bootloader 是遵循 GPL 条款的开放源码项目，UBoot 主要是引导内核的启动，支持 NFS 挂载、NAND Flash 启动；linux 内核是整个操作系统的最底层，负责整个硬件的驱动，以及提供各种系统所需的核心功能；rootfs 是用于明确磁盘或分区上的文件的方法和数据结构，即在磁盘上组织文件的方法。

2.2 环境配置

本公司提供的虚拟机系统 ubuntu 10.04，可直接编译使用。

用户名：work

密码 ：123456

编译命令：arm-linux-gnueabihf-gcc -o filename filename.c

编译链：本公司提供的 arm-linux-gnueabihf-4.7.tar.gz

非本公司提供的编译环境下，把编译链拷贝到 PC 的 LINUX 系统下，解压编译链后，把根目录下的 bin 目录添加到系统的环境变量即可。

如解压到/opt/arm-linux-gnu 目录下，则添加环境变量为：

export PATH=$PATH:/ opt/arm-linux-gnu /bin

2.3 管理机登录

IP： eth0：192.168.1.177

eth1：192.168.2.177

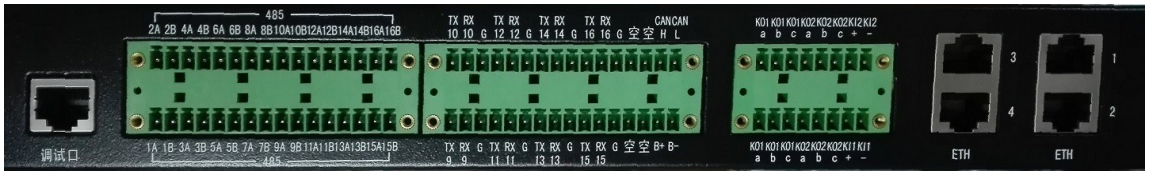
eth2：192.168.3.177

eth3：192.168.4.177

用户名：root

密码 ：root

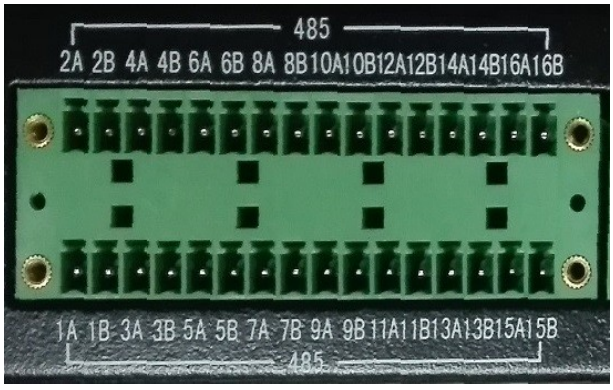
三、接口定义



1. 电源接口

编号	标识符	功能说明
1	AC/DC220V	电源接口，支持交流/直流
2		保护地（非零线 N）

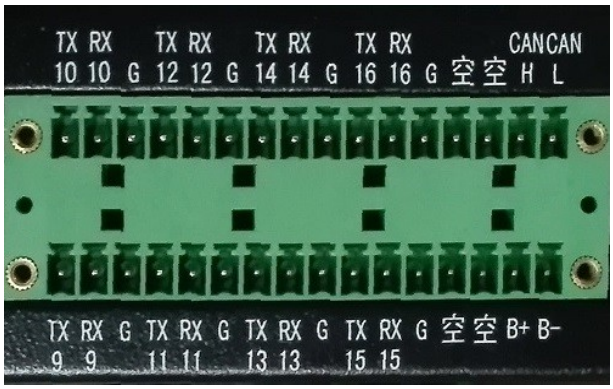
2. RS485 接口



编号	标识符	功能说明
1	nA	第 n 通道 RS485 端口 A（n=1~16）
2	nB	第 n 通道 RS485 端口 B（n=1~16）

注：RS485 第 9~16 通道与 RS232 的第 9~16 通道为复用通道。

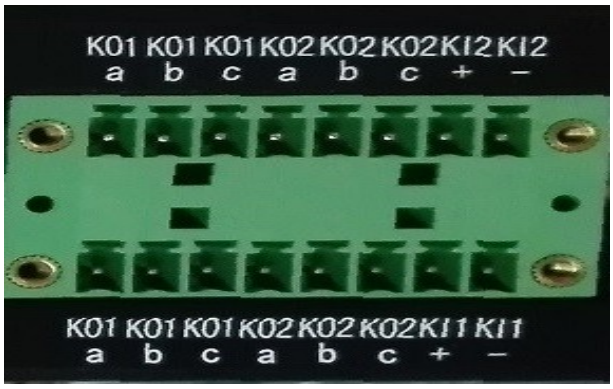
3. RS232、CAN 接口



编号	标识符	功能说明
1	TXn	第 n 通道 RS232 端口 TX（n=9~16）
2	RXn	第 n 通道 RS232 端口 RX（n=9~16）
3	G	GND，通讯地
4	空	NC，无连接
5	CAN:H	CAN 通讯 H 端
6	CAN:L	CAN 通讯 L 端
7	B+	B 码对时 RS485-A 端
8	B-	B 码对时 RS485-B 端

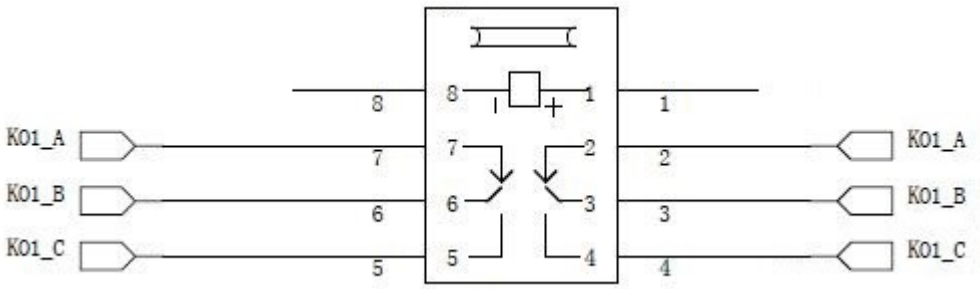
注：RS232 第 9~16 通道与 RS485 的第 9~16 通道为复用通道。对应的驱动接口相同，为分时复用通道。

4. 开关量输出输入

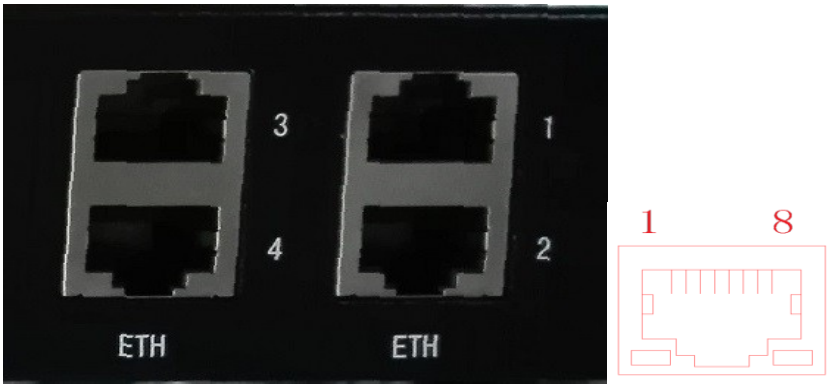


编号	标识符	功能说明
1	KOn:a	第 n 路继电器输出常闭端（n=1,2）
2	KOn:b	第 n 路继电器输出公共端（n=1,2）
3	KOn:c	第 n 路继电器输出常开端（n=1,2）
4	KIn:+	第 n 路开关量输入正端（n=1,2）
5	KIn:-	第 n 路开关量输入负端（n=1,2）

注：继电器输出为双刀双掷开关，如下图：

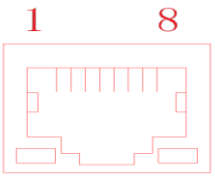


5. 网络接口



网口编号	编号	标识符	功能说明
ETH1/2/3/4	1	E0/1_TX+	以太网 ETH0/1_TX+
	2	E0/1_TX-	以太网 ETH0/1_TX-
	3	E0/1_RX+	以太网 ETH0/1_RX+
	4	NC	未使用
	5	NC	未使用
	6	E0/1_RX-	以太网 ETH0/1_RX-
	7	NC	未使用
	8	NC	未使用
IP	1	Eth1	192.168.1.177
	2	Eth2	192.168.2.177
	3	Eth3	192.168.3.177
	4	Eth4	192.168.4.177

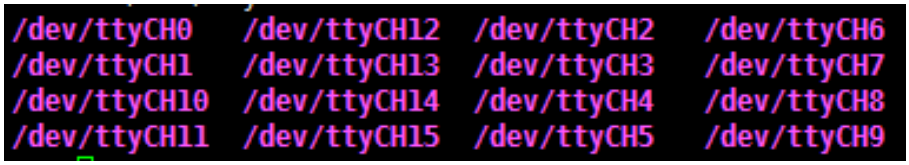
6. 调试接口

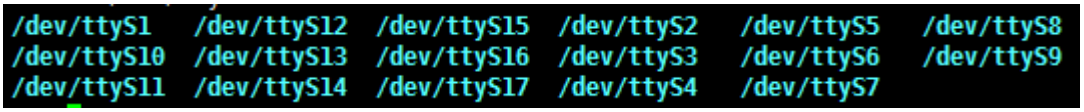


编号	标识符	功能说明
1	TX	RS232 调试串口 TX
2	RX	RS232 调试串口 RX
3	GND	系统通讯地
4-8	NC	未使用

调试口配置：波特率：115200，数据位：8，停止位：1，校验位：none，流控：无

7. RS485/RS232 驱动接口





驱动接口可以在管理机的/dev 目录下查看。

编号	标识符	功能说明
1	ttySn	对应 ttyCH(n-1)，第 n 通道 RS485 驱动接口（n=1~8）
2	ttySn	对应 ttyCH(n-1)，第 n 通道 RS485/RS232 驱动接口（n=9~16）
3	TtyS17	GPRS 通讯串口（2G 模块才会使用）

四、驱动实例

在系统的/program 目录下有相应的脚本文件，可以进行一些简单的测试。其中要确保 startup.sh 文件里，端口映射的正确的。文件内容见附录。

附录：

1、 startup.sh 文件内容：

```
#!/bin/sh
ln -sf /dev/ttyCH0 /dev/ttyS1
ln -sf /dev/ttyCH1 /dev/ttyS2
ln -sf /dev/ttyCH2 /dev/ttyS3
ln -sf /dev/ttyCH3 /dev/ttyS4
ln -sf /dev/ttyCH4 /dev/ttyS5
ln -sf /dev/ttyCH5 /dev/ttyS6
```

```
ln -sf /dev/ttyCH6 /dev/ttyS7
ln -sf /dev/ttyCH7 /dev/ttyS8
ln -sf /dev/ttyCH8 /dev/ttyS9
ln -sf /dev/ttyCH9 /dev/ttyS10
ln -sf /dev/ttyCH10 /dev/ttyS11
ln -sf /dev/ttyCH11 /dev/ttyS12
ln -sf /dev/ttyCH12 /dev/ttyS13
ln -sf /dev/ttyCH13 /dev/ttyS14
ln -sf /dev/ttyCH14 /dev/ttyS15
ln -sf /dev/ttyCH15 /dev/ttyS16
ln -sf /dev/ttyO1 /dev/ttyS17
ip link set can0 type can bitrate 100000
ifconfig can0 up
```

2、 serial.c 文件内容：

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#define max_buffer_size     100     /* buffer size */
/*****************************************/
int fd1;
int flag_close;

int open_serial(int k,int *fd)
{
```

```
    int sfd = -1;
    char str[100];
    sprintf(str,"/dev/ttyS%d",k);
    printf("open %s\n",str);

     sfd = open(str,O_RDWR|O_NOCTTY| O_NONBLOCK);
     if(sfd == -1){
          perror(str);
          return -1;
    }
    else{
          *fd = sfd;
          return 0;
    }
}
/*****************************************************************/
int main(int argc, char *argv[ ] )
{
    time_t tNow,tOld;
    int port;
    char
sbuf[]={"12345678901234567890123456789012345678901234567890\n"};/*
固定发送的数据 */
    char sbufrec[256]={0};
    int sfd,retv,i,ncount=0,mcount = 0;
    struct termios opt;
    int length=sizeof(sbuf);
    /*****************************************************************/
    if(argc < 2)
    {
```

```
        printf("input erro :serial <1~4> \n");
        return 0;
    }
    port = atoi(argv[1]);
    open_serial(port,&fd1);
    /*****************************************************************/
    printf("ready for sending data...\n");
    tcgetattr(fd1,&opt);
    cfmakeraw(&opt);
    /*****************************************************************/
    cfsetispeed(&opt,B9600); /*设置波特率为 9600bps*/
    cfsetospeed(&opt,B9600);
    /*****************************************************************/
    tcsetattr(fd1,TCSANOW,&opt);

    while(mcount < 5)
    {
        retv=write(fd1,sbuf,length);   /* 发送数据 */
        if(retv==-1){
            //perror("write");
            printf("write error .....\n");
        }
        else{
            printf("the number of char sent is %d\n",retv);
        }
        ncount=0;
        printf("ready for receiving data...\n");

        time(&tOld);
        tNow=tOld;
```

```
        ncount = 0;
        while(((tNow-tOld) < 2))          /*  设置接收超时  */
        {
            time(&tNow);
            retv=read(fd2,&sbufrec[0],1);
            if(retv==-1){
                //perror("read");
                //printf("error read \n");
                //printf("tOld=%d;tNow=%d\n",tOld,tNow);
            }
            else{
                printf("%02x ",sbufrec[0]);
                ncount+=1;
            }
        }
        mcount+=1;
        printf("\n");
    }
    flag_close = close(fd1);
    if(flag_close == -1)    /*关闭口端口*/
        printf("Close the Device1 failur!\n");
    return 0;
}
```

3、 io_out.c 文件内容：

```
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
#include <stdint.h>
```

```
#define MAXFILENAME_LEN 100
const char *gpio_dir = "/sys/class/gpio/";

int gpio[2] = {
    115,    /* OUT1 */
    114,        /* OUT2 */
};

int write_sysfs_int(int port, int val)
{
    int ret;
    FILE *sysfsfp;
    char temp[MAXFILENAME_LEN] = "\0";
    sprintf(temp, "%sgpio%d/value",gpio_dir,port);
    sysfsfp = fopen(temp, "w");
    if (sysfsfp == NULL) {
        printf("failed to open %s\n", temp);
        return -1;
    }
    fprintf(sysfsfp, "%d", val);
    fclose(sysfsfp);
    return 0;
}

int main(int argc, char *argv[ ])
{
    int port,val;
    if(argc<3)
    {
```

```
        printf("input erro :\n Useage io_out (port no) (value)\n");
        return 0;
    }
    port = atoi(argv[1]);
    val = atoi(argv[2]);
    if((port<=0) || (port>2)){
        printf("The Port no Only 1-2 is valid\n");
        return -1;
    }
    port--;
    return write_sysfs_int(gpio[port], val);
    return 0;
}
```

4、 io_in.c 文件内容：

```
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
#include <stdint.h>

#define MAXFILENAME_LEN 100
const char *gpio_dir = "/sys/class/gpio/";

int gpio[2] = {
    117,   /* IN1 */
    20, /* IN2 */
};

int read_sysfs_int(int port, int *val)
```

```
{
    int ret;
    FILE *sysfsfp;
    char temp[MAXFILENAME_LEN] = "\0";
    sprintf(temp, "%sgpio%d/value",gpio_dir,port);
    sysfsfp = fopen(temp, "r");
    if (sysfsfp == NULL) {
        printf("failed to open %s\n", temp);
        return -1;
    }
    fscanf(sysfsfp, "%d", val);
    fclose(sysfsfp);
    return 0;
}

int main(int argc, char *argv[ ])
{
    int port,val;
    if(argc<2)
    {
        printf("input erro :\n Useage io_out (port no)\n");
        return 0;
    }
    port = atoi(argv[1]);
    if((port<=0) || (port>2)){
        printf("The Port no Only 1-2 is valid\n");
        return -1;
    }
    port--;
    if(read_sysfs_int(gpio[port], &val))
```

```
        return -1;
    else
        printf("val = %d\n",val);
    return 0;
}
```