2018 Synopsys ARC杯电子设计竞赛技术论文

论文题目：**基于embARC的快递自动发货系统**

参赛单位：西安电子科技大学

队伍名称：大愚若智参赛队

指导老师：张弘老师

参赛队员：张婕 李浩然 王开元

完成时间：2018年02月27日

# 基本情况表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 队伍名称 | 大愚若智队 | | | 单位名称 | | 西安电子科技大学 | |
| 项目名称 | 基于embARC的快递自动发货系统 | | | | | | |
| 项目负责人 | 张婕 | | | 联系方式 | | | 13259891707 |
| 指导老师 | 无 | | | 职务 | | | 无 |
| 参赛  队员  信息 | 姓名 | 学历 | 证件号码 | | 专业 | | 分工情况 |
| 张婕 | 研究生 | 34250119950610032X | | 软件工程 | | 编程 |
| 李浩然 | 研究生 | 152531199504290011 | | 软件工程 | | 设计，辅助编程 |
| 王开元 | 研究生 | 412801199502170835 | | 软件工程 | | 设计，编程 |
| 项目时间 | 2018 年 1 月 20 日 - 2018 年 5月28 日 | | | | | | |
| 队伍简介 | 我们小队由张婕，李浩然，王开元三人组成，均是西电微电子学院软件工程专业 | | | | | | |
| 参与项目 | 与Intel联合办学的MCDF验证项目  自动循迹小车项目 | | | | | | |
| 获奖情况  （校级及  以上） | 校级星火杯二等奖 | | | | | | |
| 研究专长 | 基于FPGA的项目设计 | | | | | | |
| 其他 | 无 | | | | | | |

# 摘 要

在日常生活中，我们经常会碰到快递慢，快递在装卸过程中损坏或者快递发送错误的尴尬情况。在我们调查中发现，如今世面上的主流快递企业只有在大型中转分拣中心应用了自动化分拣技术，一般都在企业总部，而在市级快递分拣部门仍采用人工装卸的分拣方式，主要原因是由于自动化设备成本过高以及维修防护困难。由于采取人工分拣方式，效率较慢，错误率较高，而且在装卸过程中损坏风险较大。所以我们小队希望设计一款成本较低，使用简单，效率较高的货物自动分拣系统。

在基于emb开发板的基础上，我们设计的主要思路是：用射频芯片识别技术代替现在的扫码技术，将货物的重量，目的地，发出地，联系方式等信息录入芯片中，这样可以中距离自动扫描信息，并且无论货物摆放方式如何都可以自动将信息输入系统进行信息处理，同时个人信息不易泄漏。在系统中，将货物按照目的地不同进行分类处理，采用pwm驱动电机的方式,连接小车，按不同路径用小车进行输运，全程采取自动化分拣直至装车输运，减少输运过程中损伤的可能，提高效率。主要创新点：采用了射频芯片识别技术和pwm驱动电机。

目前主流自动化分拣技术多采用半人工半机器的方式进行，而且目前没有应用射频芯片设别技术。在运输手段上多采用履带式运输，只有京东中转中心在使用小车进行输运。我们小队希望尽可能提高效率和准确度，同时尽可能的降低成本，应用于小型分拣中心。

关键词：**自动分拣 射频芯片 电机 自动寻径**

# **ABSTRACT**

In everyday life, we often encounter courier slow delivery courier damaged during delivery or delivery to send the wrong embarrassment. In our investigation, we found that mainstream express delivery companies in the world only use automated sorting technology in large transit sorting centers, which are generally located in corporate headquarters. However, manual sorting methods are still adopted in municipal express delivery sorting departments. The main reason is due to the high cost of automation equipment and maintenance difficulties. Due to the manual sorting method, the efficiency is lower, the error rate is higher, and the risk of damage during loading and unloading is relatively large. So our team hopes to design a low cost, simple to use, high efficiency automatic cargo sorting system.

Based on the emb development board, we design the main idea is: RF chip recognition technology instead of the current code scanning technology, the weight of the goods, destination, place, contact information and other information into the chip, so you can Distance automatically scan information, and regardless of how the goods can be placed in the system automatically input information processing, while personal information is not easy to leak. In the system, the goods will be classified according to different destinations, the use of pwm drive motor way to connect the car, according to different paths with the car for transport, the entire process of sorting to the loading and unloading until the transport process to reduce damage The possibility of improving efficiency. The main innovation: the use of radio frequency chip identification technology and pwm drive motor.

At present, the mainstream of automated sorting technology to use semi-artificial and semi-mechanical way, but there is no application of radio frequency chip set technology. More means of transport on the use of crawler transport, only Jingdong transit center in the use of car transport. Our team hopes to maximize efficiency and accuracy while minimizing costs for smaller sorting centers.

**Keywords: Automatic sorting RF chip Motor Automatic pathfinder**

# 目 录

基本情况表 ii

摘 要 iii

**ABSTRACT** iv

目 录 V

第一章 方案论证 1

1.1项目概述 1

1.2资源评估 1

1.3预期结果 1

1.4项目实施评估 1

1.5补充说明 2

第二章 作品难点与创新 3

2.1作品难点分析 3

2.2创新性分析 3

2.3小结 3

第三章 系统结构与硬件实现 4

3.1系统原理分析 4

3.2 系统结构 4

3.3硬件实现 4

3.4 小结 4

# 第一章 方案论证

# 1.1项目概述

本项目--基于embARC的快递自动发货系统的主要意义是设计一个自动化物联网系统，应用于快递分拣行业，以较小的成本提高快递分拣的效率和准确度。因为采用全自动分拣技术，能够大大减少人力耗费，降低损坏率，提高工作效率。

主要采用射频芯片识别技术，分拣之前通过信息录入系统，快递的目的地，来源地，联系方式，重量等主要信息录入芯片，在系统内，进行信息收集和分类，然后按照不同的目的地采用pwm调制驱动四个电机，模拟小车行进，将货物按路径分拣到不同地点，统一装车等待运输。

预期能够实现芯片信息快速识别并进行分类处理后，采用驱动四个电机，模拟小车，采用寻径方式将货物准确运输到不同位置后，小车返回再次运输。

创新点主要采用了射频芯片识别的方式，在快递分拣中属于创新技术，提高分拣速率和准确度。通过pwm调制驱动电机可以自由控制小车在加速，减速，刹车，倒转等多个模式中切换，提高运输效率。

目前团队调研了目前主流快递分拣企业的自动化分拣设备，大多采用半人力半机器的分拣方式，例如百世快递的风暴自动分拣系统，京东快递自动分拣线，多采用履带运输方式，占用面积较大，维护及安装调试复杂，而且只存在于一级分拣机构，在下级分拣公司仍旧采用人工分拣的方式，受制于大型分拣设备成本较高，占地面积大，中小型分拣机构分拣效率较低，错误率较高，自动化程度很低。因此我们希望设计出成本较低且占用面积较小的小车型自动分拣系统。目前基于emb开发板，查询了相关射频芯片识别资料，将射频芯片识别设备与开发板相连，同时查询了pwm驱动电机方式，将直流电机与开发板相连，分类处理后直接驱动电机运行，模拟小车运行。

## 1.2资源评估

1. EM Starter Kit开发板
2. 与RFID通信的接口模块
3. 直流电机驱动模块
4. 直流电机\*4（自费购买）

## 1.3预期结果

主要意义在于促进快递分拣的自动化程度，希望能够设计出低成本高精度的自动化分拣系统。

预期能够做到快速识别货物并进行分类运输，基本速度希望达到每分钟分拣并发货10件货物左右。

1.4项目实施评估

2018.1.1—2018.2.28 命题选择和相关资料查找与学习，学习embARC开发板相关知识

2018.3.1—2018.4.1 编写程序及中心信息分类系统

2018.4.1—2018.5.1 测试和调节

2018.5.1—2018.5.28 最终检查和纠错

主要难点：通过射频芯片中数据来判断应该到达的的目的地，然后驱动电

机向该目的地前进，即自动寻径是该问题的难点。

工作分配：张 婕（FPGA芯片中数据分析相关程序撰写，辅助技术文档和展示PPT制作）

王开元（硬件结构设计及接口通信协议编写与编程任务）

李浩然（整体方案设计，文档编写和展示视频与文档制作，辅助编程）

# 第二章 作品难点与创新

## 2.1作品难点分析

2.1.1通信协议

本作品中牵涉到模块与fpga芯片间的通信，该通信是通过固定的协议来完成，协议中规定了数据交换的内容和交换方式及时间。这就需要根据使用的模块的接口类型来编写接口通信协议。

## 2.1.2自动寻径如何实现

## 本作品要实现的目标是通过读取RFID的数据，通过该数据来判断应该到达的目的地，然后驱动电机行进到该目的地。如何到达该目的地就是问题的难点，考虑到大多数的实际情况下目的地皆是已经预设好的，且目的地地点并不会经常发生变化，我们可以使用预设路径这一较为简单的方法来实现寻径。让RFID内的数据仅代表目的地编号，在程序内部预先设定好每个编号目的地的行进方式。

## 2.2创新性分析

在快递信息识别中加入了RF芯片，进行中短距离信息采集，大幅度提高信息收集速度，减少错误率。

2.3小结

## 本节主要介绍了该自动发货系统的主要难点和创新点。

# 第三章 系统结构与硬件实现

## 3.1系统原理分析

本作品中包含接口模块和直流电机驱动模块，接口模块负责与RFID模块通信，直流电机驱动模块负责产生pwm波以控制电机运行。实际所需要行进的路线是人为计算与设定完成的，将路径编写至代码中并配置到fpga芯片中。首先系统启动之后开始判断接受RFID信息，与代码内ID信息做对比，若其信息与代码内的ID之一相符，择选择相应的路径驱动电机，若不符则等待下一次接受RFID信息。到达目的地之后，系统会根据过来的路线原路返回，返回至起点之后等待下一次接受RFID信息。

## 3.2 系统结构

## 3.3硬件实现

本作品是在embARC平台下实现的快递自动发货系统，fpga芯片驱动RFID模块读取ID卡内信息。将ID卡内信息与内部预设信息做比对之后，选择预设路线驱动直流电机前进。Fpga芯片产生pwm信号控制直流电机，由于pwm无法直接驱动，因此需要驱动电路驱动电机。到达预设地点后，继续驱动电机按照预设路径原路返回出发点，等待下一次识别ID进入工作循环。

电机驱动芯片采用HG7881四路驱动模块，驱动电压2.5-12V，考虑到寻迹功能对小车灵活性的要求，采用对左右轮独立控制的方式，可以实现前转、一侧停止一侧转动，可以完成向左转、向右转、旋转等动作。该模块是用HG7881通过接收开发板传来的电机转向信号和PWM信号，经过电路的功率放大作用，控制电机的转动方向和速度。电机和HG7881模块协调作用，在开发板的控制作用下，完成小车的基本行驶功能。

射频识别模块采用PN5180模块，PN5180多协议NFC前端具有高输出效率，动态功率控制，自适应波形控制和自适应接收器控制和多个省电功能，即使在恶劣条件下，也可以实现高效，稳定且可靠的运行。通过spi驱动信息的读取和录入，稳定传输到开发板进行识别判断。

## 3.4 小结

本节主要介绍了系统的原理及主要结构，规划了整体系统所需要的硬件结构。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 第四章 软件设计流程及实现

## 4.1软件设计流程

## 

* 1. 软件代码

static long int t0,t1,t2,t3,t4,t5,t6,d\_time;

static int Duty\_cycle0,Duty\_cycle1;

static int PWM0,PWM1;

static DEV\_GPIO \*motor\_port;

static void timer0\_isr(void \*ptr)

{

timer\_int\_clear(TIMER\_0);

if(t0>=100)

t0=0;

if(t0<=Duty\_cycle0)

{

PWM0 = 1;

}

else

{

PWM0 = 0;

}

if(t0<=Duty\_cycle1)

{

PWM1 = 1;

}

else

{

PWM1 = 0;

}

if(t6)

{

motor\_write(((PWM0<<11)+(0<<10)+(PWM1<<9)+(0<<8)),EMSK\_GPIO\_C\_VAILD\_MASK);

}

else

{

if(PWM0)

{

if(PWM1)

{

motor\_write(((0<<11)+(1<<10)+(0<<9)+(1<<8)),EMSK\_GPIO\_C\_VAILD\_MASK);

}

else

{

motor\_write(((0<<11)+(1<<10)+(0<<9)+(0<<8)),EMSK\_GPIO\_C\_VAILD\_MASK);

}

}

else

{

if(PWM1)

{

motor\_write(((0<<11)+(0<<10)+(0<<9)+(1<<8)),EMSK\_GPIO\_C\_VAILD\_MASK);

}

else

{

motor\_write(((0<<11)+(0<<10)+(0<<9)+(0<<8)),EMSK\_GPIO\_C\_VAILD\_MASK);

}

}

}

t0++;

if(t2>=t5)

{

t2=0;

int\_disable(INTNO\_TIMER0);

}

t2++;

EMBARC\_PRINTF("t2 = %d\n",t2);

}

static void timer1\_isr(void \*ptr)

{

timer\_int\_clear(TIMER\_1);

if(t4>=t3)

{

t4=0;

int\_disable(INTNO\_TIMER1);

}

t4++;

EMBARC\_PRINTF("t4 = %d\n",t4);

}

void motor\_init(void)

{

motor\_port = gpio\_get\_dev(0x02);

EMSK\_GPIO\_CHECK\_EXP\_NORTN(motor\_port != NULL);

if (motor\_port->gpio\_open(DW\_GPIO\_MASK\_ALL) == (-6)) {

motor\_port->gpio\_control(GPIO\_CMD\_SET\_BIT\_DIR\_OUTPUT, (void \*)(DW\_GPIO\_MASK\_ALL));

motor\_port->gpio\_control(GPIO\_CMD\_DIS\_BIT\_INT, (void \*)(DW\_GPIO\_MASK\_ALL));

}

motor\_write(0, DW\_GPIO\_MASK\_ALL);

EMBARC\_PRINTF("motor\_init\_ok");

error\_exit:

return;

}

void motor\_write(uint32\_t motor\_val, uint32\_t mask)

{

EMSK\_GPIO\_CHECK\_EXP\_NORTN(motor\_port != NULL);

motor\_val = (~motor\_val) & mask;

motor\_val = motor\_val;

mask = (mask) & DW\_GPIO\_MASK\_ALL;

motor\_port->gpio\_write(motor\_val, mask);

error\_exit:

return;

}

void time\_init(void)

{

t0 = 0;

t1 = 0;

t2 = 0;

t3 = 0;

t4 = 0;

t5 = 0;

d\_time = 0;

Duty\_cycle0 = 70;

Duty\_cycle1 = 70;

int\_disable(INTNO\_TIMER0);

int\_disable(INTNO\_TIMER1);

timer\_stop(TIMER\_0);

timer\_stop(TIMER\_1);

int\_handler\_install(INTNO\_TIMER0, timer0\_isr);

int\_handler\_install(INTNO\_TIMER1, timer1\_isr);

int\_pri\_set(INTNO\_TIMER0, INT\_PRI\_MIN);

int\_pri\_set(INTNO\_TIMER1, INT\_PRI\_MIN);

int\_enable(INTNO\_TIMER0);

int\_enable(INTNO\_TIMER1);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

timer\_start(TIMER\_1, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

int\_disable(INTNO\_TIMER0);

int\_disable(INTNO\_TIMER1);

EMBARC\_PRINTF("time\_init\_ok");

// motor\_write(0xa00, DW\_GPIO\_MASK\_ALL);

}

void slvdev\_init(uint32\_t freq)

{

spi\_master = spi\_get\_dev(SPI\_MASTER\_ID); /\* Get SPI device, DW\_SPI\_0\_ID/DW\_SPI\_1\_ID \*/

spi\_master->spi\_open(DEV\_MASTER\_MODE, freq); /\* Open SPI device in master mode \*/

spi\_master->spi\_control(SPI\_CMD\_SET\_CLK\_MODE, CONV2VOID(SPI\_CLK\_MODE\_3)); /\* Set SPI clock mode \*/

spi\_master->spi\_control(SPI\_CMD\_SET\_DFS, CONV2VOID(SPI\_SLV\_DFS)); /\* Set SPI data frame size \*/

}

void dump\_data(uint8\_t \*buf, uint8\_t len)

{

EMBARC\_PRINTF("DUMP DATA:");

for (int i = 0; i < len; i++) {

EMBARC\_PRINTF("%d\t", buf[i]);

}

EMBARC\_PRINTF("\r\n");

}

DEV\_SPI\_TRANSFER xfer;

/\*\*

\* \brief SPI device transfer function

\* \param[in] \*tx\_buf transmit buffer pointer

\* \param[in] \*rx\_buf receive buffer pointer

\* \param[in] len buffer length

\*/

void slvdev\_xfer(uint8\_t \*tx\_buf, uint8\_t \*rx\_buf, uint8\_t len)

{

DEV\_SPI\_XFER\_SET\_TXBUF(&xfer, tx\_buf, 0, len); /\* Set transmit buffer \*/

DEV\_SPI\_XFER\_SET\_RXBUF(&xfer, rx\_buf, 0, len); /\* Set receive buffer \*/

DEV\_SPI\_XFER\_SET\_NEXT(&xfer, NULL); /\* Set next SPI transfer \*/

spi\_master->spi\_control(SPI\_CMD\_MST\_SEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Set SPI slave device \*/

spi\_master->spi\_control(SPI\_CMD\_TRANSFER\_POLLING, &xfer); /\* Start the transfer by polling \*/

spi\_master->spi\_control(SPI\_CMD\_MST\_DSEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Deset SPI slave device \*/

}

void slvdev\_write(uint8\_t \*buf, uint8\_t len)

{

spi\_master->spi\_control(SPI\_CMD\_MST\_SEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Select SPI slave device \*/

spi\_master->spi\_write(buf, len); /\* Write operation by polling \*/

spi\_master->spi\_control(SPI\_CMD\_MST\_DSEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Deselect SPI slave device \*/

}

/\*\*

\* \brief SPI device read function

\* \param[in] \*buf buffer pointer

\* \param[in] len buffer length

\*/

void slvdev\_read(uint8\_t \*buf, uint8\_t len)

{

spi\_master->spi\_control(SPI\_CMD\_MST\_SEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Select SPI slave device \*/

spi\_master->spi\_read(buf, len); /\* Read operation by polling \*/

spi\_master->spi\_control(SPI\_CMD\_MST\_DSEL\_DEV, CONV2VOID(SPI\_SLV\_CS)); /\* Deselect SPI slave device \*/

}

//#define forward 0

//#define left\_forward 1

//#define right\_forward 2

//#define backward 3

//#define left\_backward 4

//#define right\_backward 5

void main(void)

{

int i;

int temp0[10];

int temp1[10];

uint8\_t \*tx\_buf = (uint8\_t \*)tx\_data;

uint8\_t \*rx\_buf = (uint8\_t \*)rx\_data;

temp0 [0] = forward;

temp0 [1] = left\_forward;

temp0 [2] = forward;

temp0 [3] = right\_forward;

temp0 [4] = forward;

temp0 [5] = backward;

temp0 [6] = right\_backward;

temp0 [7] = backward;

temp0 [8] = left\_backward;

temp0 [9] = backward;

temp1 [0] = forward;

temp1 [1] = right\_forward;

temp1 [2] = forward;

temp1 [3] = left\_forward;

temp1 [4] = forward;

temp1 [5] = backward;

temp1 [6] = left\_backward;

temp1 [7] = backward;

temp1 [8] = right\_backward;

temp1 [9] = backward;

EMBARC\_PRINTF("temp0=%p",temp0);

EMBARC\_PRINTF("temp1=%p",temp1);

cpu\_lock();

uint32\_t temp;

board\_init();

MUX\_REG \*mux\_regs;

mux\_regs = (MUX\_REG \*)(MUX\_REG \*)(\_arc\_aux\_read(0x20a)|0x00000000U);

mux\_init(mux\_regs);

set\_pmod\_mux(mux\_regs,0x10000);

temp=get\_pmod\_mux(mux\_regs);

EMBARC\_PRINTF("pmod reg = %x",temp);

motor\_init();

time\_init();

slvdev\_init(SPI\_SLAVE\_FREQ);

EMBARC\_PRINTF("e\_ok\n");

cpu\_unlock();

while(1)

{

//0x00,0x00,0xFF,0x03,0xFD,0xD4,0x14,0x01,0x17,0x00

tx\_buf[0] = 0x00;

tx\_buf[1] = 0x17;

tx\_buf[2] = 0x01;

tx\_buf[3] = 0x14;

tx\_buf[4] = 0xd4;

tx\_buf[5] = 0xfd;

tx\_buf[6] = 0x03;

tx\_buf[7] = 0xff;

tx\_buf[8] = 0x00;

tx\_buf[9] = 0x00;

slvdev\_write(tx\_buf, 10);

delay(1000);

//0x00,0x00,0xFF,0x04,0xFc,0xd4,0x4A,0x02,0x00,0xdf,0x00;

tx\_buf[0] = 0x00;

tx\_buf[1] = 0xdf;

tx\_buf[2] = 0x00;

tx\_buf[3] = 0x02;

tx\_buf[4] = 0x4a;

tx\_buf[5] = 0xd4;

tx\_buf[6] = 0xfc;

tx\_buf[7] = 0x04;

tx\_buf[8] = 0xff;

tx\_buf[9] = 0x00;

tx\_buf[10] = 0x00;

slvdev\_write(tx\_buf, 11);

delay(1000);

slvdev\_read(rx\_buf, BUF\_LEN);

delay(1000);

slvdev\_read(rx\_buf, BUF\_LEN);

//85206854 85206855

if( rx\_buf[7] == 8 && rx\_buf[6] == 5 && rx\_buf[5] == 2 && rx\_buf[4] == 0 && rx\_buf[3] == 6 && rx\_buf[2] == 8 && rx\_buf[1] == 5 && rx\_buf[0] == 4 )

{

EMBARC\_PRINTF("rf\_ok\n");

for(i = 0;i < 10;i++)

{

car\_set(temp0[i]);

delay(1000);

}

}

if( rx\_buf[7] == 8 && rx\_buf[6] == 5 && rx\_buf[5] == 2 && rx\_buf[4] == 0 && rx\_buf[3] == 6 && rx\_buf[2] == 8 && rx\_buf[1] == 5 && rx\_buf[0] == 5 )

{

EMBARC\_PRINTF("rf\_ok\n");

for(i = 0;i < 10;i++)

{

car\_set(temp1[i]);

delay(1000);

}

}

}

}

void car\_set(const unsigned int mode)

{

switch(mode)

{

case forward : {

t6 = 1;

t5 = 1000;

Duty\_cycle0 = 99;

Duty\_cycle1 = 99;

EMBARC\_PRINTF("forward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("forward finish\n");

break;}

case left\_forward : {

t6 = 1;

t5 = 1000;

Duty\_cycle0 = 30;

Duty\_cycle1 = 99;

EMBARC\_PRINTF("left\_forward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("left\_forward finish\n");

break;}

case right\_forward :{

t6 = 1;

t5 = 1000;

Duty\_cycle0 = 99;

Duty\_cycle1 = 30;

EMBARC\_PRINTF("right\_forward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("right\_forward finish\n");

break;}

case backward :{

t6 = 0;

t5 = 1000;

Duty\_cycle0 = 99;

Duty\_cycle1 = 99;

EMBARC\_PRINTF("backward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("backward finish\n");

break;}

case left\_backward : {

t6 = 0;

t5 = 10000;

Duty\_cycle0 = 30;

Duty\_cycle1 = 99;

EMBARC\_PRINTF("left\_backward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("left\_backward finish\n");

break;}

case right\_backward : {

t6 = 0;

t5 = 10000;

Duty\_cycle0 = 99;

Duty\_cycle1 = 30;

EMBARC\_PRINTF("right\_backward\n");

int\_enable(INTNO\_TIMER0);

timer\_start(TIMER\_0, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 100);

motor\_write(0x0,EMSK\_GPIO\_C\_VAILD\_MASK);

EMBARC\_PRINTF("right\_backward finish\n");

break;}

}

}

void delay(const unsigned long int time)

{

t3 = time;

int\_enable(INTNO\_TIMER1);

timer\_start(TIMER\_1, TIMER\_CTRL\_IE | TIMER\_CTRL\_NH, 1000);

}

## 4.3小结

本节主要介绍作品软件结构。

# 第五章 系统测试与分析

## 5.1系统分析测试

系统调试，首先对本系统中的各个硬件模块进行调试，然后对整个软件系统的调度进行调试。调试的目的一是为了初步完成各个功能，使他们可以运行起来，正常工作。二是为了进一步优化程序，提高程序的运行效率，减少资源的浪费和时间的耗费。不论是硬件调试还是软件调试，均是从输入级，到中间级，最后到输出级，这样逐级进行检查，验证来完成的。对于调试的顺序，当然也是根据设计的顺序来进行。设计一个模块，就调试一个模块，保证做一个软硬件，实现一个功能。本设计中的硬件模块，直接用驱动代码就可以完成调试。当驱动加载后，就由输入级依次观察和检测中间各级的结果输出和最终的结果是否为预期结果。如果不符，最可能出现的原因就是开路和短路。然后断电用万用表对每个可能出现问题的焊点，每条线进行排查。若这些都没有问题，就用万用表检测芯片的各个管脚电压，是否在正常电压范围内。通过以上的方法，对硬件调试完成后，再进一步调试编写好的顶层软件。

## 5.2 测试环境

### 5.2.1验证开发平台

ARC GNU IDE\_2015

### 5.2.2测试方案

1.先测试小车驱动程序，能否分开控制四个轮子的转速来实现转弯减速，刹车等操作。

2．测试小车能否按照内部设计路径前进，并在设定位置刹车。

3．测试RFID识别模块驱动程序，能否读取RF芯片内部信息，并将信息发送至开发板。

4．在读取到信息后，测试开发板对信息处理和识别，并能否驱动小车按路径前进。

## 5.3测试结果

### 5.3.1功能测试

小车能够按照内部设定将小车运送到指定地点，并返回。

### 5.3.2指标测试

小车能够较精准实现货物运输，并且速度较快。

## 5.3结果分析

测试结果显示，小车基本完成设计功能，单个小车能够出色完成任务。

# 

# 第六章 总结展望

经过本次竞赛，极大的提高了团队的合作力，以及创新思维，大家在一起发挥各自聪明才智，提出了很多重要的设计思路，同时提高了动手能力，以及编程能力。学习了spi,gpio,uart等接口驱动编写，了解了ARC GNU编译工具，极大的增长了自己的学习能力。

但是，由于一些硬件和时间问题，只是做了快递自动分拣系统的基本模型，希望能为快递分拣工作提供新的思路，将全自动快递分拣系统推广的更远。

在经过内部调试后，我们希望能够在之后的作品中加入多个小车协调工作，并通过zigbee模块将信息发送，多辆小车能够在一个系统下协调工作，完成小车调度工作，更贴近快递行业的真实情况。

# 参考文献

1. 基于ARC+EM处理器的嵌入式编程实验
2. ARC\_EM\_Starter\_Kit\_UserGuide
3. PN5180 Evaluation board quick start guide
4. Synopsys\_FOSS\_Notices\_embARC
5. ARC\_EM\_Databook