Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный электротехнический университет

«ЛЭТИ»

Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

отчет

**по лабораторной работе №7**

**по дисциплине «ПСРВ»**

Тема: Программирование низкоуровневого интерфейса

робота манипулятора

|  |  |  |
| --- | --- | --- |
| Студент гр. 3391 |  | Малина А. |
| Преподаватель |  | Дорогов А.Ю. |

Санкт-Петербург

## 2018

## Цель работы:

## 1. Освоение низкоуровневого интерфейса с роботом манипулятором.

## 2. Изучение системных функций QNX идентификации процесса по имени.

## 3. Изучение функции посылки сообщения

***Общие сведения об УРТК***

Учебный робототехнический комплекс состоит из манипулятора, блока управления, устройства ввода-вывода информации и ПЭВМ (Рис.1)



Рис. 1 Структурная схема УРТК

X,Y,Z - координаты линейного перемещения;

F - поворот основания;

W - поворот головки;

S - управление схватом;

D - управление двигателем.

**Манипулятор.**Манипулятор представляет собой устройство из трех взаимно-перпендикулярных ходовых винтов, установленных на подвижном основании и поворотной головки. Привод на ходовые винты и поворотную головку осуществляется от электродвигателей постоянного тока со встроенным редуктором. Поворотная головка оборудована схватом и двигателем, имитирующим привод сверлильного станка. УРТК позволяет имитировать работу обрабатывающих и транспортно-складских устройств.

*Блок управления*. Блок управления и устройство ввода-вывода информации представляет собой электронное устройство, которое позволяет осуществлять работу манипулятора в режиме ручного и автоматического управления. Управление УРТК в режиме ручного управления осуществляется с клавиатуры блока управления, а в режиме автоматического от ПЭВМ. В режиме автоматического управления для определения текущего положения каретки манипулятора используются датчики импульсного типа, установленные по координатам **X,Y,Z**. Импульсы датчика порождаются вращением 6-ти лепестковой крыльчатки, расположенной на валу винтовой пары. ЭВМ осуществляет подсчет импульсов поступивших от датчика с момента начала движения, что позволяет с высокой точностью определить текущее положение каретки манипулятора.

Для правильной работы системы в автоматическом режиме, перед началом работы манипулятор должен быть установлен в исходное состояние, которое затем принимается за нулевое. Для координат **X,Y,Z** существуют герконовые датчики начального положения. Для координат **W,F** - импульсные датчики перемещения отсутствуют, есть только герконовые датчики конечных положений.

**Контроллер робота-манипулятора*.***Контроллер робота построен на основе программируемой микросхемы 580ВВ55 (рис.2). Микросхема предназначена для организации обмена 8-ми битовыми данными и содержит три независимых регистра.



Рис.2.Функциональная схема контроллера робота

*Регистры контроллера настроены на выполнение следующих операций*

регистр A[0...7] - на вывод данных;

регистр C[0...3] -на ввод данных;

регистр C[4...5] -на вывод данных;

регистр B[0...7] -на ввод данных.

**Общие сведения об эмуляторе робота**

Имя программы эмулятора **- roby**.

Эмулятор при инициализации регистрирует свое имя - **"apu/roby".**

***Программный интерфейс*** с эмулятором обеспечивается посредством передачи сообщений. Каждое сообщение состоит из:

- фиксированной части - код команды длиной в 1 байт:

- переменной части - данные команды – слова (переменная типа **unsigned int**):

Эмулятор **roby** возвращает ответ: на команды чтения - ответ содержит прочитанные данные. На команды записи и инициализации ответа не предусмотрено (т.е. формируется ответ нулевой длины),

*Формат управляющих команд и сообщений* ***программного интерфейса****:*

1). Запись в регистр **А** код – 0, данные - записываемое слово;

2). Запись в регистр **С** код – 1, данные - записываемое слово;

3). Чтение регистра **С** код – 2, ответ - прочитанный байт;

4). Чтение регистра **В** код – 3, ответ - прочитанный байт;

5). Инициализация датчика координаты **X**  код – 4, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово;

6). Инициализация датчика координаты **Y** код – 5, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово;

7). Инициализация датчика координаты **Z** код – 6, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово.

**Ход работы:**

1) В среде Momentix создан проект PROG1

2) В функции main() ожидается команда ввода с клавиатуры.

- выбрана команда E – выход из программы

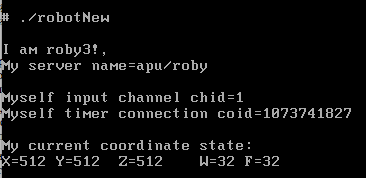
- выбрана команда I – перевод робота-эмулятор в начальное положение

- выбрана команда + или = - вывод в консоль содержимого регистра С в 16-ричном формате

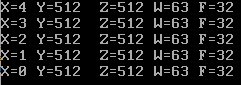
- выбрана команда ! – вывод в консоль содержимого регистра B в 16-ричном формате

- выбрана команда Y – чтение следующей команды. Допустимые команды: Х – движение вперед по координате Х, х – движение назад по координате Х, Y - движение вперед по координате Y, y – движение назад по координате Y, Z - движение вперед по координате Z, z – движение назад по координате Z, W - движение вперед по координате W, w – движение назад по координате W, F - движение вперед по координате F, f – движение назад по координате F, S - Включить/выключить схват S, D - Включить/выключить дрель D.

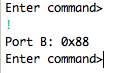
3) Запуск программы эмулятора



4) Выполним команды «движение по Х назад» и «движение по W вперед». Дождемся окончания движения



Прочитаем регистр В

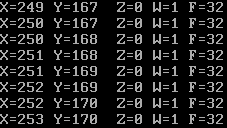


Значение 0x88 – это 0х80 + 0х08, что соответстует константам B\_W\_END и B\_X\_BEGIN, определенным в файле roby.h

5) Выполним команду «движение по Х назад», замерим время перемещения из координаты 512 до 0: 21 секунда

Выполним командыу«движение по Y назад», замерим время перемещения из координаты 512 до 0: 21 секунда

6) Опробуем режим одновременного движения по нескольким коорданатам. Выполним команды «движение по Х вперед», затем «движение по Y вперед»



Эмулятору посылается одно сообщение (число), которое формируется посредством побитовой операции “исключающее или” над числами, которые представляют команды “движение вперед/назад”. Корректная интерпретация возможноа благодаря тому, что константы, описанные в файле roby.h, записаны в особом порядке.

**Приложение**

**Листинг программы prog1.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/iofunc.h>

#include <sys/dispatch.h>

#include <sys/neutrino.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "roby.h"

#define ATTACH\_POINT "apu/roby"

#define COMMANDS\_MAX\_COUNT 10

// пересылаемое сообщение

struct MESSAGE

{

// message type

unsigned char type;

// send data

unsigned int buf;

};

int main()

{

unsigned char PA=0, PC=0; //Эти переменные хранят значения регистров PA и PC

struct MESSAGE msg; // буфер посылаемого сообщения

unsigned char rmsg; // Буфер ответного сообщения

char command, ch[3];

int coid;

coid = name\_open(ATTACH\_POINT, 0);

if(coid == -1)

{

return EXIT\_FAILURE;

}

printf("apu/roby has coid = %d\n", coid);

do

{

printf("Enter command>\n");

scanf("%s", ch);

switch(ch[0])

{

case 'E':

return EXIT\_SUCCESS;

break;

case 'I':

// Двигаться в начальное положение по всем координатам

// и сбросить значения датчиков положений.

PA = 0;

msg.buf = PA;

msg.type = 0;

MsgSend(coid, &msg, sizeof(msg), NULL, 0);

PC = 0; // Для этого сбросить в 0 переменные PA и PC.

command ='C'; // Подать команду записи в регистр PC роботу Roby.

break;

case '+': case '=':

command = 'c';

break;

case '!':

command = 'B';

break;

case 'Y':

scanf("%s", ch);

switch(ch[0])

{

case 'X':

PA ^= A\_X\_FORWARD;

command ='A';

break;

case 'x':

PA ^= A\_X\_BACK;

command ='A';

break;

case 'Y':

PA ^= A\_Y\_FORWARD;

command ='A';

break;

case 'y':

PA ^= A\_Y\_BACK;

command ='A';

break;

case 'Z':

PA ^= A\_Z\_FORWARD;

command ='A';

break;

case 'z':

PA ^= A\_Z\_BACK;

command ='A';

break;

case 'F':

PC ^= C\_F\_FORWARD;

command ='C';

break;

case 'f':

PC ^= C\_F\_BACK;

command ='C';

break;

case 'W':

PC ^= C\_W\_FORWARD;

command ='C';

break;

case 'w':

PC ^= C\_W\_BACK;

command ='C';

break;

case 'S':

PA ^= A\_S;

command ='A';

break;

case 'D':

PA ^= A\_D;

command ='A';

break;

}

}

switch(command)

{

case 'A':

msg.buf = PA;

msg.type = 0;

MsgSend(coid, &msg, sizeof(msg), NULL, 0);

break;

case 'B':

msg.type = 3;

MsgSend(coid, &msg, sizeof(msg), &rmsg, sizeof(rmsg));

printf("Port B: 0x%X\n", rmsg);

break;

case 'c':

msg.type = 2;

MsgSend(coid, &msg, sizeof(msg), &rmsg, sizeof(rmsg));

printf("Port C: 0x%X\n", rmsg);

break;

case 'C':

msg.type = 1;

msg.buf = PC;

MsgSend(coid, &msg, sizeof(msg), NULL, 0);

break;

default:

printf("default last\n");

break;

}

} while(1);

}

**Листинг заголовочного файла roby.h**

**#ifndef** ROBY\_H\_

**#define** ROBY\_H\_

**#define** A\_D 0x01

**#define** A\_S 0x02

**#define** A\_X\_FORWARD 0x04

**#define** A\_X\_BACK 0x08

**#define** A\_Z\_BACK 0x10

**#define** A\_Z\_FORWARD 0x20

**#define** A\_Y\_BACK 0x40

**#define** A\_Y\_FORWARD 0x80

**#define** B\_X 0x01

**#define** B\_Y 0x02

**#define** B\_Z 0x04

**#define** B\_W\_END 0x08

**#define** B\_W\_BEGIN 0x10

**#define** B\_Z\_BEGIN 0x20

**#define** B\_Y\_BEGIN 0x40

**#define** B\_X\_BEGIN 0x80

**#define** C\_F\_END 0x04

**#define** C\_F\_BEGIN 0x08

**#define** C\_W\_FORWARD 0x10

**#define** C\_W\_BACK 0x20

**#define** C\_F\_FORWARD 0x40

**#define** C\_F\_BACK 0x80

**#define** X\_MIN 0

**#define** X\_MAX 1024

**#define** Y\_MIN 0

**#define** Y\_MAX 1024

**#define** Z\_MIN 0

**#define** Z\_MAX 1024

**#define** W\_MIN 0

**#define** W\_MAX 100

**#define** F\_MIN 0

**#define** F\_MAX 50

**#define** S\_MIN 0

**#define** S\_MAX 1

**#define** D\_MIN 0

**#define** D\_MAX 1

**#define** W\_END 0x08

**#define** W\_BEGIN 0x10

**#define** F\_END 0x20

**#define** F\_BEGIN 0x40

**#endif** /\* ROBY\_H\_ \*/