Находящиеся в эксплуатации печи (ламповые, тиристорные) имеют низкий КПД, моральный и физический износ, большие габариты.

Транзисторные печи имеют высокий КПД и благодаря постоянству частоты в индукторе достигается равномерность прогрева.

***Актуальность*** выбранной темы состоит в том что индукционный нагрев применяется с каждым годом все шире и во многих отраслях промышленности и правильное управление процессом плавки, нагрева является главной задачей решаемой сейчас.

***Новизна*** данного проекта заключается в том что используется микроконтроллерная система управления построенная на контроллере ATtiny2313 и с помощью которого осуществляется:

* Плавная регулировка частоты и мощности
* Контроль входных и выходных характеристик
* Защита от ненормальных режимов

***Практической значимостью*** работыявляется:

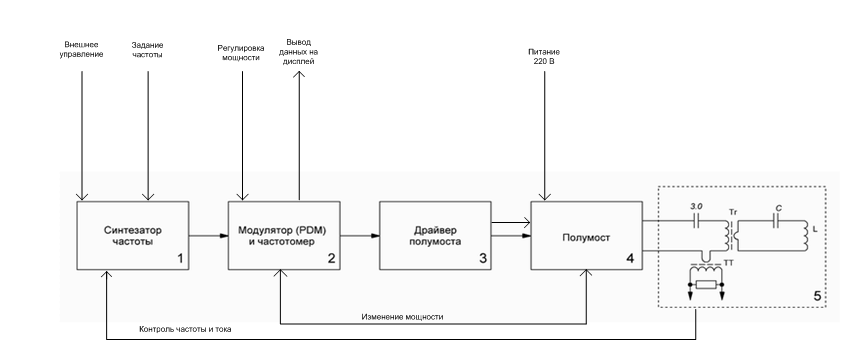
Возможность применения индукционных печей для плавки в вакууме для получения сверхчистых металлов и их сплавов благодаря бесконтактному способу нагрева и перемешиванию расплава магнитными полями, а так же для закалки и пайки.

Результаты исследования могут быть использованы для реального применения на заводских установках индукционного нагрева.

Таким образом, необходимо решить следующие задачи:

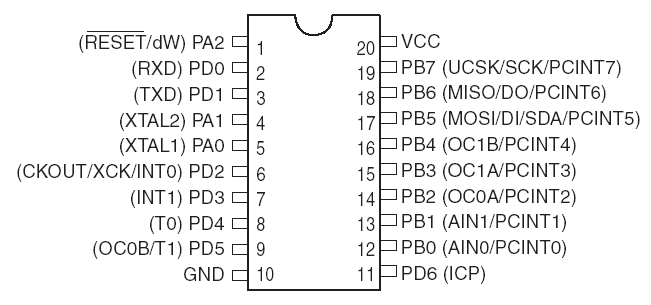
* Разработать структурную и принципиальную схему
* Написать алгоритм по которому будет работать МК
* Выбрать уставки защит

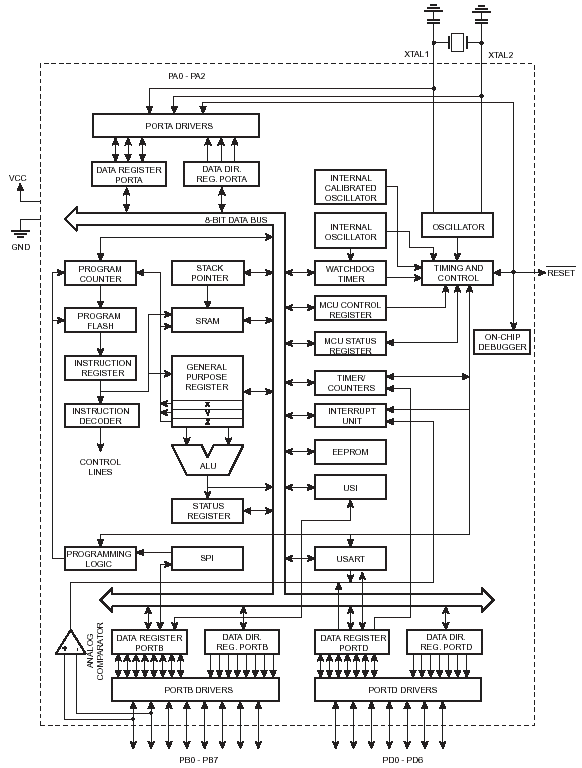
**Структурная схема**

**СТРУКТУРНАЯ СХЕМА**

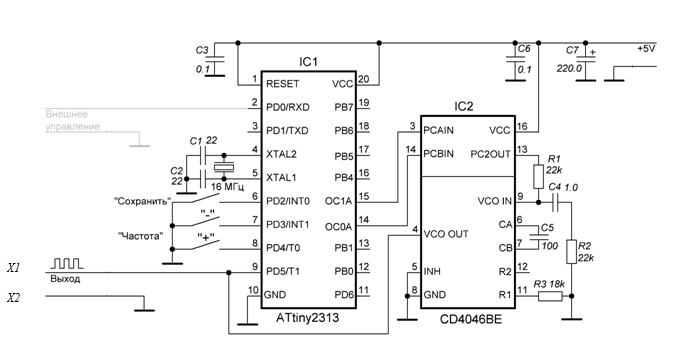


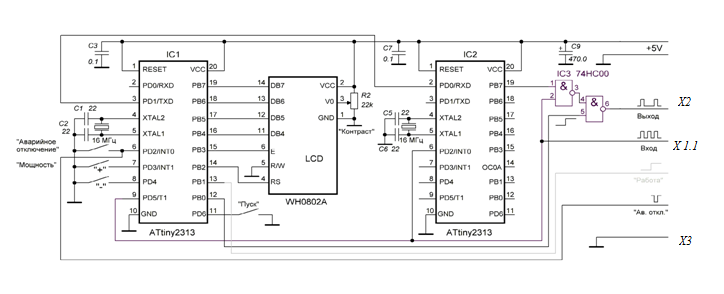
**ATtiny2313 РАСПИНОВКА И ВНУТРЕННЯЯ СТРУКТУРА**



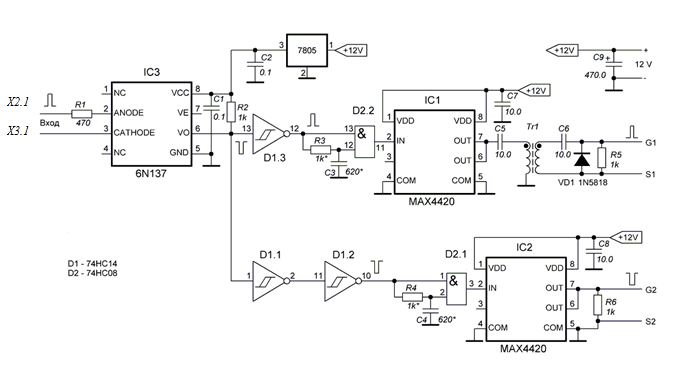


**ПРИНЦИПИАЛЬНАЯ СХЕМА ГЕНЕРАТОРА ЧАСТОТЫ**

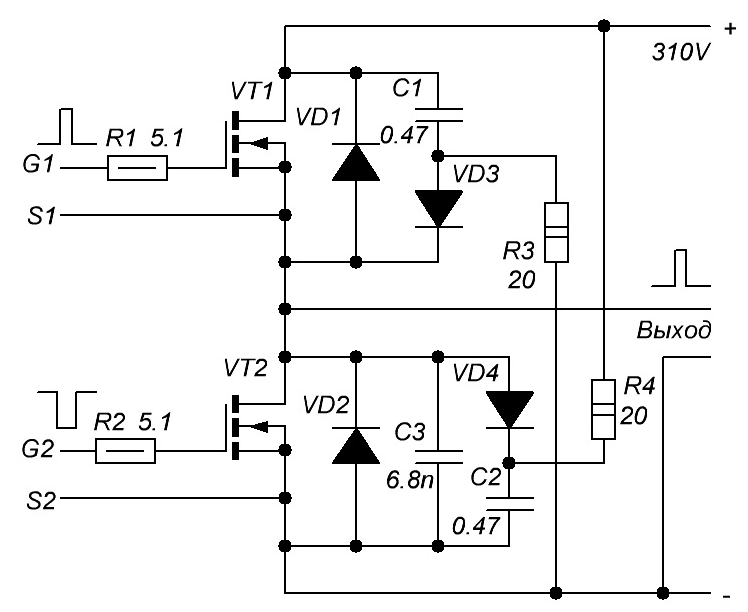


**ПРИНЦИПИАЛЬНАЯ СХЕМА МОДУЛЯТОРА ПЛОТНОСТИ ИМПУЛЬСОВ**

**ПРИНЦИПИАЛЬНАЯ СХЕМА ДРАЙВЕРА**



**ПРИНЦИПИАЛЬНАЯ СХЕМА СИЛОВОЙ ЧАСТИ ИНВЕРТОРА**



**Листинг программы МК**

//------------------------------------------------------------------------------

// Имя файла: Syn/main.c

// Назначение: ФАПЧ-синтезатор частоты

// Обеспечиваются следующие функции:

// 1. Опрос кнопок установки частоты ("Частота" +/-)

// 2. Опрос кнопки сохранения в EEPROM текущего значения частоты ("Сохранить")

// 3. Установка коэффициента пересчета счетчика TIMER1 (по каналу A) в зависимости

// от частоты, установленной пользователем.

//

// Основные характеристики:

// Диапазон регулирования частоты: 100 - 300 кГц с минимальным шагом 62.5 Гц

// с возможностью сохранения текущей частоты в EEPROM микроконтроллера

// При следующем рестарте системы установится сохраненная частота

// При удержании кнопок "Частота +/-" более 0.5 сек соответствующие

// шаги постепенно увеличиваются до 1 кГц.

//------------------------------------------------------------------------------

// Устройство: ATtiny2313

// Тактовая частота: 16МГц

//------------------------------------------------------------------------------

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <avr/signal.h>

#include <avr/eeprom.h>

#define SaveBut PD2

//--------------------------------------------------------------------------------

// Эти определения (до следующего разделителя) изменять нельзя, т.к.

// (для компактности кода !) конкретное расположение битов используется в алгоритме

#define NoClick 0

#define FPlusShortClick 0b00001000

#define FMinusShortClick 0b00010000

#define FPlusLongClick 0b00000001

#define FMinusLongClick 0b00000010

#define FPButsMask 0b00011000

//--------------------------------------------------------------------------------

#define PAUSE 10 // Пауза в мс для подавления дребезга контактов кнопок/энкодера

#define NMin 1599 // Минимальное значение коэффициента пересчета счетчика в петле ФАПЧ

#define NMax 4799 // Его максимальное значение

unsigned int N; // Коэффициент умножения

unsigned int EEMEM Neep = 2399;

unsigned char M = 249; // 124; // Коэффициент деления

unsigned char d;

unsigned char rStep = 0; // Шаг изменения N при длительном удерживании кнопки

unsigned char FChangeFlag = 1, PChangeFlag = 1;

unsigned char P; // Установленное значение мощности (%)

unsigned long F; // Установленное значение частоты (Гц)

//-----------------------------------------------------------------------------

// Опрос кнопок "Частота" и "Мощность"

//-----------------------------------------------------------------------------

unsigned char get\_key(void){

unsigned char i, bFlag = PIND & FPButsMask;

if(bFlag == FPButsMask){ // Не нажата ни одна из кнопок

rStep = 0; return NoClick;

}

\_delay\_ms(PAUSE); // Пережидаем дребезг контактов

if((PIND & FPButsMask) == FPButsMask){ // Какое-то недоразумение...

rStep = 0; return NoClick;

}

if(rStep){ // Продолжается "длинное нажатие"

if(rStep < 16) // Увеличиваем шаг

rStep++;

\_delay\_ms(200); // Подождем, чтобы не сильно было повторение

return (bFlag >> 3); // Продолжается длинное нажатие

}

for(i = 0; i < 50; i++){ // Тестируем состояние кнопок в течение 0.5 сек

if((PIND & FPButsMask) == FPButsMask) // Все-таки было короткое нажатие

return bFlag;

\_delay\_ms(10);

}

rStep = 2; // Началось "длинное нажатие"

return (bFlag >> 3);

}

//-----------------------------------------------------------------------------

// Опрос кнопки "Сохранить"

//-----------------------------------------------------------------------------

void save(void){

if(PIND & \_BV(SaveBut)) // Не нажата ни одна из кнопок

return;

\_delay\_ms(PAUSE); // Пережидаем дребезг контактов

if(PIND & \_BV(SaveBut)) // Какое-то недоразумение...

return;

eeprom\_write\_word(&Neep, N); // Сохраним N в EEPROM

}

//------------------------------------------------------------------------------

// Обработка прерывания по совпадению TIMER1 по каналу A

// Для устранения сбоев в работе счетчика при модификации OCR1A это лучше делать

// в момент сброса счетчика

//------------------------------------------------------------------------------

ISR(TIMER1\_COMPA\_vect){

TIMSK &= ~\_BV(OCIE1A); // Запрещаем прерывания по совпадению с OCR1A

OCR1AH = (N >> 8) & 0x00FF; // Задаем новое значение коэффициента пересчета

OCR1AL = N & 0x00FF;

FChangeFlag = 0;

}

//------------------------------------------------------------------------------

// Основная программа

//------------------------------------------------------------------------------

int main (void) {

DDRB = 0b00001100; PORTB = 0b11110011; // Настраиваем порт B

DDRD = 0b00000000; PORTD = 0b11111111; // Весь порт D - на ввод. Активируем подтягивающие резисторы

// Задаем режим работы таймера-счетчика 1

TCCR1A = \_BV(COM1A0); // COM1A1 = 0 и COM1A0 = 1 - переключение OC1A в момент совпадения

TCCR1B = \_BV(WGM12) | // WGM = 0100 - режим CTC

\_BV(CS12)| // CS = 111 - от внешнего источника (по фронту)

\_BV(CS11)|

\_BV(CS10);

N = eeprom\_read\_word(&Neep); // Читаем значение N из EEPROM

// OCR1AH = (N >> 8) & 0x00FF; // Инициализируем счетчик

// OCR1AL = N & 0x00FF;

// Задаем режим работы таймера-счетчика 0

TCCR0A = \_BV(COM0A0) | // COM0A1 = 0 и COM0A0 = 1 - переключение OC0A в момент совпадения

\_BV(WGM01); // WGM = 010 - режим CTC

TCCR0B = \_BV(CS02)| // CS = 101 - от внутреннего источника

\_BV(CS00); // с предделителем 1024

OCR0A = M; // Инициализируем счетчик

FChangeFlag = 1;

sei();

while(1) {

switch(get\_key()){ // Читаем состояние клавиш

case FPlusShortClick: // Короткое нажатие клавиши увеличения частоты

if(N < NMax){

N++; FChangeFlag = 1;

}

break;

case FMinusShortClick: // Короткое нажатие клавиши уменьшения частоты

if(N > NMin){

N--; FChangeFlag = 1;

}

break;

case FPlusLongClick: // Длинное нажатие клавиши увеличения частоты

if(N <= (NMax - rStep)){

N += rStep; FChangeFlag = 1;

}else{

N = NMax; FChangeFlag = 1;

}

break;

case FMinusLongClick: // Длинное нажатие клавиши уменьшения частоты

if(N >= (NMin + rStep)){

N -= rStep; FChangeFlag = 1;

}else{

N = NMin; FChangeFlag = 1;

}

break;

}

if(FChangeFlag){

TIMSK |= \_BV(OCIE1A); // Разрешаем прерывания по совпадению для модификации OCR1A

}

save(); // Проверим состояние кнопки "Save" и сохраним соотвествующие переменные, если нажата

}

return 0;

}

Имя файла: PDM.asm

; Назначение: Программа для МК ATtiny2313, который работает в качестве модулятора

;плотности импульсов (частный случай сигма-дельта модуляции).

;Сигнал от внешнего генератора меандра подается на вход INT0

;микроконтроллера. Сигнал разрешения прохождения импульса - с вывода PB7.

;Уровень модуляции (т.е. количество выходных импульсов на 100 тактов

;внешнего генератора меандра) поступает в МК через USART от внешнего

;управляющего устройства (другой МК, ПК и т.п.). Формат кадра: 8 бит,

;2 стоп-бита, скорость - 250 кбит/сек)

;Модулятор предназначен для блока управления инвертора для индукционного нагрева.

; Основные характеристики модулятора:

;Диапазон регулировки плотности импульсов: 0 - 100% с шагом 1%

;Максимальная частота внешнего генератора меандра - до 300 кГц

;Уровень сигналов: ТТЛ 5V.

; Микроконтроллер: ATtiny2313

; Тактовая частота: 16МГц

;------------------------------------------------------------------------------

.include "tn2313def.inc"

.list

.def temp = R16 ; Рабочая переменная

.def lev = R17 ; Задаваемый уровень сигнала

.def err = R18 ; Текущая ошибка дискретизации

.def a = R19 ; Текущий уровень сигнала с учетом ошибки

.def bout = R20 ; Байт для вывода в порт B

.equ max = 100 ; "Аналоговый" сигнал изменяется от 0 до max

.equ maxD2 = 50 ; Порог дискретизации

rjmp RESET ; Reset Handler

rjmp INT\_0 ; External Interrupt0 Handler

;------------------------------------------------------------------------------

; Эта процедура вызывается в момент прихода спада импульса меандра (начало нулевого

; полупериода). В течение этого полупериода принимается решение о том, какой

; уровень сигнала будет на выходной шине в течение следующего полупериода и

; вычисляется ошибка дискретизации для следующего периода входного меандра.

; Краткое описание:

; С учетом ошибки (err), полученной на предыдущем такте, вычисляется новое значение

; уровня модуляции (a). Это значение сравнивается с серединой дипазона (maxD2 = 50).

; Если оно больше половины, то - на следующем полупериоде (высокий уровень меандра)

; на выходной шине будет установлен высокий уровень сигнала, если нет - низкий.

; В зависимости от этого вычисляется новая ошибка дискретизации для ее переноса на

; следующий период.

;------------------------------------------------------------------------------

INT\_0: cbi PORTB, 7 ; Сбросим сигнал разрешения

sbis UCSRA,RXC ; Если байт с USART еще не получен, то lev

rjmp Calc ; остается прежним

in lev, UDR ; В противном случае считываем новое значение.

Calc: mov a, lev ; Вычисляем уровень сигнала с коррецией на ошибку

add a, err ; которую мы внесли на предыдущем такте a = lev+err

brvs HiSig ; Если a > 127, то устанавливается флаг переполнения.

; Здесь нужно идти сразу на HiSig. Иначе это значение

; МК воспримет как отрицательное и < maxD2 (!)

cpi a, maxD2 ; В остальных случаях сравниваем "a" с половиной

brge HiSig ; диапазона

LoSig: andi bout, 0b01111111 ; На следующем полупериоде будет 0

mov err, a ; Т.к. ноль в 7-м бите bout уже есть (см. выше), то

rjmp GoOn ; просто вычислим ошибку err = a - 0

HiSig: ori bout, 0b10000000 ; Поставим 1 в 7-й бит bout. Ошибка err = a - max,

mov err, a ; т.е. если a > max, то все равно "недобор" (err > 0),

subi err, max ; а если maxD2 < a < max, то "перебор" (err < 0)

GoOn: out PORTB, bout ; Выставляем бит "Enable"

Exit: reti

;------------------------------------------------------------------------------

; Инициализация и организация общего цикла программы

;------------------------------------------------------------------------------

Reset: ldi temp, RAMEND ; Загрузка указателя стека

out SPL, temp

VarInit: ; Инициализация переменных

clr lev

clr err

clr bout

PortsInit: ; Инициализация портов ввода/вывода

ldi temp, 0xFF ; Весь порт B - на вывод.

out DDRB, temp

out PORTB, bout

ldi temp, 0b00000000 ; Порт D - на ввод

out DDRD, temp

ldi temp, 0b01111111 ; Активируем подтягивающие резисторы

out PORTD, temp

UARTInit: ; Инициализация USART

ldi temp, 3

out UBRRL, temp

clr temp

out UBRRH, temp ; UBRR = 3 - при 16 МГц - 250 кбит/сек

out UCSRA, temp

ldi temp, (1<<RXEN)

out UCSRB, temp ; Прием разрешен

ldi temp, (1<<UCSZ0)|(1<<UCSZ1)|(1<<USBS)

out UCSRC, temp ; Формат кадра - 8 бит и 2 стоп-бита

ExtIntInit: ; Инициализация регистров внешних прерываний

ldi temp, (1<<ISC01)|(1<<ISC11)

out MCUCR, temp ; Оба прерывания - по спаду

ldi temp, (1<<INT0)|(1<<INT1)

out GIMSK, temp ; Разрешаем аппаратные прерывания INT0 и INT1

sei

Start: ; Главный цикл

rjmp Start

//------------------------------------------------------------------------------

// Имя файла: Control/main.c

// Назначение: Программа для МК ATtiny2313, который выполняет сервисные функции

// для функционирования модуля PDM.

// Обеспечиваются следующие функции:

// 1. Измерение частоты внешнего генератора меандра, необходимого для работы

// модулятора и отображение этой частоты на экране ЖК-дисплея

// 2. Опрос кнопок установки мощности (уровня PDM) ("Мощность" +/-)

// 3. Обработка прерывания от кнопки аварийного отключения "Ав.откл"

// 4. Отправка установленной мощности по по последовательному каналу (USART)

// в модулятор (модуль PDM)

//

// Основные характеристики модуля:

// Диапазон измерения частоты: 1 - 1000 кГц (квант - 1 Гц)

// Диапазон установки мощности: 0 - 100% с минимальным шагом 1%. Предусмотрен

// быстрый сброс уровня мощности в 0 при нажатии кнопки "Ав. откл".

// При удержании кнопок "Мощность" более 0.5 сек шаг изменения мощности

// постепенно увеличиваются до 10%.

//

// Передача информации через USART

// Кадр - 8 бит, 2 стоп-бита без контроля четности

// Скорость 250 кбит/сек для кварца 16 МГц

// Дисплей: 2 строчный 8 символьный алфавитно-цифровой дисплей, совместимый

// с HD44780

//

// Устройство: ATtiny2313

// Тактовая частота: 16МГц

//------------------------------------------------------------------------------

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <avr/signal.h>

#define RS PB2

#define E PB3

#define OffBut PD2 // Кнопка аварийного отключения подключена на INT0 (PD2)

#define PPlusBut PD3 // Кнопка мощность "+" - на PD3

#define PMinusBut PD4 // Кнопка мощность "-" - на PD4

#define SingleBut PD6 // Кнопка одиночного упраляющего импульса

//--------------------------------------------------------------------------------

#define NoClick 0

#define PPlusShortClick 0b01010000

#define PMinusShortClick 0b01001000

#define SingleShortClick 0b00011000

#define PPlusLongClick 0b00001010

#define PMinusLongClick 0b00001001

#define SingleLongClick 0b00000011

#define FPButsMask 0b01011000

//--------------------------------------------------------------------------------

#define PAUSE 10 // Пауза в мс для подавления дребезга контактов кнопок/энкодера

#define PMax 100 // Максимальное значение мощности (100%). Минимальное - всегда 0%

char fs[] = {0x20, 0x20, 0x20, 0x2E, 0x20, 0x20, 0x20, 0x6B, 0x00},

ps[] = {0x50, 0x44, 0x4D, 0x3D, 0x20, 0x20, 0x20, 0x25, 0x00};

unsigned long f = 1000000L; // Измеренное значение Частоты

unsigned char fReady = 1, cnt = 0;

unsigned int tcnt;

unsigned char rStep = 0; // Шаг изменения мощности при длительном удерживании кнопки

unsigned char PChangeFlag = 1; // 1, если значение мощности изменено пользователем

unsigned char P; // Установленное значение мощности (%)

//------------------------------------------------------------------------------

// Инициализация USART

//------------------------------------------------------------------------------

void usart\_init(unsigned int ubrr) {

UBRRH = (unsigned char)(ubrr >> 8);

UBRRL = (unsigned char)ubrr;

UCSRB = \_BV(TXEN); // Разрешение передачи

UCSRC = \_BV(UCSZ0) | \_BV(UCSZ1) | \_BV(USBS); //размер слова 8 разрядов и

// 2 стоп-бита

}

//------------------------------------------------------------------------------

// Посылка байта

//------------------------------------------------------------------------------

void usart\_sb(unsigned char b) {

while(!(UCSRA & \_BV(UDRE))); // Ожидаем пока не освободится буфер передатчика

UDR = b;

}

//-----------------------------------------------------------------------------

// Опрос кнопок "Мощность +/-"

//-----------------------------------------------------------------------------

unsigned char get\_key(void){

unsigned char i, bFlag = PIND & FPButsMask;

if(bFlag == FPButsMask){ // Не нажата ни одна из кнопок

rStep = 0; return NoClick;

}

\_delay\_ms(PAUSE); // Пережидаем дребезг контактов

if((PIND & FPButsMask) == FPButsMask){ // Какое-то недоразумение...

rStep = 0; return NoClick;

}

if(rStep){ // Продолжается "длинное нажатие"

if(rStep < 10) // Увеличиваем шаг

rStep++;

\_delay\_ms(200); // Подождем, чтобы не сильно было повторение

return (bFlag >> 3); // Продолжается длинное нажатие

}

for(i = 0; i < 50; i++){ // Тестируем состояние кнопок в течение 0.5 сек

if((PIND & FPButsMask) == FPButsMask) // Все-таки было короткое нажатие

return bFlag;

\_delay\_ms(10);

}

rStep = 2; // Началось "длинное нажатие"

return (bFlag >> 3);

}

//-----------------------------------------------------------------------------

// Обработка прерывания INT0 от кнопки аварийного выключения

//-----------------------------------------------------------------------------

ISR(INT0\_vect){

P = 0;

usart\_sb(P & 0x7F);

PChangeFlag = 1;

}//------------------------------------------------------------------------------

// Запись команды в ЖКИ

//------------------------------------------------------------------------------

void lcd\_cmd(unsigned char p) { // p - байт команды

PORTB = (p & 0b11110000) |

(PORTB & 0b00001111); // Старшая тетрада команды - на шину PB4-PB7

PORTB &= ~\_BV(RS); // RS = 0

PORTB |= \_BV(E); \_delay\_us(100); // E = 1

PORTB &= ~\_BV(E); \_delay\_us(500); // E -> 0 - фронт записи команды в ЖКИ

PORTB = ((p << 4) & 0b11110000) |

(PORTB & 0b00001111); // Младшая тетрада команды - на шину PB4-PB7

PORTB &= ~\_BV(RS); // RS = 0

PORTB |= \_BV(E); \_delay\_us(100); // E = 1

PORTB &= ~\_BV(E); \_delay\_us(500); // E -> 0 - фронт записи команды в ЖКИ

}

//------------------------------------------------------------------------------

// Запись данных в ЖКИ

//------------------------------------------------------------------------------

void lcd\_dat(unsigned char p) { // p - байт данных

PORTB = (p & 0b11110000) | \_BV(RS) |

(PORTB & 0b00001111); // Старшая тетрада данных - на шину PB4-PB7

PORTB |= \_BV(RS); // RS = 1

PORTB |= \_BV(E); \_delay\_us(100); // E = 1

PORTB &= ~\_BV(E); \_delay\_us(500); // E -> 0 - фронт записи данных в ЖКИ

PORTB = ((p<<4) & 0b11110000) | \_BV(RS) |

(PORTB & 0b00001111); // Младшая тетрада данных - на шину PB4-PB7

PORTB |= \_BV(RS); // RS = 1

PORTB |= \_BV(E); \_delay\_us(100); // E = 1

PORTB &= ~\_BV(E); \_delay\_us(500); // E -> 0 - фронт записи данных в ЖКИ

}

//------------------------------------------------------------------------------

// Запись полубайта

//------------------------------------------------------------------------------

void ini\_cmd(unsigned char p) { // p - байт команды, используется только старшая тетрада

PORTB = (p & 0b11110000) |

(PORTB & 0b00001111); // Старшая тетрада команды - на шину PB4-PB7

PORTB &= ~\_BV(RS); // RS = 0

PORTB |= \_BV(E); \_delay\_us(100); // E = 1

PORTB &= ~\_BV(E); \_delay\_us(500); // Фронт записи команды в ЖКИ

}

//------------------------------------------------------------------------------

// Инициализация ЖКИ

//------------------------------------------------------------------------------

void lcd\_init(void) {

\_delay\_ms(20);

ini\_cmd(0x30); \_delay\_ms(5);

ini\_cmd(0x30); \_delay\_us(200);

ini\_cmd(0x30); \_delay\_us(100);

ini\_cmd(0x20); \_delay\_us(100);

lcd\_cmd(0x28); \_delay\_us(100);

lcd\_cmd(0x08); \_delay\_us(100);

lcd\_cmd(0x01); \_delay\_us(2000);

lcd\_cmd(0x06); \_delay\_us(100);

lcd\_cmd(0x0D); \_delay\_us(100);

}

//------------------------------------------------------------------------------

// Запись строки в ЖКИ

//------------------------------------------------------------------------------

void lcd\_str(unsigned char l, char\* s) {

if(l == 0) // Запись в первую строку

lcd\_cmd(0x80);

else // ну тогда во вторую...

lcd\_cmd(0xC0);

\_delay\_us(100);

unsigned char i = 0;

while(s[i] != 0x00)

lcd\_dat(s[i++]);

}

//------------------------------------------------------------------------------

// Обработка прерывания по совпадению TIMER0 по каналу A

// Прерывния инициируются каждые 8 мс, 125 циклов как раз дают 1 сек

//------------------------------------------------------------------------------

ISR(TIMER0\_COMPA\_vect){

tcnt = TCNT1; TCNT1 = 0; cnt++;

f += tcnt;

if(cnt == 125){ // Прошла 1 сек запретим прерывания и установим флаг

fReady = 1; // "готовности" частоты

TIMSK &= ~\_BV(OCIE0A); // Запрет прерываний TIMER0 по каналу A

}

}

//------------------------------------------------------------------------------

// Пуск таймера на 1 сек

//------------------------------------------------------------------------------

void timer\_start(void){

f = 0; fReady = 0; cnt = 0;

TCNT1 = 0; TCNT0 = 0; // Сброс счетчиков

TIFR |= \_BV(OCF0A); // Очистим флаг прерывания (запишем туда 1)

TIMSK |= \_BV(OCIE0A); // Разрешение прерываний TIMER0 по каналу A

}

//------------------------------------------------------------------------------

// Основная программа

//------------------------------------------------------------------------------

int main (void) {

unsigned char i, s, d0, d1, d2, d[6];

DDRB = 0xFF; PORTB = 0x01; // Весь порт B - на вывод.

// Задаем режим работы таймера-счетчика 1. Этот счетчик служит для измерения

// частоты внешнего источника. Временной интервал (1 сек) в течении которого

// происходит счет, задается счетчиком TIMER0 (125 интервалов по 8 мс)

TCCR1A = 0x00; // COM1 = 0000 WGM = 0000 - режим Normal

TCCR1B = \_BV(CS12)| // CS = 111 - от внешнего источника (по фронту)

\_BV(CS11)|

\_BV(CS10);

// Задаем режим работы таймера-счетчика 0. Этот счетчик задает временной

// интервал 8 мс, в течении которого при помощи счетчика TIMER1 подсчитывается

// количество импульсов сигнала, частота которого измеряется

TCCR0A = \_BV(WGM01); // COM0 = 0000, WGM = 010 - режим CTC

TCCR0B = \_BV(CS02)| // CS = 101 - от внутреннего источника

\_BV(CS00); // с предделителем 1024

OCR0A = 124; // 125 тактов дают временной интервал - 8 мс.

// -1 такт - это коррекция на задержки, возникающие

// при вызове обработчиков, суммирования и т.п.

GIMSK |= \_BV(INT0); // Разрешаем аппаратные прерывания Int0. Оно инициируется

// при нажатии кнопки аварийного отключения.

MCUCR = \_BV(ISC01); // Прерывания будут инициированы по спаду

DDRD &= 0b10000011; // Пины 2-6 порта D - на ввод

PORTD |= 0b01111100; // Активируем подтягивающие резисторы

lcd\_init(); // Инициализация ЖКИ

lcd\_cmd(0x0C); \_delay\_us(100);

usart\_init(3); // 250 кбит/сек для кварца 16МГц

TIMSK |= \_BV(OCIE0A); // Разрешаем прерывания по совпадению на счетчике 0

sei();

while(1) {

if(fReady){ // Измерение частоты закончено - отобразим ее на экране LCD

if(f <1000000L) {

for(i = 0; i < 6; i++){ // Определяем десятичные цифры числа

d[i] = f % 10; f /= 10;

}

s = 0;

for(i = 0; i < 6; i++){ // Формируем строку для вывода на LCD

if(i == 3)

s = 1;

fs[i + s] = 0x30 + d[5 - i];

}

if(d[5] == 0){

fs[0] = 0x20;

if(d[4] == 0)

fs[1] = 0x20;

}

} else { // Частота выше 1 МГц - обозначим это звездочками

s = 0;

for(i = 0; i < 6; i++){ // Формируем строку для вывода на LCD

if(i == 3)

s = 1;

fs[i + s] = '\*';

}

}

lcd\_str(0, fs);

timer\_start();

}

switch(get\_key()){ // Читаем состояние клавиш

case PPlusShortClick: // Короткое нажатие клавиши увеличения мощности

if(P < PMax){

P++; PChangeFlag = 1;

}

break;

case PMinusShortClick: // Короткое нажатие клавиши уменьшения мощности

if(P > 0){

P--; PChangeFlag = 1;

}

break;

case PPlusLongClick: // Длинное нажатие клавиши увеличения мощности

if(P <= (PMax - rStep)){

P += rStep; PChangeFlag = 1;

}else{

P = PMax; PChangeFlag = 1;

}

break;

case PMinusLongClick: // Длинное нажатие клавиши уменьшения мощности

if(P >= rStep){

P -= rStep; PChangeFlag = 1;

}else{

P = 0; PChangeFlag = 1;

}

break;

case SingleShortClick: // Короткое нажатие клавиши выдачи одиночного импульса

case SingleLongClick: // Длинное нажатие клавиши выдачи одиночного импульса

PORTB |= \_BV(PB0); // Разрешим прохождение тактовых импульсов на инвертор

break;

}

if(PChangeFlag){

usart\_sb(P & 0x7F); // Отправляем установленное значение мощности в

// модуль PDM

// Отображаем мощность на дисплее

d0 = P % 10; // Определяем десятичные цифры

d1 = (P % 100)/10;

d2 = P/100;

ps[4] = ((d2 == 0)? 0x20 : (0x30 + d2)); // Убираем незначащий 0 слева

ps[5] = ((d2 == 0) && (d1 == 0))? 0x20 : (0x30 + d1); // Тоже...

ps[6] = 0x30 + d0;

lcd\_str(1, ps);

PChangeFlag = 0;

if(P == 0)

PORTB &= ~\_BV(PB0); // Запретим прохождение тактовых импульсов на инвертор

}

}

return 0;

}