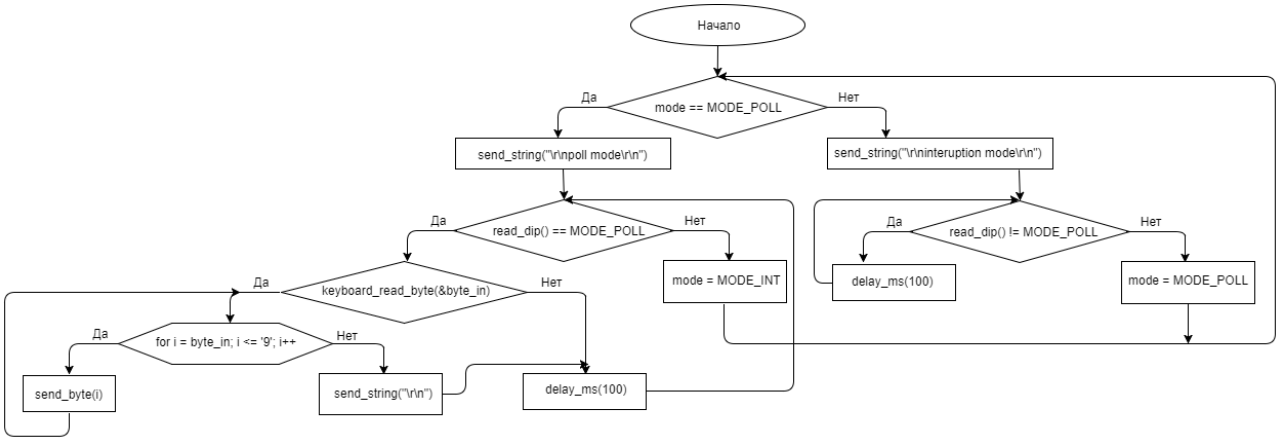
Университет ИТМО
Кафедра ВТ

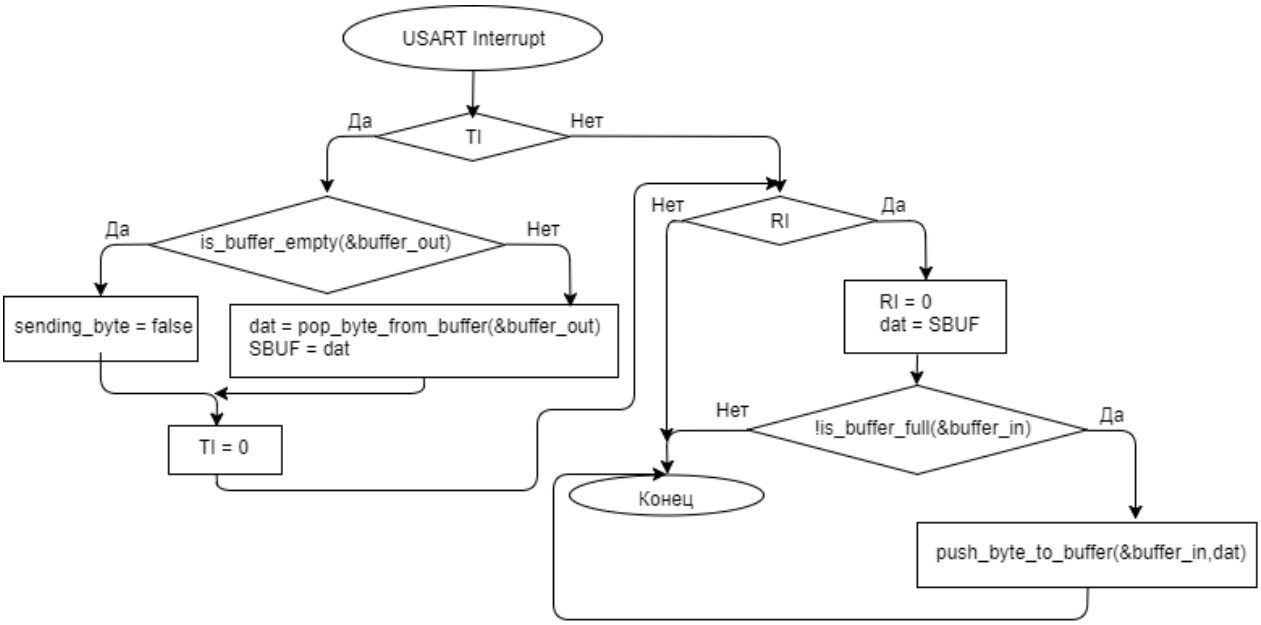
Лабораторная работа по предмету:
ИУС
Лабораторная работа №4

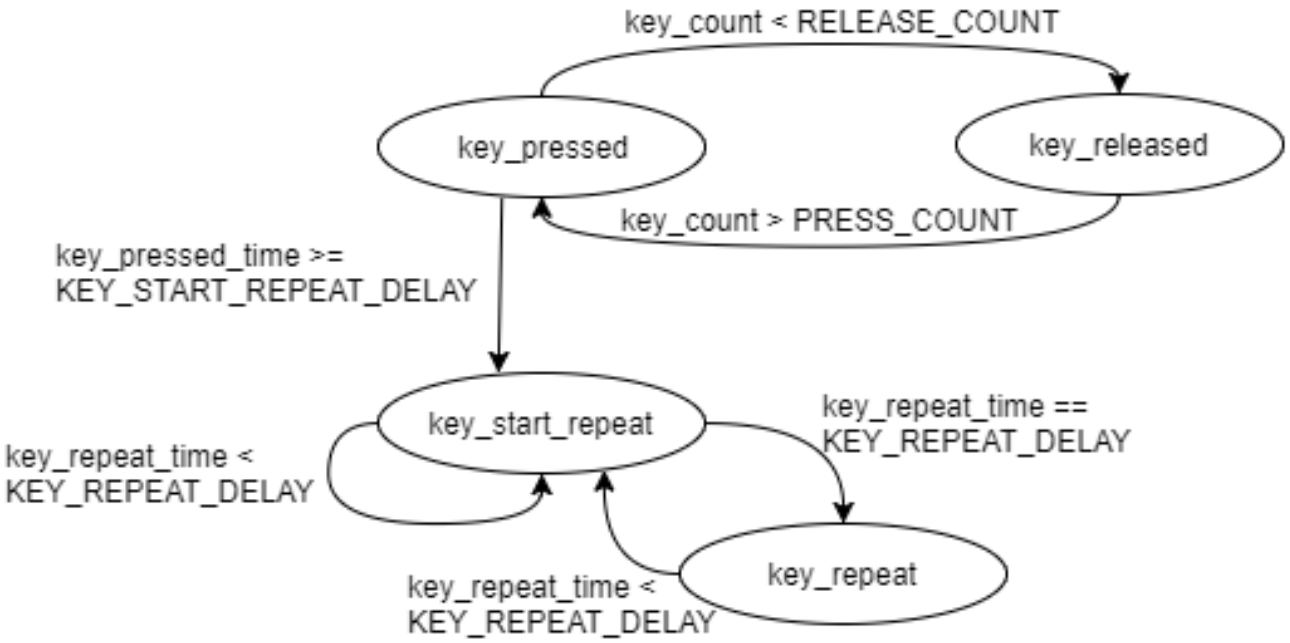
Выполнили:
Гулямова С. И.
Разумовская А. В.
Шляков А. К.

Санкт - Петербург
2017г.

Блок схемы:


Начало
mode == MODE_POLL
Да
Нет
send_string("\r\npoll mode\r\n")
send_string("\r\ninteruption mode\r\n")
read_dip() == MODE_POLL
read_dip() != MODE_POLL
mode = MODE_INT
delay_ms(100)
mode = MODE_POLL
keyboard_read_byte(&byte_in)
for i = byte_in; i <= '9'; i++
send_byte(i)
send_string("\r\n")
delay_ms(100)


USART Interrupt
TI
RI
is_buffer_empty(&buffer_out)
sending_byte = false
dat = pop_byte_from_buffer(&buffer_out)
SBUF = dat
TI = 0
RI = 0
dat = SBUF
!is_buffer_full(&buffer_in)
Конец
push_byte_to_buffer(&buffer_in,dat)


key_count < RELEASE_COUNT
key_pressed
key_released
key_count > PRESS_COUNT
key_pressed_time >=
KEY_START_REPEAT_DELAY
key_start_repeat
key_repeat_time ==
KEY_REPEAT_DELAY
key_repeat_time <
KEY_REPEAT_DELAY
key_repeat
key_repeat_time <
KEY_REPEAT_DELAY

buffer.c

```
void initialize_buffer(Buffer* buffer) {
      buffer->head=0;
      buffer->tail=0;
      buffer->len=0;
}

bool is_buffer_empty(Buffer* buffer) {
      return buffer->len == 0;
}

bool is_buffer_full(Buffer* buffer) {
      return buffer->len == BUFFER_LEN;
}
void push_byte_to_buffer(Buffer* buffer, u8 dat) {
      buffer->mem[buffer->head] = dat;
      buffer->head++;
      buffer->len++;
    if (buffer->head == BUFFER_LEN) {
            buffer->head = 0;
    }
}

u8 pop_byte_from_buffer(Buffer* buffer) {
      u8 dat;
      dat= buffer->mem[buffer->tail];
      buffer->tail++;
      buffer->len--;
      if (buffer->tail == BUFFER_LEN) {
            buffer->tail = 0;
      }
      return dat;
}
```

common.c

```
void SetVector(unsigned char __xdata * Address, void * Vector)
{
      unsigned char __xdata * TmpVector;

      // Первым байтом по указанному адресу записывается
     // код команды передачи управления ljmp, равный 02h
      *Address = 0x02
      // Далее записывается адрес перехода Vector
      TmpVector = (unsigned char __xdata *) (Address + 1);
      *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
      ++TmpVector;
      *TmpVector = (unsigned char) Vector;
      // Таким образом, по адресу Address теперь
      // располагается инструкция ljmp Vector
}
```

handler.c

```
u8 mode;
u8 number;
u8 state;
u8 v;
u8 read_dip(){
	u8 dip;
	//ET2=0;
	dip=read_max(EXT_LO);
	//ET2=1;
	return dip;
}

void reset() {
	number=0;
	state=STATE_NUMBER;
}

void initialize_handler() {
	mode = MODE_POLL;
	reset();
}

void poll_loop() {
	u8 i;
	u8 byte_in;
	while( read_dip()==DIP_POLL_MODE ){
		if( keyboard_read_byte(&byte_in) ){
			for( i=byte_in;i<='9';i++ ){
				send_byte(i);
			}
			send_string("\r\n");
		}
		delay_ms(100);
	}
	mode=MODE_INT;
}

void int_loop() {
	while( read_dip()!=DIP_POLL_MODE ){
		delay_ms(100);
	}
	mode=MODE_POLL;
}

void handler_loop() {
	while(1) {
		if( mode==MODE_POLL ) {
			send_string("\r\npoll mode\r\n");
			poll_loop();

		}else{
```

```
                send_string("\r\ninteruption mode\r\n");
                int_loop();
            }
      }
}

void error() {
      send_string("\r\nerror\r\n");
      leds(0xAA);
      state=STATE_ERROR;
}




u8 to_hex(u8 val) {
      if( val>9 ) {
            return 'A'+val-10;
      }
      return '0'+val;
}

void handler_int() {
      u8 num;
      u8 sym;
      if( !keyboard_read_byte(&sym) ){
            error();
            return;
      }

      if( state==STATE_ERROR ){
            reset();
            leds(0x00);
      }
      switch (state) {
            case STATE_NUMBER:
                  if(sym>='0' && sym<='9'){
                        send_byte(sym);
                        num=sym-'0';
                        if( num > NUMBER_LIMIT-number*10 ) {
                              error();
                              return;
                        }
                        number*=10;
                        number+=num;
                  }else if( sym=='*' ) {
                        state=STATE_CR;
                  }else{
                        error();
                  }
                  break;
```

```
            case STATE_CR:
                  if( sym=='#' ) {
                        send_string("\r\nHex:");
                        send_byte(to_hex(number>>4));
                        send_byte(to_hex(number&0x0F));
                        send_string("\r\n");
                        reset();
                  }else{
                        error();
                  }
                  break;
      }
}
keyboard.c
const u8 ROWS=4;
const u8 COLS=4;
const u8 PRESS_COUNT=8;
const u8 RELEASE_COUNT=3;
const u8 KEY_COUNT_LIMIT=20;
const u8 KEY_START_REPEAT_DELAY=60;
const u8 KEY_REPEAT_DELAY=100;
const u8 SPEAKER_TIME=100;

__xdata Buffer key_clicks;
u8 key_count[KEYS_NUMBER];
u8 key_pressed_time[KEYS_NUMBER];
u8 key_repeat_time[KEYS_NUMBER];
char key_value[]="147*2580369#ABCD";
u8 col;
u8 prescaler;
u8 speaker;

void initialize_keyboard() {
      initialize_buffer(&key_clicks);
      col=0;
      prescaler=0;
}

bool keyboard_read_byte(u8* dat) {
      bool is_data;
      ET2=0;
      is_data=!is_buffer_empty(&key_clicks);
      if( is_data ){
            *dat=pop_byte_from_buffer(&key_clicks);
      }
      ET2=1;
      return is_data;
}
```

```
u8 scan_keyboard() {
	u8 scan_mask;
	u8 row_mask;
	scan_mask=~(1 << col);
	write_max(KB, scan_mask);
	row_mask=~(read_max(KB) & 0xF0)>>4;
	return row_mask;
}

void key_click(u8 key){
	speaker=1;
	enable_speaker();
	if( !is_buffer_full(&key_clicks) ){
		push_byte_to_buffer(&key_clicks, key_value[key]);
		if( mode==MODE_INT ) {
			handler_int();
		}
	}
}

void scan_keyboard_int(){
	u8 row;
	u8 row_mask;
	u8 key;
	prescaler++;
	if( prescaler==2 ){
		prescaler=0;
		row_mask=scan_keyboard();
		for( row=0;row<ROWS;row++ ){
			key=col*ROWS + row;
			if( row_mask>>row & 1 ){
				if( key_count[key]<KEY_COUNT_LIMIT ) {
					key_count[key]++;
					if( key_count[key]>PRESS_COUNT &&
key_pressed_time[key]==0 ){
						key_click(key);
						key_pressed_time[key]=1;
					}
				}
			}else{
				if( key_count[key]>0 ) {
					key_count[key]--;
					if( key_count[key]<RELEASE_COUNT &&
key_pressed_time[key]>0 ){
						key_pressed_time[key]=0;
						key_repeat_time[key]=0;
					}

				}

			}
```

```
                  if( key_pressed_time[key]>0 ){
if( key_pressed_time[key]<KEY_START_REPEAT_DELAY ){
                              key_pressed_time[key]++;
                        }else{
                              key_repeat_time[key]++;
                              if( key_repeat_time[key]==KEY_REPEAT_DELAY ) {
                                    key_click(key);
                                    key_repeat_time[key]=0;
                              }
                        }
                  }
            }
            col++;
            if( col==COLS ){
                  col=0;
            }
      }
      if( speaker>0 ){
            speaker++;
            if(speaker==SPEAKER_TIME){
                  disable_speaker();
                  speaker=0;
            }
      }
}

lab4.c
void delay ( unsigned long ms );
void main( void ) {
      initialize_system_timer();
      initialize_keyboard();
      initialize_uart(S9600);
      initialize_handler();
      initialize_speaker();
      EA=1;
      handler_loop();
}

speaker.c
bool speaker_on;
void T0_ISR( void ) __interrupt ( 2 );
void initialize_speaker(){
      SetVector( 0x200B, (void *)T0_ISR );
      TMOD|=0b00000010;
      ET0=1;
      TH0=-250;
}


void enable_speaker(){
      speaker_on=false;
      TL0=-250;
```

```
	TR0=1;
}

void disable_speaker(){

	TR0=0;
	speaker_on=false;
	write_max(ENA, 0b0100000);
}

void T0_ISR( void ) __interrupt ( 2 ){
	if( speaker_on ){
		write_max(ENA, 0b0111100);
	}else{
		write_max(ENA, 0b0100000);
	}

	speaker_on=!speaker_on;
}

system_timer.c
u16 cnt=0
const u8 LEVEL_ON=0xF0;
const u8 LEVEL_OFF=0x0F;
time cur_ms;
void T2_ISR( void ) __interrupt ( 2 );
void initialize_system_timer() {
	cur_ms=0;
	cnt=0;
	SetVector( 0x202B, (void *)T2_ISR );
	TH2=(-1000)>>8;0;
	TL2=(-1000)&0xFF;
	RCAP2H=(-1000)>>8;
	RCAP2L=(-1000)&0xFF;
	ET2=1;
	TR2=1;
}

time get_ms(void){
	return cur_ms;
}

time get_ms_after(time t0){
	return cur_ms-t0;
}

void delay_ms(time t){
	time now=get_ms();
	while( get_ms_after(now)<t){}
}
```

```
void T2_ISR( void ) __interrupt ( 2 ){
      cur_ms++;
      scan_keyboard_int();
}

uart.c
__xdata Buffer buffer_in;
__xdata Buffer buffer_out;
bool sending_byte;
void UART_ISR( void ) __interrupt ( 4 );
void initialize_uart(u8 speed) {
      initialize_buffer(&buffer_in);
      initialize_buffer(&buffer_out);
      SetVector( 0x2023, (void *)UART_ISR );
      TH1       =  speed;
    TMOD     |=  0x20;
    TCON     |=  0x40;
    SCON      =  0x50;
      ES=1;
}

void send_byte(u8 dat) {
      ES=0;
      if( !sending_byte ){
            sending_byte=true;
            SBUF=dat;
      }else if( !is_buffer_full(&buffer_out) ){
            push_byte_to_buffer(&buffer_out,dat);
      }
      ES=1;
}




void send_string(char * str){
      ES=0;
      if( !sending_byte ){
            sending_byte=true;
            SBUF=*str;
            str++;
      }

      while( *str ) {
            if( !is_buffer_full(&buffer_out) ){
                  push_byte_to_buffer(&buffer_out,*str);
                  str++;
            }
      }
      ES=1;
}
```

```
bool read_byte(u8* dat) {
	bool is_data;
	ES=0;
	is_data=!is_buffer_empty(&buffer_in);
	if( is_data ){
		*dat=pop_byte_from_buffer(&buffer_in);
	}
	ES=1;
	return is_data;
}

void UART_ISR( void ) __interrupt ( 4 ) {
	u8 dat;
	if( TI ){
		if( is_buffer_empty(&buffer_out) ){
			sending_byte=false;
		}else{
			dat=pop_byte_from_buffer(&buffer_out);
			SBUF=dat;
		}
		TI=0;
	}
	if( RI ){
		RI=0;
		dat=SBUF;
		if( !is_buffer_full(&buffer_in) ){
			push_byte_to_buffer(&buffer_in,dat);
		}
	}
}
```