```
/*************************************************************************
       Read/write Control program for Ring bufer on EEPROM

       Programmed by Kenji Arai/JH1PJL
       E-mail:     kenjia@sannet.ne.jp  jh1pjl@arrl.net
       URL:        http://www.page.sannet.ne.jp/kenjia/

              August     1,2004     Start coding
              November  22,2004
              September 18,2005     Change to H8/3694F
              October  23,2005

       Copyright (C) 2004,'05  Kenji Arai/JH1PJL
       All rights reserved. Permission is granted to use, modify,or redistribute
       this software so long as it is not sold or exploited for profit.

       THIS SOFTWARE IS PROVIDED AS IS AND WITHOUT WARRANTY OF ANY KIND,
       EITHER EXPRESSED OR IMPLIED.
*************************************************************************/
/*  -----< Include Files >-------------------------------------------------*/
#include      "rtm_H8_3664.h"
#include      "ringbuf.h"
#include      "iiceep.h"
#include      "const.h"
#include      "task.h"

/*  -----< Function Prototype >--------------------------------------------*/
void ebzr(void);                    // Buzzer on = 1 or off =0
void eclearlap(void);         // Clear lap data in EEPROM
void edisplap(void);          // display lap data
void edsp_all_lap(void);      // display all of lap data
void edsp_all_r_lap(void);
void edsp_all_both_lap(unsigned char);
void esetlapptr(void);              // Set lap data pointer for latest
void ehelp(void);                   // Help

void pick_data_of_lap(unsigned char);
void uschar2dec(unsigned char); // 1byte change to decimal ascii
void show_ptr(lap_ptr *);     // Show EEPROM ringbuffer pointer information

/*  -----< Function Prototype (Extern) >-----------------------------------*/
extern void xerrcmd(void);   // Command Error
extern char *eqstrf( char * , char * );
extern void formater4timer_dbl(unsigned long, char *, char);
extern void xedumpb( void );
extern unsigned long ogetdata( unsigned long );
extern void cmdsrch( void );

/* -----< RAM assign >-----------------------------------------------------*/
one_lap  tmp_eep[4];          // temporary data buffer in RAM
                                          // (0= SW), 1= IR, (2= MG), 3= read from EEPROM
unsigned int lap_bf_cnt;      // pointer counter for above lap_data_buf[]
static lap_ptr  tmp_eep_ptr;// temporary pointer buffer in RAM
static union {                      // ready status for Switch, InfraRed and Magnet

       unsigned char BYTE;                      /*  Byte Access */
              struct {                                 /*  Bit  Access */
                     unsigned char SW:1;         /*  Bit 7       */
```

```
                    unsigned char IR:1;         /*  Bit 6       */
                    unsigned char MG:1;         /*  Bit 5       */
                    unsigned char NON:5; /*  Bit 4 to 0  */
        }       BIT;
} lap_data_status;
unsigned char id_save;              // number of Lapdata
static union {                      // ready status for Switch, InfraRed and Magnet

        unsigned char BYTE;                         /*  Byte Access */
                struct {                                    /*  Bit  Access */
                    unsigned char TGLPR:1;      /*  Bit 7       */
                    unsigned char TGLPW:1;      /*  Bit 6       */
                    unsigned char MILPR:1;      /*  Bit 5       */
                    unsigned char MILPW:1;      /*  Bit 4       */
                    unsigned char TMPOR:1;      /*  Bit 3       */
                    unsigned char BZFGR:1;      /*  Bit 2       */
                    unsigned char BZFGW:1;      /*  Bit 1       */
                    unsigned char OSCCR:1;      /*  Bit 0       */
        }       BIT;
} eep_sys_rw;
unsigned char   Device_id;
unsigned char   eep_clr_flg;

/* ----< RAM assign (External data) >---------------------------------- */
// lap time
extern unsigned int  latest_dt;     // which tool is used for latest data
extern unsigned long lap_ir; // Lap timer by using InfraRed sensor input
extern one_lap lap_for_led;         // Latest lap dat for LCD display
extern one_lap  lap_data_buf[];     // lap data buffer from EEPROM
extern unsigned char lap_ptr_buf_top;       // pointer for lap_data_buf[] top data
extern unsigned char eepdt_md;      // transfer complete flag, EEPROM to lap_data_buf[]
extern unsigned char eepdt_flg;     // transfer complete flag, EEPROM to lap_data_buf[]
extern unsigned long lap_best;      // best lap data
extern unsigned char bz_onoff_flg;// Buzzer On/Off control flag
extern char osc_cmp;                // Xtal freq. offset

// monitor
extern char line[BFSZ], *lp;
extern char timbuf[20];
extern unsigned long datap;
extern unsigned int  tim_usr2;      /* user timer no.2 */

/* ----< Constant >---------------------------------------------------- */
/*                           0        10        20        30        40        50 */
/*                           01234567890123456789012345678901234567890123456789 */
char *const mntrmsg0        = "    ------ Lap data control  [?=H(ret)]   ------";
//static char *const exitmsg = " Return to command mode?    'Y'(ret) or 'N'";
static char *const lap_msg   = " No.  ID   By  Laptime";
static char *const lpc_msg   = "      -- Continue? 'Y'(ret) or 'N'";
static char *const clr_msg0  = "Are your sure to clear EEPROM? 'Y'(ret) or 'N'";
static char *const clr_msg1  = "Cleared!";
static char *const clr_msg2  = "Canceled";
static char *const helpmsgB  = "   B       Buzzer on = 1 or off =0";
static char *const helpmsgC  = "   C       Clear Lap data in EEPROM";
static char *const helpmsgD  = "   D       Display lap data (new to old)";
static char *const helpmsgDR= "   DR      Display lap data (old to new)";
static char *const helpmsgE  = "   E       Dump EEPROM contents";
static char *const helpmsgM  = "   M       Go to H8 Monitor";
```

```
static char *const helpmsgH  = "   H       You know this";
/*                          012345678901234567890123456789012345678901234567890 */

/*  -----< Control program >------------------------------------------------ */
////////////////////////////////////////////////////////////////////////////
//
//     Data control routine for EEPROM Read/Write for Lapdata
//
////////////////////////////////////////////////////////////////////////////

// EEPROM Read/Write analysis
void IdleEEP( void )
{
       unsigned long dt;
       unsigned int  addr;
       unsigned char i;

       if (lap_data_status.BYTE){
              if (lap_data_status.BIT.IR == 1){
                     tmp_eep[IR_LAP].laptim = lap_ir;    // save lap time into temp. buff
                     tmp_eep[IR_LAP].msr_mode = IR_LAP;  // save masured tool
                     tmp_eep[IR_LAP].id = ++id_save;            // save new(+1) id
                     ring_put_data(&tmp_eep_ptr, &tmp_eep[IR_LAP]);
                     lap_data_status.BIT.IR = 0;
                     lap_for_led = tmp_eep[IR_LAP];
                     latest_dt = 0x80 + IR_LAP;
                     return;
              }
       }
       if (eepdt_flg == REQEST){
              pick_data_of_lap(eepdt_md);                        // pick up measured tool(SW,IR,MG)
              eepdt_flg = READY;                                       // Ready to read data in buffer
       }
       if (eep_sys_rw.BYTE != 0){
              if (eep_sys_rw.BIT.TGLPR == 1){
                     rd_eep_trg();
              } else if (eep_sys_rw.BIT.TGLPW == 1){
                     eep_sys_rw.BIT.TGLPW = 0;
                     addr = EEP_BSTLAP;
                     dt = lap_best;
                     i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 16777216));
                     i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 65536));
                     i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
                     i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt));
              } else if (eep_sys_rw.BIT.BZFGR == 1){
                     rd_eep_bz();
              } else if (eep_sys_rw.BIT.BZFGW == 1){
                     eep_sys_rw.BIT.BZFGW = 0;
                     addr = EEP_BZONOF;
                     i = Write_byte_EEPROM((unsigned short)addr, bz_onoff_flg);
              }
       }
       if (eep_clr_flg == 1){
              eep_clr_flg = 0;
              if (ring_is_empty(&tmp_eep_ptr)){   // check buffer empty
                     ;
              } else {                            // not empty
                     ring_clear(&tmp_eep_ptr);
```

```
            }
        }
}

void init_eep_lap(void)
{
        // EEPROM I/F
        Init_eeprom();                  // Initialize EEPROM with port on/off mode
        Init_iic2();                    // Initialize IIC2 I/F
        Device_id = 0x00;               // Set slave address of EEPROM: 0 (A2,A1,A0)
        // Clear buffer
        tmp_eep[0].laptim = 0;
        tmp_eep[0].msr_mode = 0;    // save masured tool
        tmp_eep[0].id = 0;              // save new(+1) id
        tmp_eep[1].laptim = 0;          // save lap time into temp. buff
        tmp_eep[1].msr_mode = 0;    // save masured tool
        tmp_eep[1].id = 0;              // save new(+1) id
        tmp_eep[2].laptim = 0;          // save lap time into temp. buff
        tmp_eep[2].msr_mode = 0;    // save masured tool
        tmp_eep[2].id = 0;              // save new(+1) id
        tmp_eep[3].laptim = 0;          // save lap time into temp. buff
        tmp_eep[3].msr_mode = 0;    // save masured tool
        tmp_eep[3].id = 0;              // save new(+1) id
        lap_data_buf[0].laptim = TIM_EMPTY;
        lap_data_buf[0].msr_mode = NOTDATA;
        lap_data_buf[0].id = NOTDATA;
        eep_sys_rw.BYTE = 0;
        eep_clr_flg = 0;
        lap_data_status.BYTE = 0;
        ring_init(&tmp_eep_ptr);
        if (ring_rd_ptr_top(&tmp_eep_ptr)){
            id_save = 0;                // if EEPROM is empty, clear id
            return;
        }
        ring_get_data(&tmp_eep_ptr, &tmp_eep[RD_LAP]);
        id_save = tmp_eep[RD_LAP].id;// read id and save it into RAM
        // Data read from EEPROM move to task.c general initialize routine
        //      rd_eep_trg();           // read best lap data from EEPROM
        //      rd_eep_bz();            // read buzzer flag from EEPROM
}

void rd_eep_trg(void)
{
        unsigned long dt;
        unsigned int  addr;

        eep_sys_rw.BIT.TGLPR = 0;
        addr = EEP_BSTLAP;
        dt  = (unsigned long)Read_byte_EEPROM((unsigned short)addr++) * 16777216;
        dt += (unsigned long)Read_byte_EEPROM((unsigned short)addr++) * 65536;
        dt += (unsigned long)Read_byte_EEPROM((unsigned short)addr++) * 256;
        dt += (unsigned long)Read_byte_EEPROM((unsigned short)addr++);
        lap_best = dt & TIM_MAX_TR_MASK;
}

void rd_eep_osc(void)
{
        eep_sys_rw.BIT.OSCCR = 0;
```

```
	osc_cmp = (char)Read_byte_EEPROM((unsigned short)EEP_CLKCMP);
}

void rd_eep_bz(void)
{
	eep_sys_rw.BIT.BZFGR = 0;
	bz_onoff_flg  = (unsigned char)Read_byte_EEPROM((unsigned short)EEP_BZONOF);
}

void ready_sw_data(void)
{
	lap_data_status.BIT.SW = 1;
}

void ready_ir_data(void)
{
	lap_data_status.BIT.IR = 1;
}

void ready_mg_data(void)
{
	lap_data_status.BIT.MG = 1;
}

void req_lapdata_clear(void)
{
	eep_clr_flg = 1;
}

void req_rd_tglap(void)
{
	eep_sys_rw.BIT.TGLPR = 1;
}

void req_wr_tglap(void)
{
	eep_sys_rw.BIT.TGLPW = 1;
}

void req_rd_milap(void)
{
	eep_sys_rw.BIT.MILPR = 1;
}

void req_wr_milap(void)
{
	eep_sys_rw.BIT.MILPW = 1;
}

void req_rd_bzfg(void)
{
	eep_sys_rw.BIT.BZFGR = 1;
}

void req_wr_bzfg(void)
{
	eep_sys_rw.BIT.BZFGW = 1;
}
```

```
void req_rd_osccmp(void)
{
	eep_sys_rw.BIT.OSCCR = 1;
}

void pick_data_of_lap(unsigned char md)
{
	unsigned char i, skip;

	if (ring_is_empty(&tmp_eep_ptr)){   // check buffer empty
		lap_data_buf[0].laptim = TIM_EMPTY;
		lap_data_buf[0].msr_mode = NOTDATA;
		lap_data_buf[0].id = NOTDATA;
		return;
	}
	ring_rd_ptr_top(&tmp_eep_ptr);
	skip = lap_ptr_buf_top;
	if (skip){
		for (i = 0; i < skip;){
			ring_get_data(&tmp_eep_ptr, &tmp_eep[RD_LAP]);
			if (tmp_eep[RD_LAP].msr_mode == md){
				i++;
			}
			if (ring_rd_ptr_nxt(&tmp_eep_ptr)){
				lap_data_buf[0].laptim = TIM_EMPTY;
				lap_data_buf[0].msr_mode = NOTDATA;
				lap_data_buf[0].id = NOTDATA;
				return;
			}
		}
	}
	for (i = 0; i < SCRNSIZ;){
		ring_get_data(&tmp_eep_ptr, &tmp_eep[RD_LAP]);
		if (tmp_eep[RD_LAP].msr_mode == md){
			lap_data_buf[i] = tmp_eep[RD_LAP];
			i++;
		}
		if (ring_rd_ptr_nxt(&tmp_eep_ptr)){
			lap_data_buf[i].laptim = TIM_EMPTY;
			lap_data_buf[i].msr_mode = NOTDATA;
			lap_data_buf[i].id = NOTDATA;
			return;
		}
	}
}

////////////////////////////////////////////////////////////////////////
//     Part of monitor function for EEPROM control
////////////////////////////////////////////////////////////////////////
/*  -----Command Table -----*/
typedef const struct{
	char   *cmd;
	void   (*func)( void );
}TBLENTRY;

TBLENTRY ecmdtbl[] = {
	{"B",ebzr},                          // Buzzer on = 1 or off =0
```

```
        {"C",eclearlap},            // Clear lap data in EEPROM
//      {"DN",edisplap},            // Display one lap data
        {"DR",edsp_all_r_lap},      // Display all lap data
        {"D",edsp_all_lap},         // Display all lap data
        {"E",xedumpb},              // Dump EEPROM
        {"M",cmdsrch},              // H8 Monitor
//      {"SP",esetlapptr},          // Set lap data pointer for latest
        {"H",ehelp},                // Help
        {"\0",xerrcmd},                     // Command Error */
};

/*  -----Help -------  */
void ehelp( void )
{
        SciPutS( helpmsgB );  PutCRLF();
        SciPutS( helpmsgC );  PutCRLF();
        SciPutS( helpmsgD );  PutCRLF();
        SciPutS( helpmsgDR);  PutCRLF();
        SciPutS( helpmsgE );  PutCRLF();
        SciPutS( helpmsgM );  PutCRLF();
        SciPutS( helpmsgH );  PutCRLF();
/*
        SciPutS(" B      Buzzer on = 1 or off =0");                  PutCRLF();
        SciPutS(" C      Clear Lap data in EEPROM");            PutCRLF();
        SciPutS(" D      Display lap data (new to old)"); PutCRLF();
        SciPutS(" DR     Display lap data (old to new)"); PutCRLF();
//      SciPutS(" DN     Display newest");                           PutCRLF();
        SciPutS(" E      Dump EEPROM contents");                PutCRLF();
        SciPutS(" M      Go to H8 Monitor");                         PutCRLF();
//      SciPutS(" SP     Set lapdata pointer for latest");      PutCRLF();
        SciPutS(" H      You know this");                            PutCRLF();
//      SciPutS(" [RET]  Return to called routine");            PutCRLF();
*/
}

/*  -----Command analysis and excute -----  */
void lap_cmd_srch( void )
{
        char *c;
        TBLENTRY *p;

        SciPutS( mntrmsg0 );
        for(;;){
                PutCRLF();
                SciPutC('>');
                getline();
/*
                if(line[0] == '\0'){
                        SciPutS( exitmsg );
                        getline();

                        if((line[0] == '\0') || (line[0] == 'Y')){
                                PutCRLF();
                                opning_msg();
                                return;
                        } else {
                                continue;
                        }
```

```
                }
*/
                for(p = ecmdtbl; *p  -> cmd; p++){
                        if((c = eqstrf(p  -> cmd,lp)) != 0){
                                break;
                        }
                }
                lp = c;
                (*p  -> func)();
        }
}

/*  -----Buzzer on = 1 or off =0  -------*/
void ebzr()
{
        skpspc();
        if (ogetdata( datap ) == 0){
                SciPutS(" Buzzer OFF");      PutCRLF();
                bz_onoff_flg = OFF;
        } else {
                SciPutS(" Buzzer ON");       PutCRLF();
                bz_onoff_flg = ON;
                tim_usr2 = T_500MS;                 // for Buzzer ON longer time!!
        }
        req_wr_bzfg();
}

/*  -----Clear lap data in EEPROM  -------*/
void eclearlap()
{
        SciPutS( clr_msg0 );
        getline();

        if((line[0] == '\0')  || (line[0] == 'Y')){
                SciPutS( clr_msg1 );
                ring_clear(&tmp_eep_ptr);
                show_ptr(&tmp_eep_ptr);
                return;
        } else {
                SciPutS( clr_msg2 );
        }
}

/*  -----Show pointer data  -------*/
void show_ptr(lap_ptr *ptr)
{
        unsigned int dt;

        SciPutS(" EEPROM ring buffer pointer (HEX)");     PutCRLF();
        dt = (unsigned int)ptr->buf_top;
        SciPutS(" Buffer top = ");
        puthxw(dt);
        dt = (unsigned int)ptr->head;
        SciPutS(", head = ");
        puthxw(dt);
        dt = (unsigned int)ptr->tail;
        SciPutS(", tail = ");
```

```
        puthxw(dt);
        dt = (unsigned int)ptr->size;
        SciPutS(", size = ");
        puthxw(dt);
        PutCRLF();
}

/* ------Show Lap data -------*/
void edisplap()
{
        if (ring_is_empty(&tmp_eep_ptr)){   // check buffer empty
                SciPutS(" EEPROM is empty "); PutCRLF();
                return;
        }
        ring_get_data(&tmp_eep_ptr, &tmp_eep[RD_LAP]);
        SciPutS(" Lap time = ");
        (tmp_eep[RD_LAP].laptim, timbuf, 1);
        SciPutS(timbuf);
        SciPutS(", measured by ");
        if (tmp_eep[RD_LAP].msr_mode == SW_LAP){
                SciPutS("SW");
        } else if (tmp_eep[RD_LAP].msr_mode == IR_LAP){
                SciPutS("IR");
        } else if (tmp_eep[RD_LAP].msr_mode == MG_LAP){
                SciPutS("MG");
        } else {
                SciPutS("??");
        }
        SciPutS(", ID =");
        puthxb(tmp_eep[RD_LAP].id);
        PutCRLF();
}

/* ------Show All of Lap data -------*/
void edsp_all_lap()
{
        edsp_all_both_lap(1);
}

void edsp_all_r_lap()
{
        edsp_all_both_lap(0);
}

void edsp_all_both_lap(unsigned char flg)
{
        unsigned char n, i;

        n = 1;
        SciPutS(lap_msg);  PutCRLF();
        if (ring_is_empty(&tmp_eep_ptr)){   // check buffer empty
                SciPutS(" EEPROM is empty "); PutCRLF();
                return;
        }
        if (flg){
                ring_rd_ptr_top(&tmp_eep_ptr);
        } else {
                ring_rd_ptr_end(&tmp_eep_ptr);
```

```
	}
	do{
		for (i = 0; i < 10; i++){
			ring_get_data(&tmp_eep_ptr, &tmp_eep[RD_LAP]);
			SciPutS(" ");
			uschar2dec(n++);
			SciPutS("  ");
			uschar2dec(tmp_eep[RD_LAP].id);
			SciPutS("  ");
			if (tmp_eep[RD_LAP].msr_mode == SW_LAP){
				SciPutS("SW");
			} else if (tmp_eep[RD_LAP].msr_mode == IR_LAP){
				SciPutS("IR");
			} else if (tmp_eep[RD_LAP].msr_mode == MG_LAP){
				SciPutS("MG");
			} else {
				SciPutS("??");
			}
			SciPutS("  ");
			formater4timer_dbl(tmp_eep[RD_LAP].laptim, timbuf, 1);
			SciPutS(timbuf);
			PutCRLF();
			if (flg){
				if (ring_rd_ptr_nxt(&tmp_eep_ptr)){
					SciPutS(" reach to end! no more lap data in EEPROM "); PutCRLF();
					return;
				}
			} else {
				if (ring_rd_ptr_nxt_r(&tmp_eep_ptr)){
					SciPutS(" reach to top! no more lap data in EEPROM "); PutCRLF();
					return;
				}
			}
		}
		SciPutS(lap_msg);
		SciPutS(lpc_msg);
		getline();
	} while((line[0] == '\0') || (line[0] == 'y') || (line[0] == 'Y'));
}

/* -----Set poiner for latest -------*/
void esetlapptr()
{
	ring_rd_ptr_top(&tmp_eep_ptr);
	show_ptr(&tmp_eep_ptr);
}

/* -----1 byte to Decimal -------*/
void uschar2dec(unsigned char dt)
{
	SciPutC((char)(dt  / 100) + '0');
	dt = dt  - ((dt / 100) * 100);
	SciPutC((char)(dt  / 10) + '0');
	dt = dt  - ((dt / 10) * 10);
	SciPutC((char)(dt + '0'));
}

//////////////////////////////////////////////////////////////////////////
```

```
//
//      subroutine for EEPROM control
//
////////////////////////////////////////////////////////////////////////////
// clear EEPROM buffer
int ring_clear (lap_ptr *ring)
{
        ring->buf_top = EEP_LAP_TOP;
        ring->size = LAP_DT_SZ;
        ring->head = ring->tail = ring->rd_ptr = ring->buf_top;
        id_save = 0;                                // if EEPROM is empty, clear id
        return(ring_ptr_save(ring));
}

// pointer initialize for next power on
int ring_init(lap_ptr *ring)
{
        return(ring_ptr_load(ring)); // data pointer read from EEPROM
}

// pointer initialize for data read
int ring_rd_ptr_top(lap_ptr *ring)
{
        ring->rd_ptr = ring->tail;          // set latest pointer
        if (ring_rd_ptr_nxt(ring)){         // if empty, return with error
                return -1;
        }
        return 0;                                           // normal end
}

int ring_rd_ptr_end(lap_ptr *ring)
{
        ring->rd_ptr = ring->head;          // set latest pointer
        return 0;                                           // normal end
}

// pointer change to next read
int ring_rd_ptr_nxt(lap_ptr *ring)
{
        unsigned int i;

        if (ring->rd_ptr == ring->head){
                return -1;                                  // if current pointer is bottom(older) then
return w/ error
        }
        i = (unsigned int)ring->rd_ptr - SIZ;
        // check next pointer
        if ( (i < (unsigned int)ring->buf_top) || (i > (unsigned int)ring->buf_top + ring->size)){
                ring->rd_ptr = (void *)((unsigned int)ring->buf_top + ring->size -SIZ);
        } else {
                ring->rd_ptr = (void *)i;
        }
        return 0;
}

int ring_rd_ptr_nxt_r(lap_ptr *ring)
{
        unsigned int i;
```

```
        i = (unsigned int)ring->tail  - SIZ;
        if (i == (unsigned int)ring->rd_ptr){
                return  -1;                                     // if current pointer is bottom(older) then
return w/ error
        }
        i = (unsigned int)ring->rd_ptr + SIZ;
        // check next pointer
        if ( i == (unsigned int)ring->buf_top + ring->size){
                ring->rd_ptr = ring->buf_top;
        } else {
                ring->rd_ptr = (void *)i;
        }
        return 0;
}

// check empity condition
int ring_is_empty(lap_ptr *ring)
{
        if (ring->head == ring->tail){
                return 1;
        }
        return 0;
}

// check read pointer
int ring_is_end(lap_ptr *ring)
{
        if (ring->head == ring->rd_ptr){
                return 1;
        }
        return 0;
}

// write data into the buffer
int ring_put_data(lap_ptr *ring, one_lap *data)
{
        unsigned int pt0,pt1;

        ring_data_save(ring, data);
        // set next pointer
        pt0 = (unsigned int)ring->tail + SIZ;
        pt1 = (unsigned int)ring->buf_top + ring->size;
        if (pt0 == pt1){                                // if reach EEPROM ring buffer end, then set to top
                ring->tail = ring->buf_top;
        } else {                                                // not reach buf. end yet
                ring->tail = (void *)pt0;
        }
        // if buffer is reached full, then modify head pointer
        if ( ring->tail == ring->head){     // full condition then action
                pt0 = (unsigned int)ring->head + SIZ;
//              pt1 = (unsigned int)ring->buf_top + ring->size; // calculated by above
                if (pt0 == pt1){                        // if reach EEPROM ring buffer end, then set to top
                        ring->head = ring->buf_top;
                } else {                                        // not reach buf. end yet
                        ring->head = (void *)pt0;
                }
        }
```

```
	ring_ptr_save(ring);
	return 0;
}

// read data from the buffer
int ring_get_data(lap_ptr *ring, one_lap *data)
{
	ring_data_load(ring, data);
	return 0;
}

// check how many data in the buffer
int ring_get_capacity(lap_ptr *ring)
{
	if (ring->head < ring->tail) {
		return ((unsigned int)ring->tail  - (unsigned int)ring->head);
	}
	return (unsigned int)ring->size;
}

// pointer save action into EEPROM
int ring_ptr_save(lap_ptr *ring)
{
	unsigned int addr, dt;
	unsigned char i;

	// addr
	addr = EEP_PTR;
	// buffer top addr
	dt = (unsigned int)ring->buf_top;
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
//	if (error_status == ERROR){
//		;
//	}
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)dt );
	// data head(older)
	dt = (unsigned int)ring->head;
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)dt );
	// data tail(latest)
	dt = (unsigned int)ring->tail;
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)dt );
	// data pointer for data read
	dt = (unsigned int)ring->rd_ptr;
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)dt );
	// EEPROM ring buffer size
	dt = ring->size;
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));
	i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)dt );
	return 0;
}

// pointer read action from EEPROM
int ring_ptr_load(lap_ptr *ring)
{
	unsigned int addr;
```

```
        unsigned char i, dt, dt0;

        // addr
        addr = EEP_PTR;
        // buffer top addr
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
//      if (error_status == ERROR){
//              ;
//      }
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        ring->buf_top = (void *)((unsigned int)dt0 * 256 + (unsigned int)dt);
        // data head(older)
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        ring->head    = (void *)((unsigned int)dt0 * 256 + (unsigned int)dt);
        // data tail(latest)
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        ring->tail    = (void *)((unsigned int)dt0 * 256 + (unsigned int)dt);
        // data pointer for data read
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        ring->rd_ptr  = (void *)((unsigned int)dt0 * 256 + (unsigned int)dt);
        // EEPROM ring buffer size
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        ring->size    = (unsigned int)dt0 * 256 + (unsigned int)dt;
        return 0;
}

// data write action into EEPROM
int ring_data_save(lap_ptr *ring, one_lap *data)
{
        unsigned long dt;
        unsigned int addr;
        unsigned char i;

        // addr
        addr = (unsigned int)ring->tail;
        // lap time
        dt = data->laptim;
        i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 16777216));// 2^24
//      if (error_status == ERROR){
//              ;
//      }
        i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 65536));    // 2^16
        i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt / 256));      // 2^8
        i = Write_byte_EEPROM((unsigned short)addr++, (unsigned char)(dt ));           // 2^0
        // measured tool
        i = Write_byte_EEPROM((unsigned short)addr++, data->msr_mode);
        // id number
        i = Write_byte_EEPROM((unsigned short)addr++, data->id);
        return 0;
}

// data read action from EEPROM
int ring_data_load(lap_ptr *ring, one_lap *data)
{
```

```
        unsigned int addr;
        unsigned char i, dt,dt0,dt1,dt2;

        // addr
        addr = (unsigned int)ring->rd_ptr;
        // lat time
        dt2 = Read_byte_EEPROM((unsigned short)addr++);
        dt1 = Read_byte_EEPROM((unsigned short)addr++);
        dt0 = Read_byte_EEPROM((unsigned short)addr++);
        dt  = Read_byte_EEPROM((unsigned short)addr++);
        data->laptim = (unsigned long)dt2 * 16777216 + (unsigned long)dt1 * 65536
                                    + (unsigned long)dt0 * 256 + (unsigned long)dt;
        // measured tool
        data->msr_mode = Read_byte_EEPROM((unsigned short)addr++);
        // id number
        data->id = Read_byte_EEPROM((unsigned short)addr++);
        return 0;
}
```