```c
/*************************************************************************
        Event and Cyclic Task's for
                Lap-Timer project using RTM_H8/3664F

                May      25,1998        New version on H8/3048F
                November  4,2001        Start 3664F project
                December 23,2001
                July     11,2004        Use RENESAS HEW functions
                August   12,2004        confirmed basic function then add new function
                                                likes Dataset change, Lapdata display and so on
                November  6,2004        Start with New PCB
                November 14,2004        Create Data entry mode
                November 16,2004        latest modification

        Copyright (C) 1998, 2001,'04 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.
*************************************************************************/
/*      #pragma interrupt ed0_ISR */

/*  ----< Include Files >------------------------------------------------- */
#include        "3664f.h"
#include        "rtm_H8_3664.h"
#include        "sci.h"
#include        "const.h"
#include        "lcd.h"
#include        "ringbuf.h"
#include        "task.h"
#include        <setjmp.h>
#include        <machine.h>             // Use RENESAS HEW functions

/*  ----< Function Prototype >--------------------------------------------- */
void main( void );
void abort(void);

void ed0_main( void );
void ed1_main( void );
void ed2_main( void );
void ed3_main( void );
void cy0_init( void );
void cy0_main( void );
void cy1_init( void );
void cy1_main( void );
void cy2_init( void );
void cy2_main( void );
void cy3_init( void );
void cy3_main( void );
void cy4_init( void );
void cy4_main( void );
void cy5_init( void );
void cy5_main( void );
void cy6_init( void );
void cy6_main( void );
void cy7_init( void );
void cy7_main( void );
void general_init( void );
void port_init ( void );
void IdleNop( void );
void help_msg( void );
void PutCRLF( void );
void opning_msg( void );

extern void cmdsrch( void );

//void sw_on_action(void);
void swich_status_check(unsigned char);
void swchk_normal(void);
void swchk_mode_select(void);
void swchk_lap_display(void);
void swchk_lap_sw_disp(void);
void swchk_lap_ir_disp(void);
void swchk_lap_mg_disp(void);
void sw_lap_scrol(unsigned char);
void swchk_data_entry(void);
void swchk_dt_lapmin_ent(void);
void swchk_dt_mgnof_ent(void);
void swchk_dt_bzoff_ent(void);
void swchk_dt_stgtm_ent(void);
void swchk_dt_mclr_ent(void);
void swchk_gps_inf(void);

void swchk_md(unsigned char);
void swchk_s(char);
void swchk_i();

void data_modify( void );
void bf_zero_chk( void );
unsigned long bf_cnvrt_to_base_time( void );
unsigned long enter_sw_chk( void );

void eeprom_rd_req(unsigned char);

void lcd_mode_select(void);
void lcd_lap_display(void);
void lcd_lap_sw_disp(void);
void lcd_lap_ir_disp(void);
void lcd_lap_mg_disp(void);
void lcd_on_lap(void);
void lcd_data_entry(void);
void lcd_dt_lapmin_ent(void);
void lcd_dt_mgnof_ent(void);
void lcd_dt_bzoff_ent(void);
void lcd_dt_stgtm_ent(void);
void lcd_dt_mclr_ent(void);
void lcd_dt_cmn_ent(void);

void lcd_gps_inf(void);
void error_screen(void);

void formater4timer(unsigned long, char *, char);
void conv_id(unsigned char, char *);
void lcd_updating_action(void);
void lcd_initial_screen(void);

void sw_status(void);
void act0(void);
void act1(void);
void act2(void);
void act4(void);
void act8(void);
void act9(void);

void test_mode();

/*  ----< Function Prototype (Extern) >----------------------------------*/
extern void irq_enable( void );
extern void irq_disable( void );
//void SciPutC(char);
//void SciPutS(char *);
//char SciGetC(short);

extern void lap_cmd_srch( void );

extern char *eqstrf( char * , char * );

extern void reset_timerV(void);
extern void reset_timerW(void);
extern void eclearlap(void);        // Clear lap data in EEPROM

/*  ----< Constant data in ROM >------------------------------------------*/
static char *const copyrigh =
"Arai,Kenji/JH1PJL(c)2004 kenjia@sannet.ne.jp <2nd ver. w/ PCB Nov.,'04>";

/*      Cyclic Task Period                   */
const unsigned char cyclic_period[8] = {
        T_20MS,        /* cyclic task no.0 */ // Swiches
        T_10MS,        /* cyclic task no.1 */ // Check operation mode
        T_100MS,       /* cyclic task no.2 */ // LCD
        T_500MS,       /* cyclic task no.3 */ // NOT USE
        T_500MS,       /* cyclic task no.4 */ // NOT USE
        T_500MS,       /* cyclic task no.5 */ // NOT USE
        T_500MS,       /* cyclic task no.6 */ // NOT USE
        T_500MS        /* cyclic task no.7 */ // NOT USE
};

// ONLY FOR POWER ON INITIALIZE
const unsigned char cyclic_init_period[8] = {
        T_500MS,              /* cyclic task no.0 */
        T_500MS+T_10MS,       /* cyclic task no.1 */
        T_500MS+T_50MS,       /* cyclic task no.2 */
        T_500MS+T_50MS,       /* cyclic task no.3 */
        T_200MS,              /* cyclic task no.4 */
        T_200MS,              /* cyclic task no.5 */
        T_500MS,              /* cyclic task no.6 */
        T_500MS               /* cyclic task no.7 */
};

//      LCD Panel Layout
//                10        20        30        40
static char *const init_scrn_msg0 =
//       0123456789012345678901234567890123456789
        "        IR   --: --. -- + --: --. --   --: --. --";
static char *const init_scrn_msg1 =
//       0123456789012345678901234567890123456789
        "TR --: --. --   --: --. --SW            --: --. --";
static char *const mode_slct_msg0 =
//       0123456789012345678901234567890123456789
        " Mode  Lap -check GPS Data -entry    HOME ";
static char *const mode_slct_msg1 =
//       0123456789012345678901234567890123456789
        "Select   [ ]     [ ]    [ ]         [ ] ";
static char *const lapdata_msg0 =
//       0123456789012345678901234567890123456789
        "Lap data    SW      IR      MG     HOME ";
static char *const lapdata_msg1 =
//       0123456789012345678901234567890123456789
        "Select      [ ]     [ ]     [ ]    [ ]  ";
static char *const lapdata_msg2 =
//       0123456789012345678901234567890123456789
        "Lap by XX      --: --. --  --: --. --  --: --. --";
static char *const lapdata_msg3 =
//       0123456789012345678901234567890123456789
        "Bst  --: --. --   --: --. --  --: --. --  --: --. --";
static char *const lapdata_msg4 =
//       0123456789012345678901234567890123456789
        "        XXX= --: --. --        xxx= --: --. --";
static char *const dtentr_msg0 =
//       0123456789012345678901234567890123456789
        "Set data Target -Lap Min. -Lap MGxN  HOME ";
static char *const dtentr_msg1 =
//       0123456789012345678901234567890123456789
        "Select     [ ]       [ ]     [ ]   [ ] ->";
static char *const dtentr_msg2 =
//       0123456789012345678901234567890123456789
        "Set data  Mem -Clr   BZ -on/off      HOME ";
static char *const dtentr_msg3 =
//       0123456789012345678901234567890123456789
        "Select     [ ]         [ ]         [ ] ->";
static char *const dtentr_msg4 =
//       0123456789012345678901234567890123456789
        "Data -in by SHFT,INCR                   ";
static char *const dtentr_msg5 =
//       0123456789012345678901234567890123456789
        "     --: --. --        ENTR or CANCEL(MODE)";
static char *const dtent_clr_msg0 =
//       0123456789012345678901234567890123456789
        "Clear all Lap data     Cancel = MODE    ";
static char *const dtent_clr_msg1 =
//       0123456789012345678901234567890123456789
        " CANNOT RECOVER!!    or Clear = ENTR    ";
static char *const dtent_clr_msg2 =
//       0123456789012345678901234567890123456789
        "Clear all Lap data                      ";
static char *const dtent_clr_msg3 =
//       0123456789012345678901234567890123456789
        "Are you sure?         YES=ENTR  NO=MODE ";
static char *const dtent_clr_msg4 =
//       0123456789012345678901234567890123456789
```

```
	"Clear all Lap data    Cleared           ";
static char *const dtent_clr_msg5 =
//	 0123456789012345678901234567890123456789
	"                    all of Lap data!!   ";
static char *const dtent_bzr_msg0 =
//	 0123456789012345678901234567890123456789
	"Buzzer On/Off       MODE=keep(no change)";
static char *const dtent_bzr_msg1 =
//	 0123456789012345678901234567890123456789
	"  control            ENTR = change      ";
static char *const err_msg0 =
//	 0123456789012345678901234567890123456789
	"Not impliment yet                       ";
static char *const err_msg1 =
//	 0123456789012345678901234567890123456789
	"Please push MODE key                    ";

/* -----< RAM assign >------------------------------------------------------- */
unsigned long datap;
char rxBuf[20];					// RS232 receive char
char Buf;
char line[BFSZ], *lp;
jmp_buf jb_error;

// lap time
one_lap lap_for_lcd;		// Latest lap dat for LCD display
unsigned int latest_dt;			// which tool is used for latest data
unsigned long lap_sw;		// Lap timer by using manual switch input
unsigned long lap_mg;		// Lap timer by using magnetic sensor input
unsigned long lap_ir;		// Lap timer by using InfraRed sensor input
unsigned long lap_start;	// lap start time (from timer base)
unsigned long lap_best;		// best lap data
unsigned long lap_min;			// Minimum lap data for best lap data
unsigned long lap_latest;	// latest data current
unsigned long lap_last1;	// latest data -1
unsigned long lap_last2;	// latest data -2
static union {				// power on then clear, if each bit = 0 the 1st

	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_1st_one;
char timbuf[20];
one_lap  lap_data_buf[BZ4LAP];		// lap data buffer from EEPROM
unsigned char lap_ptr_buf_top;		// pointer for lap_data_buf[] top data
unsigned char eepdt_md;				// transfer complete flag, EEPROM to lap_data_buf[]
unsigned char eepdt_flg;		// transfer complete flag, EEPROM to lap_data_buf[]
unsigned char tmp_datain[6]; // work area for data manuplation

// LCD
unsigned char lcd_update;	// flag for LCD updating
unsigned char lcd_updating; // under updating
static union {
	unsigned char BYTE;
	struct {
			unsigned char modify:1;	//     Bit 7
			unsigned char status:1;	//  Bit 6
			unsigned char actual:1;	//  Bit 5
			unsigned char dirctn:1;	//  Bit 4
			unsigned char cunter:4;	//  Bit 3-0
	} BIT;
} sw_condition[4];

unsigned char mode;					// function and display mode
static char entry_no;				//
static char mode_step;					// Step number of each Mode
unsigned char bz_onoff_flg;			// Buzzer On/Off control flag
char entry_lcdbf[10];				// Temp. buf for data entry

/* ----< RAM assign (External data) >------------------------------------*/
extern unsigned long base_timer;	// base timer for measuring
extern int ir_state;				// InfraRed sensor detect condition
// capture data and base timer data(@ captured)
extern unsigned int capture_b_data; // InfraRed
extern unsigned int capture_c_data; // Magnetic
extern unsigned int capture_d_data; // Switch
extern unsigned long btimr_b;		// InfraRed
extern unsigned long btimr_c;		// Magnetic
extern unsigned long btimr_d;		// Switch

extern unsigned int tim_usr1,		// use for wait control,
				tim_usr2;		// timer1 is drived by RTM (2mS tick)
extern char lcdbuf[];				// data buffer for LCD display
extern char anthr_lcdbuf[];			// another buffer for diffrent screen mode
extern one_lap  tmp_eep[];			// temporary data buffer in RAM
								// 0= SW, 1= IR, 2= MG, 3= read from
EEPROM
/*********************************************************************************************
******************* Event driven task and Cyclic task *****************************************
*********************************************************************************************/

/*********************************************************************************************
	Event Driven Tasks
*********************************************************************************************/
/* ------------------------------------------------------------------------------------------
	ED0 Task  --- Lap time calculation by InfraRed sensor with TimerW interrupt
 -----------------------------------------------------------------------------------------*/
void ed0_main(void)
{
	static unsigned long time_ir_temp;

	if (tim_usr1 == 0){
		lap_ir = btimr_b  - time_ir_temp;
		time_ir_temp = lap_start = btimr_b;
		tim_usr1 = T_5S_16B;
		tim_usr2 = T_200MS;				// for Buzzer ON
		if(lap_1st_one.BIT.IR == 1){ // can save to EEPROM
			ready_ir_data();
		} else {							// this is first time from power on
			lap_1st_one.BIT.IR = 1;	// next data save to EEPROM
		}
	} else {
			if (ir_state == 0){
				tim_usr1 = T_5S_16B;
			}
		}
}

/* -----------------------------------------------------------------------------------------
		ED1 Task  --- Lap time calculation by Magnetic sensor with TimerW interrupt
 -----------------------------------------------------------------------------------------*/
void ed1_main(void)
{
		static unsigned long time_mg_temp;

		lap_mg = btimr_c  - time_mg_temp;
		time_mg_temp = lap_start = btimr_c;
		tim_usr2 = T_200MS;						// for Buzzer ON
		if(lap_1st_one.BIT.MG == 1){		// can save to EEPROM
			ready_mg_data();
		} else {								// this is first time from power on
			lap_1st_one.BIT.MG = 1;			// next data save to EEPROM
		}
}

/* -----------------------------------------------------------------------------------------
		ED2 Task  --- Lap time calculation by manual switch with TimerW interrupt
 -----------------------------------------------------------------------------------------*/
void ed2_main(void)
{
		static unsigned long time_sw_temp;

		lap_sw = btimr_d  - time_sw_temp;
		time_sw_temp = lap_start = btimr_d;
		tim_usr2 = T_200MS;						// for Buzzer ON
		if(lap_1st_one.BIT.SW == 1){		// can save to EEPROM
			ready_sw_data();
		} else {								// this is first time from power on
			lap_1st_one.BIT.SW = 1;			// next data save to EEPROM
		}
}

/* -----------------------------------------------------------------------------------------
		ED3 Task  --- Not use
 -----------------------------------------------------------------------------------------*/
void ed3_main(void) { ; }

/*******************************************************************************************
		Cyclic Tasks  !!! WARNING !!! CY0 & CY1 is opposite order for only description point of view
*******************************************************************************************/
/* -----------------------------------------------------------------------------------------
		CY1 Task  --- check operation mode & read switch status
 -----------------------------------------------------------------------------------------*/
void cy1_init( void )
{
		sw_condition[MODESW].BIT.cunter = 3;
		sw_condition[MODESW].BIT.dirctn = DR_ON;
		sw_condition[MODESW].BIT.status = SWOFF;
		sw_condition[MODESW].BIT.modify = SWOFF;
		sw_condition[ENTRSW].BYTE = sw_condition[MODESW].BYTE;
```

```
        sw_condition[SHFTSW].BYTE = sw_condition[MODESW].BYTE;
        sw_condition[INCRSW].BYTE = sw_condition[MODESW].BYTE;
}

void cy1_main( void )
{
        void *p;

        if (tim_usr2 == 0){
                BUZZER = OFF_DR_HS;             // Buzzer OFF
        } else {
                if (bz_onoff_flg == ON){
                        BUZZER = ON_DR_HS;      // Buzzer ON
                } else {
                        BUZZER = OFF_DR_HS;     // Buzzer OFF
                }
        }
        swich_status_check(MODESW);
        swich_status_check(ENTRSW);
        swich_status_check(SHFTSW);
        swich_status_check(INCRSW);
}

void swich_status_check(unsigned char swno)
{
        unsigned char i;

        i = SWOFF;
        switch (swno){
                case MODESW:{
                        if (SW_MODE == SWON){
                                i = SWON;
                        }
                        break;
                }
                case ENTRSW:{
                        if (SW_ENTR == SWON){
                                i = SWON;
                        }
                        break;
                }
                case SHFTSW:{
                        if (SW_SHFT == SWON){
                                i = SWON;
                        }
                        break;
                }
                case INCRSW:{
                        if (SW_INCR == SWON){
                                i = SWON;
                        }
                        break;
                }
                default:{
                        ;/*  Error */
                }
        }
        if (i == SWON){
```

```
		if (sw_condition[swno].BIT.status == SWOFF){
			if (sw_condition[swno].BIT.dirctn == DR_ON){
				sw_condition[swno].BIT.cunter  -= 1;
				if (sw_condition[swno].BIT.cunter == 0){
					sw_condition[swno].BIT.cunter = 3;
					sw_condition[swno].BIT.dirctn = DR_OFF;
					sw_condition[swno].BIT.status = SWON;
					sw_condition[swno].BIT.modify = SWON;
				}
			}
		} else {
			if (sw_condition[swno].BIT.dirctn == DR_OFF){
				sw_condition[swno].BIT.cunter = 3;
			}
		}
	} else {
		if (sw_condition[swno].BIT.status == SWON){
			if (sw_condition[swno].BIT.dirctn == DR_OFF){
				sw_condition[swno].BIT.cunter  -= 1;
				if (sw_condition[swno].BIT.cunter == 0){
					sw_condition[swno].BIT.cunter = 3;
					sw_condition[swno].BIT.dirctn = DR_ON;
					sw_condition[swno].BIT.status = SWOFF;
				}
			}
		} else {
			if (sw_condition[swno].BIT.dirctn == DR_ON){
				sw_condition[swno].BIT.cunter = 3;
			}
		}
	}
}

/* -----------------------------------------------------------------------------------------------
	CY0 Task  --- control for Front switches and check operation mode
 ------------------------------------------------------------------------------------------------*/
void cy0_init( void )
{
	lcd_update = NEED_UPDATE;
	mode = M_NORML;
	lap_ptr_buf_top = 0;
	eepdt_flg = 0;
}

//
//       M_NORMAL
//    |-->M_MDSLT
//         |-->M_LPDSP
//         |    |-->M_SWD
//         |    |-->M_IRD
//         |    |-->M_MGD
//         |-->M_DTENT
//              |-->M_STGTM
//              |-->M_LPMIN
//              |-->M_MGNOF
//              |-->M_MCLR
//              |-->M_BZOFF
//              |-->M_GPSIF
```

```
//
void cy0_main( void )
{
	switch (mode){
		case M_NORML:{				// Main mode for Lap data display
			swchk_normal();
			break;
		}
		case M_MDSLT:{				// Select memory data and data set
			swchk_mode_select();
			break;
		}
		case M_LPDSP:{				// Memory data  - SW, IR, MG
			swchk_lap_display();
			break;
		}
		case M_SWD:{
			swchk_lap_sw_disp();
			break;
		}
		case M_IRD:{
			swchk_lap_ir_disp();
			break;
		}
		case M_MGD:{
			swchk_lap_mg_disp();
			break;
		}
		case M_DTENT:{				// Data set
			swchk_data_entry();
			break;
		}
		case M_STGTM:{
			swchk_dt_stgtm_ent();
			break;
		}
		case M_LPMIN:{
			swchk_dt_lapmin_ent();
			break;
		}
		case M_MGNOF:{
			swchk_dt_mgnof_ent();
			break;
		}
		case M_MCLR:{
			swchk_dt_mclr_ent();
			break;
		}
		case M_BZOFF:{
			swchk_dt_bzoff_ent();
			break;
		}
		case M_GPSIF:{
			swchk_gps_inf();
			break;
		}
		default: {
			/*  Error */
```

```
                        entry_no = 0;
                        mode = M_NORML;
                }
        }
}

// ----------- Normal Mode  ------------------------
void swchk_normal()
{
        swchk_md(M_MDSLT);
}

// ----------- Mode select  ------------------------
void swchk_mode_select()
{
        swchk_md(M_NORML);
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
                lcd_update = NEED_UPDATE;
                tim_usr2 = T_50MS;                  // for Buzzer ON
                switch (entry_no){
                        case 0:{
                                entry_no = 0;
                                mode = M_LPDSP;
                                break;
                        }
                        case 1:{
                                entry_no = 0;
                                mode = M_GPSIF;
                                break;
                        }
                        case 2:{
                                entry_no = 0;
                                mode = M_DTENT;
                                break;
                        }
                        case 3:{
                                entry_no = 0;
                                mode = M_NORML;
                                break;
                        }
                        default: {
                                /*  back to Normal mode */
                                entry_no = 0;
                                mode = M_NORML;
                        }
                }
        }
        swchk_s(4);
}

// ----------- Lap display Mode  --------------------
void swchk_lap_display()
{
        swchk_md(M_MDSLT);
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
                lcd_update = NEED_UPDATE;
                tim_usr2 = T_50MS;                  // for Buzzer ON
                switch (entry_no){
                        case 0:{
                                mode = M_SWD;
                                lap_ptr_buf_top = 0;
                                eeprom_rd_req(SW_LAP);
                                break;
                        }
                        case 1:{
                                mode = M_IRD;
                                lap_ptr_buf_top = 0;
                                eeprom_rd_req(IR_LAP);
                                break;
                        }
                        case 2:{
                                mode = M_MGD;
                                lap_ptr_buf_top = 0;
                                eeprom_rd_req(MG_LAP);
                                break;
                        }
                        default:{
                                /*  back to Normal mode */
                                entry_no = 0;
                                mode = M_NORML;
                        }
                }
        }
        swchk_s(4);
}

void swchk_lap_sw_disp()
{
        swchk_md(M_LPDSP);
        sw_lap_scrol(SW_LAP);
}

void swchk_lap_ir_disp()
{
        swchk_md(M_LPDSP);
        sw_lap_scrol(IR_LAP);
}

void swchk_lap_mg_disp()
{
        swchk_md(M_LPDSP);
        sw_lap_scrol(MG_LAP);
}

void sw_lap_scrol(unsigned char md)
{
        if (sw_condition[SHFTSW].BIT.modify == SWON){
                sw_condition[SHFTSW].BIT.modify = SWOFF;
                lap_ptr_buf_top += SCRNSIZ;
                eeprom_rd_req(md);
                tim_usr2 = T_50MS;                  // for Buzzer ON
        }
        if (sw_condition[INCRSW].BIT.modify == SWON){
                sw_condition[INCRSW].BIT.modify = SWOFF;
                if (lap_ptr_buf_top == 0){
                        return;
                } else {
                        lap_ptr_buf_top -= SCRNSIZ;
                        eeprom_rd_req(md);
                        tim_usr2 = T_50MS;           // for Buzzer ON
                }
        }
}

void eeprom_rd_req(unsigned char md)
{
        eepdt_md = md;
        eepdt_flg = REQEST;
}

//----------- Data entory Mode --------------------
void swchk_data_entry()
{
        swchk_md(M_MDSLT);
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
                lcd_update = NEED_UPDATE;
                tim_usr2 = T_50MS;                   // for Buzzer ON
                switch (entry_no){
                        case 0:{
                                mode = M_STGTM;
                                break;
                        }
                        case 1:{
                                mode = M_LPMIN;
                                break;
                        }
                        case 2:{
                                mode_step = 0;
                                entry_no = 0;
                                mode = M_MGNOF;
                                break;
                        }
                        case 3:{
                                /*  back to Normal mode */
                                entry_no = 0;
                                mode = M_NORML;
                                break;
                        }
                        case 4:{
                                mode_step = 0;
                                mode = M_MCLR;
                                break;
                        }
                        case 5:{
                                mode = M_BZOFF;
                                break;
                        }
                        case 6:{
                                /*  back to Normal mode */
                                entry_no = 0;
                                mode = M_NORML;
                                break;
                        }
                        default:{
                                /*  back to Normal mode */
                                entry_no = 0;
                                mode = M_NORML;
                        }
                }
        }
        swchk_s(7);
}

void swchk_dt_stgtm_ent()
{
        unsigned long dt;

        if (mode_step == 0){
                formater4timer(lap_best, &entry_lcdbf[0], TIM_MIN);
                mode_step++;
        }
        swchk_md(M_DTENT);
        data_modify();
        dt = enter_sw_chk();
        if ( dt != 0){
                lap_best = dt;
                req_wr_tglap();
        }
}

void swchk_dt_lapmin_ent()
{
        unsigned long dt;

        if (mode_step == 0){
                formater4timer(lap_min, &entry_lcdbf[0], TIM_MIN);
                mode_step++;
        }
        swchk_md(M_DTENT);
        data_modify();
        dt = enter_sw_chk();
        if ( dt != 0){
                lap_min = dt;
                req_wr_milap();
        }
}

void data_modify()
{
        swchk_s(6);
        swchk_i();
}

void bf_zero_chk()
{
        ;
}

/*
```

```
void bf_clr()
{
        ;
}
*/

unsigned long enter_sw_chk()
{
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
                lcd_update = NEED_UPDATE;
                tim_usr2 = T_50MS;                      // for Buzzer ON
                mode = M_DTENT;
                mode_step = 0;
                entry_no = 0;
                return(bf_cnvrt_to_base_time());
        }
        return 0;
}

unsigned long bf_cnvrt_to_base_time()
{
        unsigned long dt;

        dt  = (unsigned long)(entry_lcdbf[0] -'0') * 30000;     // xx:Tx:xx.xx
        dt += (unsigned long)(entry_lcdbf[1] -'0') * 3000; // xx:xT:xx.xx
        dt += (unsigned long)(entry_lcdbf[3] -'0') * 500;  // xx:xx:Tx.xx
        dt += (unsigned long)(entry_lcdbf[4] -'0') * 50;        // xx:xx:xT.xx
        dt += (unsigned long)(entry_lcdbf[6] -'0') * 5;         // xx:xx:xx.Tx
        dt += (unsigned long)(entry_lcdbf[7] -'0') * 2;         // xx:xx:xx.xT
        return dt;
}

void swchk_dt_mgnof_ent()
{
        swchk_md(M_DTENT);
}

void swchk_dt_bzoff_ent()
{
        swchk_md(M_DTENT);
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
                if (bz_onoff_flg == ON){
                        bz_onoff_flg = OFF;
                } else {
                        bz_onoff_flg = ON;
                        tim_usr2 = T_500MS;             // for Buzzer ON longer time!!
                }
                req_wr_bzfg();                          // write buzzer flag into EEPROM
        }
}

void swchk_dt_mclr_ent()
{
        swchk_md(M_DTENT);
        if (sw_condition[ENTRSW].BIT.modify == SWON){
                sw_condition[ENTRSW].BIT.modify = SWOFF;
```

```
		if (mode_step == 0){
			mode_step = 1;
			tim_usr2 = T_50MS;			// for Buzzer ON
		} else if(mode_step == 1){
			eclearlap();
			mode_step = 2;
			tim_usr2 = T_500MS;			// for Buzzer ON longer time!!
		}
	}
}

//----------- GPS data display Mode --------------
void swchk_gps_inf()
{
	swchk_md(M_DTENT);
}

//----------- Switch condition check -------------
void swchk_md(unsigned char md)
{
	if (sw_condition[MODESW].BIT.modify == SWON){
		sw_condition[MODESW].BIT.modify = SWOFF;
		mode = md;
		lcd_update = NEED_UPDATE;
		entry_no = 0;
		tim_usr2 = T_50MS;			// for Buzzer ON
	}
}

void swchk_s(char maxno)
{
	if (sw_condition[SHFTSW].BIT.modify == SWON){
		sw_condition[SHFTSW].BIT.modify = SWOFF;
		lcd_update = NEED_UPDATE;
		if (++entry_no >= maxno){
			entry_no = 0;
		}
		tim_usr2 = T_50MS;			// for Buzzer ON
	}
}

void swchk_i()
{
	if (sw_condition[INCRSW].BIT.modify == SWON){
		sw_condition[INCRSW].BIT.modify = SWOFF;
		lcd_update = NEED_UPDATE;
		tim_usr2 = T_50MS;			// for Buzzer ON
		// data change
		switch (entry_no){
			case 0:{
				entry_lcdbf[0]++;
				if (entry_lcdbf[0] > '5'){
					entry_lcdbf[0] = '0';
				}
				break;
			}
			case 1:{
				entry_lcdbf[1]++;
```

```
                    if (entry_lcdbf[1] > '9'){
                        entry_lcdbf[1] = '0';
                    }
                    break;
                }
                case 2:{
                    entry_lcdbf[3]++;
                    if (entry_lcdbf[3] > '5'){
                        entry_lcdbf[3] = '0';
                    }
                    break;
                }
                case 3:{
                    entry_lcdbf[4]++;
                    if (entry_lcdbf[4] > '9'){
                        entry_lcdbf[4] = '0';
                    }
                    break;
                }
                case 4:{
                    entry_lcdbf[6]++;
                    if (entry_lcdbf[6] > '9'){
                        entry_lcdbf[6] = '0';
                    }
                    break;
                }
                case 5:{
                    entry_lcdbf[7]++;
                    if (entry_lcdbf[7] > '9'){
                        entry_lcdbf[7] = '0';
                    }
                    break;
                }
                default:{
                    /*  error */
                    entry_no = 0;
                }
            }
        }
}

/* ---------------------------------------------------------------------------------------------
        CY2 Task  --- LCD control
 ---------------------------------------------------------------------------------------------*/
void cy2_init( void )
{
        lcd_initial_screen();
}

void cy2_main( void )
{
        wr_lcd(RETURN_HOME,C);
        switch (mode){
            case M_NORML:{
                if (lcd_updating == STOP){
                    formater4timer((base_timer  -  lap_start), &lcdbuf[72], TIM_MIN);
                    lcdbuf[79] = 'x';     // 10mS position
                    lcdbuf[80] = ' ';
```

```
                        put_lcd(lcdbuf);
                        lcd_updating = STOP;
                }
                break;
        }
        case M_MDSLT:{
                lcd_mode_select();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_LPDSP:{
                lcd_lap_display();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_SWD:{
                lcd_lap_sw_disp();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_IRD:{
                lcd_lap_ir_disp();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_MGD:{
                lcd_lap_mg_disp();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_DTENT:{
                lcd_data_entry();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_LPMIN:{
                lcd_dt_lapmin_ent();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_MGNOF:{
                lcd_dt_mgnof_ent();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_BZOFF:{
                lcd_dt_bzoff_ent();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_STGTM:{
                lcd_dt_stgtm_ent();
                put_lcd(anthr_lcdbuf);
                break;
        }
        case M_MCLR:{
                lcd_dt_mclr_ent();
                put_lcd(anthr_lcdbuf);
```

```
                        break;
                }
                case M_GPSIF:{
                        lcd_gps_inf();
                        put_lcd(anthr_lcdbuf);
                        break;
                }
                default: {
                        /*  Error */
                }
        }
}

// This routine is not CY02 but cooperate with CY02
// This routine is called from Idling routine
void lcd_updating_action()
{
        char *p;
        unsigned int i;
        unsigned long dt,tmp;

        dt = 0;
        lcd_updating = RUN;
        // preparation of LCD data
        p = init_scrn_msg0;
        for (i = 0; i <40; i++){            // copy base layout Line#1
                lcdbuf[i] = *p++;
        }
        p = init_scrn_msg1;
        for (i = 40; i <80; i++){           // copy base layout Line#2
                lcdbuf[i] = *p++;
        }
        // ---line #1 ----
        if (latest_dt == 0x80+IR_LAP){                      // lap by IR
                lap_last2 = lap_last1;
                lap_last1 = lap_latest;
                lcdbuf[8] = 'I';
                lcdbuf[9] = 'R';
                dt = lap_latest = tmp_eep[IR_LAP].laptim;
        } else if (latest_dt == 0x80+MG_LAP){      // lap by MG
                lap_last2 = lap_last1;
                lap_last1 = lap_latest;
                lcdbuf[8] = 'M';
                lcdbuf[9] = 'G';
                dt = lap_latest = tmp_eep[MG_LAP].laptim;
        }
        if (lap_latest != TIM_MAX_DT){
                formater4timer(lap_latest, &lcdbuf[12], TIM_MIN);
                lcdbuf[20] = ' ';
        }
        if (lap_best == TIM_MAX_DT){ // fist lap measure
                dt = 0;                                     // diff = 0
                lcdbuf[21] = ' ';                   // +/ - zero
                lap_best = lap_latest;              // save new best lap
        } else if (lap_best > dt ){         // become new best
                if (dt > lap_min){
                        dt = lap_best  - dt;        // dif from old one
                        lcdbuf[21] = ' -';          // more short time
```

```
				lap_best = lap_latest;		// save new best lap
			} else {
				dt = 0;					// meaningless of diff
			}
		} else {						// not achive ne recode
			dt = dt  - lap_best;			// diff with best
			lcdbuf[21] = '+';				// takes longer time
		}
		formater4timer(dt, &lcdbuf[22], TIM_MIN);
		lcdbuf[30] = ' ';
		if (lap_latest != TIM_MAX_DT){
			formater4timer(lap_latest, &lcdbuf[32], TIM_MIN);
//			lcdbuf[40] = ' ';
		}
		// ---line #2 ----
		if (lap_best != TIM_MAX_DT){ // best lap
			formater4timer(lap_best, &lcdbuf[42], TIM_MIN);
			lcdbuf[40] = 'T';
			lcdbuf[41] = 'R';
			lcdbuf[50] = ' ';
		}
		if (lap_latest != TIM_MAX_DT){
			formater4timer(lap_latest, &lcdbuf[52], TIM_MIN);
			lcdbuf[60] = 'S';
		}
		if (lap_1st_one.BIT.SW == 1){		// lap data by manual switch
			formater4timer(tmp_eep[SW_LAP].laptim, &lcdbuf[62], TIM_MIN);
			lcdbuf[60] = 'S';
			lcdbuf[61] = 'W';
			lcdbuf[70] = ' ';
		}
		lcd_updating = STOP;
}

// ----------- Mode select -------------------------
void lcd_mode_select()
{
		char *p;
		unsigned int i;

		// preparation of LCD data
		p = mode_slct_msg0;
		for (i = 0; i <40; i++){			// copy base layout Line#1
			anthr_lcdbuf[i] = *p++;
		}
		p = mode_slct_msg1;
		for (i = 40; i <80; i++){			// copy base layout Line#2
			anthr_lcdbuf[i] = *p++;
		}
		switch (entry_no){
			case 0:{
				anthr_lcdbuf[50] = '*';
				break;
			}
			case 1:{
				anthr_lcdbuf[58] = '*';
				break;
			}
		case 2:{
			anthr_lcdbuf[65] = '*';
			break;
		}
		case 3:{
			anthr_lcdbuf[77] = '*';
			break;
		}
		default:{
			/*  Error */
			entry_no = 0;
			anthr_lcdbuf[50] = '*';
		}
	}
}

//----------- Lap display Mode  --------------------
void lcd_lap_display()
{
	char *p;
	unsigned int i;

	// preparation of LCD data
	p = lapdata_msg0;
	for (i = 0; i <40; i++){		// copy base layout Line#1
		anthr_lcdbuf[i] = *p++;
	}
	p = lapdata_msg1;
	for (i = 40; i <80; i++){		// copy base layout Line#2
		anthr_lcdbuf[i] = *p++;
	}
	switch (entry_no){
		case 0:{
			anthr_lcdbuf[53] = '*';
			break;
		}
		case 1:{
			anthr_lcdbuf[61] = '*';
			break;
		}
		case 2:{
			anthr_lcdbuf[69] = '*';
			break;
		}
		case 3:{
			anthr_lcdbuf[76] = '*';
			break;
		}
		default:{
			/*  Error */
			entry_no = 0;
			anthr_lcdbuf[53] = '*';
		}
	}
}
void lcd_lap_sw_disp()
{
	lcd_on_lap();
```

```
        anthr_lcdbuf[0] = 'S';
        anthr_lcdbuf[1] = 'W';
}

void lcd_lap_ir_disp()
{
        lcd_on_lap();
        anthr_lcdbuf[0] = 'I';
        anthr_lcdbuf[1] = 'R';
}

void lcd_lap_mg_disp()
{
        lcd_on_lap();
        anthr_lcdbuf[0] = 'M';
        anthr_lcdbuf[1] = 'G';
}

void lcd_on_lap()
{
        char *p;
        unsigned char i;

        // preparation of LCD data
        p = lapdata_msg4;
        for (i = 0; i <40; i++){            // copy base layout Line#1
                anthr_lcdbuf[i] = *p++;
        }
        p = lapdata_msg4;
        for (i = 40; i <80; i++){           // copy base layout Line#2
                anthr_lcdbuf[i] = *p++;
        }
        if (eepdt_flg == READY){
                if (lap_data_buf[0].id != NOTDATA){        // 1st
                        conv_id(lap_data_buf[0].id,&anthr_lcdbuf[68]);
                        formater4timer(lap_data_buf[0].laptim, &anthr_lcdbuf[72], TIM_MIN);
//                      anthr_lcdbuf[80] = ' ';
                        i++;
                } else {
                        return;
                }
                if (lap_data_buf[1].id != NOTDATA){        // 2nd
                        conv_id(lap_data_buf[1].id,&anthr_lcdbuf[28]);
                        formater4timer(lap_data_buf[1].laptim, &anthr_lcdbuf[32], TIM_MIN);
                        anthr_lcdbuf[40] = ' ';
                        i++;
                } else {
                        return;
                }
                if (lap_data_buf[2].id != NOTDATA){        // 3rd
                        conv_id(lap_data_buf[2].id,&anthr_lcdbuf[48]);
                        formater4timer(lap_data_buf[2].laptim, &anthr_lcdbuf[52], TIM_MIN);
                        anthr_lcdbuf[60] = ' ';
                        i++;
                } else {
                        return;
                }
                if (lap_data_buf[3].id != NOTDATA){        // 4th
```

```
                    conv_id(lap_data_buf[3].id,&anthr_lcdbuf[ 8]);
                    formater4timer(lap_data_buf[3].laptim, &anthr_lcdbuf[12], TIM_MIN);
                    anthr_lcdbuf[20] = ' ';
                    i++;
            } else {
                    return;
            }
        }
}

//----------- Data entory Mode --------------------
void lcd_data_entry()
{
        char *p;
        unsigned int i;

        // preparation of LCD data
        if (entry_no >=4){
                p = dtentr_msg2;
        } else {
                p = dtentr_msg0;
        }
        for (i = 0; i <40; i++){            // copy base layout Line#1
                anthr_lcdbuf[i] = *p++;
        }
        if (entry_no >=4){
                p = dtentr_msg3;
        } else {
                p = dtentr_msg1;
        }
        for (i = 40; i <80; i++){           // copy base layout Line#2
                anthr_lcdbuf[i] = *p++;
        }
        switch (entry_no){
                case 0:{
                        anthr_lcdbuf[52] = '*';
                        break;
                }
                case 1:{
                        anthr_lcdbuf[62] = '*';
                        break;
                }
                case 2:{
                        anthr_lcdbuf[70] = '*';
                        break;
                }
                case 3:{
                        anthr_lcdbuf[76] = '*';
                        break;
                }
                case 4:{
                        anthr_lcdbuf[52] = '*';
                        break;
                }
                case 5:{
                        anthr_lcdbuf[64] = '*';
                        break;
                }
```

```c
			case 6:{
				anthr_lcdbuf[76] = '*';
				break;
			}
			default:{
				/*  Error */
				entry_no = 0;
				anthr_lcdbuf[52] = '*';
			}
		}
}

void lcd_dt_lapmin_ent()
{
		lcd_dt_cmn_ent();
}

void lcd_dt_mgnof_ent()
{
		error_screen();			// Not impliment yet
}

void lcd_dt_bzoff_ent()
{
		char *p;
		unsigned char i;

		// preparation of LCD data
		p = dtent_bzr_msg0;
		for (i = 0; i <40; i++){			// copy base layout Line#1
			anthr_lcdbuf[i] = *p++;
		}
		p = dtent_bzr_msg1;
		for (i = 40; i <80; i++){			// copy base layout Line#2
			anthr_lcdbuf[i] = *p++;
		}
}

void lcd_dt_stgtm_ent()
{
		lcd_dt_cmn_ent();
}

void lcd_dt_cmn_ent()
{
		char *p;
		unsigned int i;

		// preparation of LCD data
		p = dtentr_msg4;
		for (i = 0; i <40; i++){			// copy base layout Line#1
			anthr_lcdbuf[i] = *p++;
		}
		p = dtentr_msg5;
		for (i = 40; i <80; i++){			// copy base layout Line#2
			anthr_lcdbuf[i] = *p++;
		}
		for (i = 0; i <8; i++){					// copy lap data
```

```
                anthr_lcdbuf[i+44] = entry_lcdbf[i];
        }
        switch (entry_no){
                case 0:{
                        anthr_lcdbuf[24] = '*';
                        break;
                }
                case 1:{
                        anthr_lcdbuf[25] = '*';
                        break;
                }
                case 2:{
                        anthr_lcdbuf[27] = '*';
                        break;
                }
                case 3:{
                        anthr_lcdbuf[28] = '*';
                        break;
                }
                case 4:{
                        anthr_lcdbuf[30] = '*';
                        break;
                }
                case 5:{
                        anthr_lcdbuf[31] = '*';
                        break;
                }
                default:{
                        /*  Error */
                        entry_no = 0;
                        anthr_lcdbuf[24] = '*';
                }
        }
}

void lcd_dt_mclr_ent()
{
        char *p;
        unsigned char i;

        // preparation of LCD data
        if (mode_step == 0){
                p = dtent_clr_msg0;
        } else if(mode_step == 1){
                p = dtent_clr_msg2;
        } else {
                p = dtent_clr_msg4;
        }
        for (i = 0; i <40; i++){              // copy base layout Line#1
                anthr_lcdbuf[i] = *p++;
        }
        if (mode_step == 0){
                p = dtent_clr_msg1;
        } else if(mode_step == 1){
                p = dtent_clr_msg3;
        } else {
                p = dtent_clr_msg5;
        }
        for (i = 40; i <80; i++){           // copy base layout Line#2
                anthr_lcdbuf[i] = *p++;
        }
}

// ----------- GPS data display Mode  ---------------
void lcd_gps_inf()
{
        error_screen();             // Not impliment yet
}

// ----------- Error / Not impliment yet  ------------
void error_screen()
{
        char *p;
        unsigned int i;

        // preparation of LCD data
        p = err_msg0;
        for (i = 0; i <40; i++){            // copy base layout Line#1
                anthr_lcdbuf[i] = *p++;
        }
        p = err_msg1;
        for (i = 40; i <80; i++){           // copy base layout Line#2
                anthr_lcdbuf[i] = *p++;
        }
}

//      LCD initialize routine
void lcd_initial_screen()
{
        char *p;
        unsigned int i;

//      Both LCD and data buffer
        p = init_scrn_msg0;
        for (i = 0; i <40; i++){            // copy base layout Line#1
                lcdbuf[i] = *p++;
        }
        p = init_scrn_msg1;
        for (i = 40; i <80; i++){           // copy base layout Line#2
                lcdbuf[i] = *p++;
        }
        wr_lcd(RETURN_HOME,C);
        put_lcd(lcdbuf);
}

void test_mode()
{
        static unsigned char x;

        if (sw_condition[0].BIT.status == SWOFF){
                anthr_lcdbuf[1] = '1';
        } else {
                anthr_lcdbuf[1] = '0';
        }
        if (sw_condition[1].BIT.status == SWOFF){
                anthr_lcdbuf[2] = '1';
        } else {
                anthr_lcdbuf[2] = '0';
        }
        if (sw_condition[2].BIT.status == SWOFF){
                anthr_lcdbuf[3] = '1';
        } else {
                anthr_lcdbuf[3] = '0';
        }
        if (sw_condition[3].BIT.status == SWOFF){
                anthr_lcdbuf[4] = '1';
        } else {
                anthr_lcdbuf[4] = '0';
        }
        if (x == 0){
                x = 1;
                anthr_lcdbuf[5] = '0';
        } else {
                x = 0;
                anthr_lcdbuf[5] = '1';
        }
        if (SW_MODE == SWON){
                anthr_lcdbuf[6] = '0';
        } else {
                anthr_lcdbuf[6] = '1';
        }
        if (SW_ENTR == SWON){
                anthr_lcdbuf[7] = '0';
        } else {
                anthr_lcdbuf[7] = '1';
        }
        if (SW_SHFT == SWON){
                anthr_lcdbuf[8] = '0';
        } else {
                anthr_lcdbuf[8] = '1';
        }
        if (SW_INCR == SWON){
                anthr_lcdbuf[9] = '0';
        } else {
                anthr_lcdbuf[9] = '1';
        }
}

/* -------------------------------------------------------------------------------------------
        CY3 Task   --- Not use
 -------------------------------------------------------------------------------------------*/
void cy3_init( void ) {      ; }
void cy3_main( void ) { ; }
/* -------------------------------------------------------------------------------------------
        CY4 Task   --- Not use
 -------------------------------------------------------------------------------------------*/
void cy4_init( void ) {      ; }
void cy4_main( void ) { ; }
/* -------------------------------------------------------------------------------------------
        CY5 Task   --- Not use
 -------------------------------------------------------------------------------------------*/
void cy5_init( void ) {      ; }
void cy5_main( void ) { ; }
/* -------------------------------------------------------------------------------------------
```

```
       CY6 Task   --- Not use
 -------------------------------------------------------------------------------------------------*/
void cy6_init( void ) {      ; }
void cy6_main( void ) { ; }
/* -----------------------------------------------------------------------------------------------
       CY7 Task   --- Not use
 -------------------------------------------------------------------------------------------------*/
void cy7_init( void ) {      ; }
void cy7_main( void ) { ; }

/*************************************************************************************************
************** General initialization for each task and base task for monitor *******************
************************************************************************************************/
void main( void )
{
       /********** Interrupt ****************************/
       irq_disable();
       /********** TimerV *******************************/
       reset_timerV();
       /********** RTM **********************************/
       rtm_h8_init();
       /********** Hardware related initialize *********/
       port_init();
       /********** SCI **********************************/
       SciInit(rxBuf, sizeof( rxBuf ), IdleEEP );
       PutCRLF();    PutCRLF();
       opning_msg();
       /********** others *******************************/
       general_init();
       /********** Interrupt ****************************/
       irq_enable();
       /********** EEPROM *******************************/
       init_eep_lap();
       /********** LCD **********************************/
       reset_lcd();
       /********** TimerW *******************************/
       reset_timerW();


       /* Initialize each task */
       cy0_init();
       cy1_init();
       cy2_init();
       cy3_init();
       cy4_init();

       while(!DOOMSDAY)
       {
              PutCRLF();
              SciPutC('*');               // prompt mark
              /*  Wait for EIA232 char  -> parse to command  */
              getline();
              Buf = line[0];              // get command
              switch(Buf){
                     /*  Buzzer control */
                     case 'B': {
                            goto job_B;
                     }
```

```
                    case 'b': {
job_B:
                           if (bz_onoff_flg == OFF){
                                  bz_onoff_flg = ON;
                                  tim_usr2 = T_500MS;
                                  while (tim_usr2 > T_200MS){
                                         ;
                                  }
                                  bz_onoff_flg = OFF;
                           } else {
                                  tim_usr2 = T_500MS;                // for Buzzer ON longer time!!
                           }
                           puts("Can you hear?");
                           break;
                    }
                    /*  Clear Time base */
                    case 'C': {
                           goto job_C;
                    }
                    case 'c': {
job_C:
                           base_timer = 0;
                           puts("Cleared time base");
                           break;
                    }
                    /*  Send Help massage */
                    case '?': {
                           goto job_H;
                    }
                    case 'H': {
                           goto job_H;
                    }
                    case 'h': {
job_H:
                           help_msg();
                           break;
                    }
                    /*  Lap data control */
                    case 'L': {
                           goto job_L;
                    }
                    case 'l': {
job_L:
                           lap_cmd_srch();
                           break;
                    }
                    /*  Monitor */
                    case 'M': {
                           goto job_M;
                    }
                    case 'm': {
job_M:
                           cmdsrch();
                           break;
                    }
                    /*  Show switch status */
                    case 'S': {
                           goto job_S;
                    }
                    case 's': {
job_S:
                            sw_status();
                            break;
                    }
                    /*  Show Timer data */
                    case 'T': {
                            goto job_T;
                    }
                    case 't': {
job_T:
//                          PutCRLF();
                            puts("Base time   SW-lap      Magnetic    InfraRed");    PutCRLF();
                            formater4timer(base_timer, timbuf, TIM_FULL);
                            puts(timbuf);
                            SciPutC(' ');
                            formater4timer(lap_sw, timbuf, TIM_FULL);
                            puts(timbuf);
                            SciPutC(' ');
                            formater4timer(lap_mg, timbuf, TIM_FULL);
                            puts(timbuf);
                            SciPutC(' ');
                            formater4timer(lap_ir, timbuf, TIM_FULL);
                            puts(timbuf);
                            break;
                    }
                    case '0': {
                            act0();
                            break;
                    }
                    case '1': {
                            act1();
                            break;
                    }
                    case '2': {
                            act2();
                            break;
                    }
                    case '4': {
                            act4();;
                            break;
                    }
                    case '8': {
                            act8();
                            break;
                    }
                    case '9': {
                            act9();
                            break;
                    }
                    default: {
                            /*  Error */
                            SciPutC('?');
                    }
            } /*  End of switch */
    } /*  End of while */
}
```

```
/*****************************************************************************************
       General subroutine
*****************************************************************************************/
void general_init( void )
{
       latest_dt = 0;
       lap_1st_one.BYTE = 0;
       lcd_updating = 0;
       lap_start = base_timer;
       lap_latest = lap_last1 = lap_last2 = TIM_MAX_DT;
       lap_best  = TIM_MIN_DT;
//     lap_min = TIM_MIN_DT;
//     bz_onoff_flg = ON;
}

/********** Port initialize ****************/
void port_init ( void )
{
       PUCR1.BYTE = 0xff;          // pull-up all inputs
       PCR1.BYTE  = 0xe0;          // P17,16,15 -- LCD control line (E, RS, and R/W)
       PDR1.BYTE  = 0x00;          //
       PCR2.BYTE  = 0x00;          //
       PDR2.BYTE  = 0;                    /* Set all low */
       PCR5.BYTE  = 0x0f;          /* Bit 0 to 3 are LCD control ports */
       PUCR5.BYTE = 0xf0;          /* Pull-up ON */
//     PDR5.BYTE  = 0;                    /* Set all low */
       PCR7.BYTE  = 0x70;          // output P74,75&76
       PDR7.BYTE  = 0x00;          /* OFF LED & Buzzer */
       PCR8.BYTE  = 0xc0;          /* Bit P86&87 = LED drive */
       PDR8.BYTE  = 0x00;          /* Set all high */
//     port B is only input
}

/* ------Opening Message------- */
void opning_msg( void )
{
       PutCRLF();
       SciPutS("---- H8/3664F Laptimer --- Ver.2 w/ PCB ----"); PutCRLF();
       SciPutS("by JH1PJL/Kenji Arai -- Nov.,2004 [?=H(ret)]"); PutCRLF();
}

void help_msg( void )
{
       opning_msg();
       SciPutS("[M]         -- Goto Simple Monitor  ");         PutCRLF();
       SciPutS("[C]         -- Clear base timer data");         PutCRLF();
       SciPutS("[T]         -- Show base timer data ");         PutCRLF();
       SciPutS("[L]         -- Control Lap data     ");         PutCRLF();
       SciPutS("[0,1,2,4,9] -- Write sample to LCD  ");         PutCRLF();
       SciPutS("[B]         -- Buzzer control       ");         PutCRLF();
       SciPutS("[S]         -- Show switch status   ");         PutCRLF();
       SciPutS("Others      -- Not impliment yet    ");         PutCRLF();

}

void sw_status(void)
{
	SciPutS(" Left <- SW1  SW2  SW3  SW4  -> Right");  PutCRLF();
	SciPutS("         ENTR MODE INCR SHFT        ");  PutCRLF();
	SciPutS("         ");
	if (SW_ENTR == ON){
		SciPutS("OFF  ");
	} else {
		SciPutS("ON   ");
	}
	if (SW_MODE == ON){
		SciPutS("OFF  ");
	} else {
		SciPutS("ON   ");
	}
	if (SW_INCR == ON){
		SciPutS("OFF  ");
	} else {
		SciPutS("ON   ");
	}
	if (SW_SHFT == ON){
		SciPutS("OFF  ");
	} else {
		SciPutS("ON   ");
	}
}

void act0(void)
{
	put_lcd("------- H8/3664F Simple Monitor  --------");
}

void act1(void)
{
	put_lcd("1234567890123456789012345678901234567890");
}

void act2(void)
{
	put_lcd("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmn");
}

void act4(void)
{
	wr_lcd(Buf,D);				// output to LCD
}

void act8(void)
{
	wr_lcd(RETURN_HOME,C);			// return to Home position
}

void act9(void)
{
	put_lcd("                                        ");
}

void formater4timer(unsigned long tim, char *dt_save, char typ)
{
	unsigned char i;
```

```
        unsigned long dt;

        dt = tim & 0x00ffffff;                                  // output Tx:xx:xx.xx
        i = (char)(dt  / 1800000) + '0';
        if (typ == TIM_FULL){
               *dt_save++ = i;
        }
        dt = dt  - ((dt / 1800000) * 1800000);      // output xT:xx:xx.xx
        i = (char)(dt  / 180000 ) + '0';
        if (typ == TIM_FULL){
               *dt_save++ = i;
               *dt_save++ = 'h';
        }
        dt = dt  - ((dt / 180000 ) * 180000 );      // output xx:Tx:xx.xx
        i = (char)(dt  / 30000  ) + '0';
        *dt_save++ = i;
        dt = dt  - ((dt / 30000  ) * 30000  );      // output xx:xT:xx.xx
        i = (char)(dt  / 3000   ) + '0';
        *dt_save++ = i;
        if (typ != TIM_DEL){
               *dt_save++ = ':';
        }
        dt = dt  - ((dt / 3000   ) * 3000   );      // output xx:xx:Tx.xx
        i = (char)(dt  / 500    ) + '0';
        *dt_save++ = i;
        dt = dt  - ((dt / 500    ) * 500   );       // output xx:xx:xT.xx
        i = (char)(dt  / 50     ) + '0';
        *dt_save++ = i;
        if (typ != TIM_DEL){
               *dt_save++ = '.';
        }
        dt = dt  - ((dt / 50   ) * 50  );           // output xx:xx:xx.Tx
        i = (char)(dt  / 5    ) + '0';
        *dt_save++ = i;
        dt = dt  - ((dt / 5    ) * 5   );           // output xx:xx:xx.xT
        i = (char)(dt  / 2    ) + '0';
        *dt_save++ = i;
        *dt_save++ = 0;
}

void conv_id(unsigned char id, char *dt_save)
{
        unsigned char i;
        unsigned char dt;

        dt = id;                                                // output dxx
        i = (char)(dt  / 100) + '0';
        *dt_save++ = i;
        dt = dt  - ((dt / 100) * 100);                  // output xdx
        i = (char)(dt  / 10 ) + '0';
        *dt_save++ = i;
        dt = dt  - ((dt / 10 ) * 10 );                  // output xxd
        i = (char)dt + '0';
        *dt_save++ = i;
        *dt_save++ = '=';
}

void abort(void)
```

```
{
        ;
}

void PutCRLF()
{
        SciPutC(0x0d);
        SciPutC(0x0a);
}

void IdleNop( void )
{
        ;
}
```