

**T-Kernel sample software  
for T-Engine/SH7727  
Users Guide**

Rev.1.00.00  
Sep.10, 2007

## Usage Precautions

- SuperH™ is a trademark of Renesas Technology Corporation.
- All other brand names and product names are trademarks of registered trademarks of the respective proprietors.
- TRON, BTRON, ITRON, eTRON, T-Engine,  $\mu$  T-Engine, T-Monitor and T-Kernel are the names given to computer specifications and do not indicate any particular product.



# Preface

This document describes the sample applications using the key switches and touch panel mounted on an LCD board.

The target board in which the sample applications are intended to be run is the T-Engine that incorporates the Renesas SuperH™ RISC engine family “SH7727.” The sample applications are designed to run under the T-Kernel.

For details about the SH7727 T-Engine hardware, refer to the user’s manual supplied with the product.

## Table of Contents

1	USING THE SAMPLE APPLICATIONS .....	2
2	FUNCTIONAL SPECIFICATIONS OF THE SAMPLE APPLICATIONS .....	3
2.1	“main” Sample Application .....	3
2.2	Key Switch Sample Application .....	3
2.3	Touch Panel Sample Application .....	6

# 1 Using the Sample Applications

When the sample application (`sample.1.00.00.sh7727.tar.gz`) is decompressed, a directory structure of extracted files like the one shown below will result.

[Directory structure]

sample.1.00.00.sh7727.tar.gz		
sample	key.c	Key switch sample application source file
	key.h	Key switch sample application header file
	tp.c	Touch panel sample application source file
	tp.h	Touch panel sample application header file
	sample_main.c	main sample application source file
	sample_main.h	main sample application header file

In the installation destination (`$BD/kernel/usermain`), execute the command shown below to decompress `sample.1.00.00.sh7727.tar.gz`

```
$ tar zxvf sample.1.00.00.sh7727.tar.gz
```

Next, prepare for building the sample applications. Before you start building, please be sure to alter the makefile `/kernel/usermain/Makefile.usermain`, as shown below.

```
# source files
SRC += usermain.c key.c tp.c sample_main.c ← Add

# source file path
VPATH += ../../../../usermain ../../../../usermain/sample ← Add
HEADER += ../../../../usermain ../../../../usermain/sample ← Add
```

When you've finished altering the makefile, launch Cygwin and build the sample applications in the `$BD/kernel/sysmain/build/std_sh7727` directory.

## 2 Functional Specifications of the Sample Applications

This section describes functional specifications of main, key switch and touch panel sample applications each.

### 2.1 “main” Sample Application

#### 2.1.1 Functional Detail

(1) void sample\_main(void) /\* main function \*/

Initializes the key switches and touch panel, as well as defines an interrupt handler.

(2) void h8\_inthdr(UINT dintno) /\* H8\_3048F interrupt handler \*/

UINT dintno Interrupt definition number

Defines an H8\_3048F interrupt handler.

### 2.2 Key Switch Sample Application

This is a sample application for the cursor switch (SW1) and push switches (SW2, SW3) mounted on an LCD board. The key switch input function is controlled by the power supply controller (H8/3048F-ONE).

#### 2.2.1 Key Switch Processing by the Power Supply Controller

The cursor switch and push switches are sampled at 10 ms intervals. When one and the same key is depressed three times in succession, key bit pattern data of cursor switch and push switches is output.

When any switch is turned on, a key-ON interrupt is generated; when any switch is turned off, a key-OFF interrupt is generated.

When one and the same switch is held down, an auto repeat interrupt is generated at intervals of 100–450 ms (in 50 ms increments).

When multiple key switches are manipulated at the same time, the processing described below applies:

When multiple key switches are depressed or released at the same time, an interrupt is generated once.

When key switches are depressed one at a time, an ON interrupt is generated.

When key switches are released one at a time, an OFF interrupt is generated. In this case, the bit pattern changes except when the last key is released.

#### 2.2.2 Key Code Assignment

The key switches are assigned the following codes each.

	Switch	Key code
Cursor switch	SW1-1(→)	0x01
	SW1-2(←)	0x02
	SW1-3(↑)	0x03
	SW1-4(↓)	0x04
	SW1-5(Enter)	0x05
Push switch	SW2	0x06
	SW3	0x07

### 2.2.3 Definition of the Auto Repeat Time

```
#define KS_100MS    0x01 /* Initial value */
#define KS_150MS    0x02
#define KS_200MS    0x04
#define KS_250MS    0x08
#define KS_300MS    0x10
#define KS_350MS    0x20
#define KS_400MS    0x40
#define KS_450MS    0x80
```

### 2.2.4 Functional Detail

#### (1) ID ks\_init(void) /\* Initialize \*/

Return value: Message buffer ID (mbfid)

Enables key operation and key-ON and key-OFF interrupts, allowing interrupt generation.

Generates a message buffer for event notification.

#### (2) UH ks\_get\_key(void) /\* Get key pattern \*/

Return value: Key pattern

Returns the current key pattern.

#### (3) ER ks\_chg\_key(UH \*onoff, UH \*code, TMO tmout) /\* Wait for key state to change \*/

UH \*onoff Indicates a change from ON to OFF or vice versa

UH \*code Key code (effective only when key ON)

TMO tmout Message buffer time-out time

Return value: Result (E\_OK, E\_PAR, E\_TMOUT)

Receives a key switch ON/OFF event.

Waits a specified **tmout** time until the key state changes.

#### (4) void ks\_ena\_int(void) /\* Enable interrupt \*/

Enables transmission of a message buffer sent from the interrupt handler.

The actual interrupt is always kept in an active state.

#### (5) void ks\_dis\_int(void) /\* Disable interrupt \*/

Disables transmission of a message buffer sent from the interrupt handler.

The actual interrupt is always kept in an active state.

#### (6) ER ks\_set\_rtime(UB time) /\* Set auto repeat interrupt time \*/

UB time Auto repeat time

Return value: Result (E\_OK, E\_PAR)

Sets the time at which an auto repeat interrupt is generated.

(7) void ks\_inthdr(UINT dintno) /\* Key interrupt handler \*/

UINT dintno Interrupt number

Notifies via a message buffer the key code of a key whose state has changed as compared with the previous key pattern.

Always clears the corresponding bits in the key input status register at the end of processing.

Events are notified using the format shown below. TDE\_KEYUP or TDE\_KEYDOWN is set in **evttyp**.

```
typedef struct {
    T_DEVEVT h;          /* Standard header */
    UH      keytop;     /* Keytop code */
    UH      code;       /* Key code */
} KeyEvt;

h.evttyp:
TDE_KEYUP          /* Key up */
TDE_KEYDOWN        /* Key down */
```

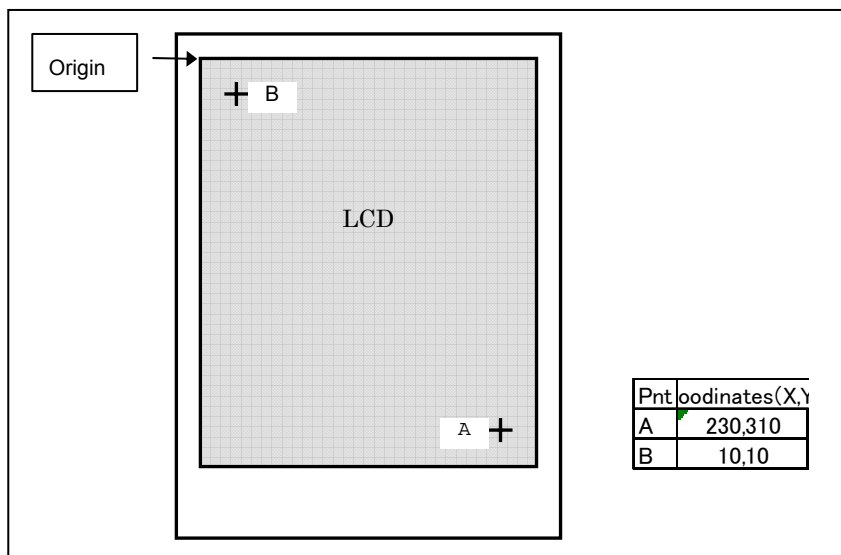
## 2.3 Touch Panel Sample Application

This is a sample application for the touch panel mounted on an LCD board. The function to read coordinate positions on the touch panel is controlled by the power supply controller (H8/3048F-ONE). Functional description of this sample application is provided below.

- (1) The A/D converted value (12-bit digital data) of pen-touched X and Y positions is output.
- (2) Pen touch ON/OFF interrupt function. Input on the touch panel is sampled at 20–100 ms intervals. If the A/D converted values of X and Y positions sampled three times in succession are close to each other, the SH7727 is solicited to generate a pen touch-ON interrupt. When the pen is moved off the touch panel, a pen touch-OFF interrupt is generated.
- (3) When the pen remains touched on, input on the touch panel is sampled at 20–100 ms intervals. If the sampled results are close to each other, a pen touch-ON interrupt is generated.
- (4) Calibration function. Calibration is performed by touching two points on the panel with a pen. After calibration is executed, the X and Y positions converted to drawing dot positions on LCD are output.

### 2.3.1 About the Calibration

To perform calibration, draw positions at which you touch the LCD with a pen. Set the coordinates used for calibration as shown below, and draw a cross centering on each coordinate point you've set.



### 2.3.2 Definition of the Sampling Period

The touch-panel sampling period can be set. You can set the sampling period in the range 20 ms to 160 ms (in 20 ms increments).

```
#define TP_20MS 0x01 /* Initial value */
#define TP_40MS 0x02
#define TP_60MS 0x04
#define TP_80MS 0x08
#define TP_100MS 0x10
#define TP_120MS 0x20
#define TP_140MS 0x40
#define TP_160MS 0x80
```

### 2.3.2 Functional Details

(1) ID `tp_init(void)` /\* Initialize \*/

Return value: Message buffer ID (mbfid)

Enables touch-panel operation and pen touch-ON and pen touch-OFF interrupts, allowing interrupt generation.

Generates a message buffer for event notification.

(2) ER `tp_calibration(void)` /\* Calibration \*/

Return value: Result (E\_OK, E\_PAR)

Performs touch-panel calibration.

(3) ER `tp_tch_pen(UH *onoff, POS pos, TMO tmout)` /\* Get pen-touch position \*/

UH \*onoff: Indicates a pen state (0: off, 1: on)

POS pos: Pen-touch position (dot coordinates)

TMO tmout Message buffer time-out time

Return value: Result (E\_OK, E\_PAR, E\_TMOOUT)

Gets a pen-touch position every sampling period.

Waits a specified `tmout` time until the panel is touched.

(4) void `tp_ena_int(void)` /\* Enable interrupt \*/

Enables transmission of a message buffer sent from the interrupt handler.

The actual interrupt is always kept in an active state.

(5) void `tp_dis_int(void)` /\* Disable interrupt \*/

Disables transmission of a message buffer sent from the interrupt handler.

The actual interrupt is always kept in an active state.

(6) ER `tp_set_cyctime(UB time)` /\* Set sampling cycle time \*/

UB time Sampling cycle time

Return value: Result (E\_OK, E\_PAR)

Sets a sampling cycle time.

(7) void `tp_inthdr(UINT dintno)` /\* Pen touch ON/OFF interrupt handler \*/

UINT dintno Interrupt number

When calibration finished, gets the values of X position dot register (XPDR) and Y position dot register (YPDR) and notifies it via a message buffer. When calibration unexecuted, it returns an invalid value.

Always clears the corresponding bits in the touch panel status register at the end of processing.

Events are notified using the format shown below. TDE\_PDMOVE is set in `evttyp`.

```
typedef struct {
    T_DEVEVT  h;      /* Standard header */
    UH        stat;   /* Button status ON/OFF */
    POS       pos;    /* Pen-touch position (dot coordinates) */
} PdEvt;
h.evttyp:      TDE_PDBUT /* Change of PD button status */
```

---

T-Kernel Sample software for T-Engine/SH7727 Users Guide

Publication Date: Sep.10.2007      Rev.1.00.00

Published by:      SH/M32R T-Engine HomePage

Edited by:      SH/M32R T-Engine HomePage

---

T-Kernel Sample software  
for T-Engine/SH7727  
Users Guide

