Using RTXC Real Time OS on the M16C

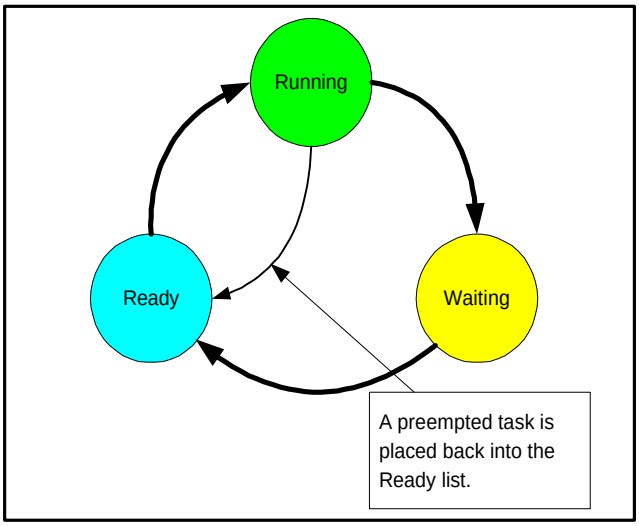
1. ABSTRACT

The use of Real Time Operating Systems (RTOS) in embedded systems has many advantages. An RTOS allows the developer to define a task as if it had the microcontroller all to itself. The RTOS ensures that the tasks are isolated one from another. This isolation between the defined tasks, protects locally allocated task memory from being corrupted by other tasks. Other advantages for using an RTOS are decreased development time, simplified debugging, prioritizing of tasks, simplified system software expansion, and ease of documentation. This note contains information on the use of RTOS, Embedded System Product's RTXC, and IAR's Embedded Workbench on Mitsubishi's M16C Microcontroller.

2. INTRODUCTION TO RTOSS

RTOSs provide application-programming interfaces to Kernel Services. These services may include functions for task control, semaphores, FIFO queues, mailboxes, event flags, and memory management. The Tasks are what the system designer creates to perform work. The three states of created and enabled task are waiting, ready, and running. Please see figure one.

Figure 1: The three states of an enabled task.



A task that is suspended is placed into an indefinite wait state. A preempted task is placed back on to the ready list. The kernel may provide task control functions for task creation, task deletion, task suspension, and task starting.

Semaphores are used to control access to shared resources, signal the occurrence of an event, and allow two tasks to synchronize their activities.

FIFO (First In, First Out) Queues are circular buffers that allow the system to pass data from one task to another. For example, serial ports use FIFOs to pass data from the interrupt handler to the task that processes the input information.

Mailboxes are used to pass message pointers from one task to another. The task that is sending the pointer and the task that is receiving the pointer decide the format of the data. Passing pointers is an efficient way of moving large blocks of data from one task to another.

Event Flags are used to synchronize events from different tasks. For example a task that controls an Analog to Digital converter could send an event flag to the task that processes the stored digital data. This allows the signaling task to continue controlling the AD converter while a background task processes the data.

Some RTOSs provide functions to manage memory. These RTOSs allow a designer to dynamically allocate blocks of memory to various tasks. These memory management functions allow a user to allocate memory, create a task, let the task do the work, delete the task, and then reclaim the memory for use by other tasks.

For more information on RTOS please see Chapter 3 of Mitsubishi's *M16C/60 Series C Language Programming Manual*, and Embedded System Product's *RTXC User's Manual*.

3. RTXC AND EMBEDDED WORKBENCH TOOLS

Embedded Systems Product's RTXC is a Real Time Operating System that provides an Application Programming Interface (API) to an extensive set of Kernel Services. These Kernel Services (KS) include services for Task, Semaphores, Mailboxes, Messages, Queues, Timer, and System Time. RTXC supports Time-Sliced, Preemptive, and Round Robin task scheduling algorithms. RTXC requires the use of IAR's Embedded Workbench for building the example program to run on Mitsubishi's M16C Microcontroller.

IAR's Embedded Workbench provides editing, project management, and make control for M16C

embedded software projects. The development environment includes C and assembly language compilers and a program linker.

4. THE RTXC_AD/LED EXAMPLE PROGRAM

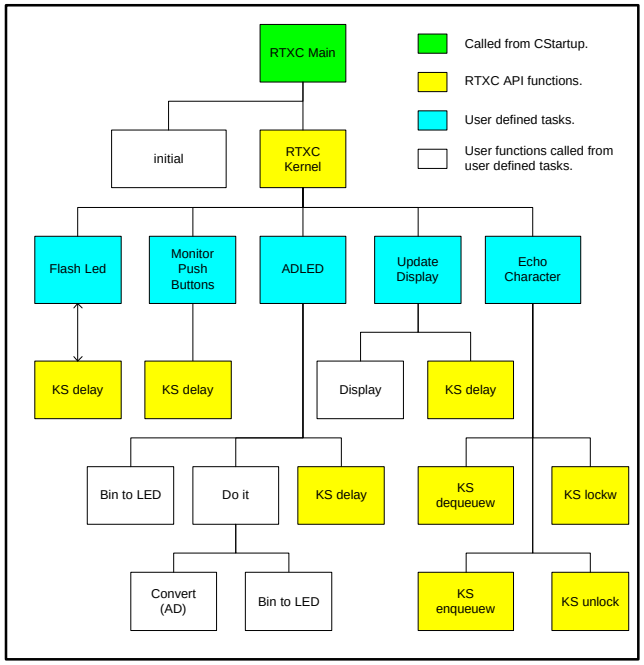
This package includes an example program that uses some of the RTXC services. The program is written to run on the Mitsubishi's MSA0600 Starter Kit or PC4701HS/PC4701L emulators with either the M30610TRPD-E or M30620T-RPD-E pods. The program uses the MTK7704B I/O board's 4 digit LED display, keypad, analog input, and RS232 serial port.

Program Structure

Excluding the RTXC supplied support tasks the RTXC_AD/LED example program consists of five tasks and five functions. The five tasks are; Flash LED, Monitor Push Buttons (keypad), ADLED (Perform A/D or Calculate LED Display Value), Update Display, and Echo Character. Please see figure 2, *Example Program Structure Chart*.

RTXC Main is called from the assembly language file "C-startup." When RTXC Main is called, it initializes all of the peripheral functions by calling "Initial" before any of the kernel startup routines are called. After "Initial" is executed then each task is initialized and placed on to the ready list. After all of the tasks are initialized, the kernel is then started. The kernel gets the highest priority task off of the ready list and executes it. When the task calls KS_delay (KS = Kernel Service) the kernel places the task into the wait list, sets the task timer and transfer CPU control to the next highest priority task.

Figure 2: Example Program Structure Chart



Here's how we divide up the tasks and decide priority and timing. The real time interrupt (RTI) fires every 5 milliseconds. For the example program, Timer A0 is used to generate the RTI. Any of the available timers on the M16C can be used to generate the RTI. When RTI occurs, the kernel will then take control of the CPU (M16C Core) and process the interrupt. During the processing of the RTI, the kernel decrements all of the waiting task timers. All tasks whose timers decrement to 0 when the kernel processes the RTI will be removed from the waiting list and placed onto the ready list. After processing the RTI, the kernel will then pass the CPU to the highest priority task that is listed on the ready list.

Tasks that execute often usually have a lower priority then tasks that seldom execute. For example, the Update Display task runs every 5 milliseconds. To prevent blocking other tasks from running, the Update Display has lower priority then some of the other tasks.

But Update Display must meet its timing requirements or the display will flicker. To prevent the display from flickering, The Echo Character and Monitor Push Buttons tasks have lower priorities then the Update Display Task. If the Monitor Push Button task is running when the RTI occurs, the kernel will preempt the Push Button Task and hand the CPU over to the Update Display task. One of the main advantages in using RTOSs is the ability to give control of the CPU to time critical processes.

Between the occurrences of the RTI, a task that finishes executing its process and calls KS_delay is placed onto the waiting list by the kernel. Next, the kernel will get the highest priority task off of the ready list and transfer control of the CPU over to that task.

For more information on the RTXC kernel, please see the RTXC Users Guide.

Program Description

The example program is made of both user application code files and RTXCgen generated code files. RTXCgen, a utility supplied with RTXC, is used to create the configuration files needed to interface the application code with the RTXC kernel. Figure 3 lists the RTXCgen generated, application code, and RTXC supplied files.

Figure 3: Example Program File List.

File Resource Color Codes

Application Code

RTXCgen generatted code

RTXC Supplied Code

File *.c – code *.h – header *.s34 - asm	C O D E	Description C*.c files are used by RTXC for kernel initialization. Include the C*.h header files in your application files.
AD.C		Application code file, contains all defined tasks and functions.
AD.H		Application code header file.
CClock		RTI timer setup.
Clka0dr		Clock driver for A0 Timer, used for RTI.
Cmbox		Mailbox definition code.
Cpart		Memory partition definition code.
Cqueue		Queue definition.
Cres		I/O resource definitions.
Csema		Semaphore definitions.
Cstartup		Startup file for system.
Ctask		Task definition code.
Cvtdate		Date/time functions.
Format		String handing routines.
Example		Configuration files, linker output.
Initvect		Initiates vectors to “NO-OPS”.
Iom16c		SFR Address for hardware configuration.
Isrs		Interrupt handlers.
Printl		Print functions.
RTXCbug		Kernel debugger.
Rtxcmain		System initialization and Kernel startup.
Sio0drv		Uart 0 driver.
Uitoa		Function for terminal support.
Ultoa		Function for terminal support.

The example program has 5 tasks. AD.C contains the application code written for this example. Figure 4 table lists all of the tasks and functions contained within AD.C.

Figure 4: Example program task/function descriptions table.

Tasks/Function of AD.C	Task Times(T) and [1]Priorities (P)	Descriptions
Flash LED (TASK)	T= 1000 ms P = 4	This task is timed on 1 second interval. LED2 located on MSA0600 board is flashed on for 1 second then off for 1 second. Flashing an LED on the system is a good indication that the kernel is running.
ADLED (TASK)	T = 500 or 100 ms P = 5	Dependent on the state that is set by the Monitor Push Buttons task, this task will determine the Analog to Digital value or count up. Depending on the button pressed the Analog to Digital resolution may be 8 or 10 bits. When sampling the Analog to Digital value, this task runs every 100 milliseconds. The value on the LED display will increase by 1 every 500 milliseconds when the Count-Up function is active.
Update Display (TASK)	T = 5 ms P = 6	This task runs every 5 milliseconds to display one digit on the MTK7704B LED display. The display is updated 50 times a second.
Monitor Push Buttons (TASK)	T = 50 ms P = 7	Every 50 milliseconds this task checks to see if ether SW1 and/or SW2 button are pressed on the MTK7704B I/O board. The function of the LED display is changed based on the buttons pushed.
Echo Character (TASK)	T = “Uses Semaphore” P = 8	This task runs every time that a character is received on UART0 communications port. A semaphore triggers it. The task sends the received character to UART0 transmit buffer.
Do It (Function)	NA	Called by ADLED task, this functions just calls two other functions. The two functions are Convert and BinToBCD.
Bin To BCD (Function)	NA	This function converts hexadecimal value to Binary Coded Decimal. The converted value is store in the LED_digit array.
Convert (Function)	NA	This function when called by ADLED performs an Analog to Digital conversion using the onboard AD converter of the M16C.
Display (Function)	NA	This function takes the values stored in LED_digit array and displays them on the MTK7704B LED display.
Initial (Function)	NA	Called by RTXCmain, this function initializes the hardware peripherals of the M16C.

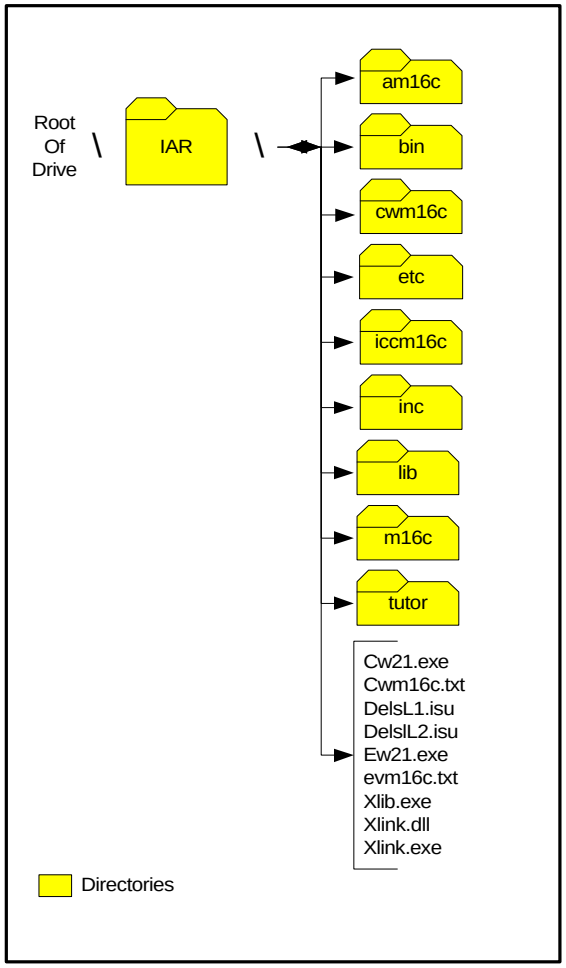
[1] The tasks provided in the RTXC distribution have higher priorities.

5. INSTALLATION OF THE EXAMPLE PROGRAM

Before installing the example program, both RTXC and Embedded Workbench must be installed per manufacture's requirements. After installing Embedded Workbench, modify the directory tree per figure 3.

Figure 5: IAR Revised Directory Tree

Building the Example Program



(RTXC_AD/LED)

To build the example program for the 10 MHz starter kit, follow the instructions listed in section 4.3.1. To build the example programs for the 10 MHz emulator pods use the instructions listed in section 4.3.2. Use section 4.3.3 to build the example program for the 16M Hz M16C/62 (M30620T-RPD-E) group emulator.

Building for 10M Hz Starter Kit (MSA0600 and MTK7704B)

1. Start IAR's Embedded Workbench.
2. On the Menu, click File then Open.
3. In the Open file box, change "Files of Type" to "Project Files (*.prj)."
4. Use the browse features of the Open box to locate one of the installed projects. Click OK after choosing a project.
5. Open the RTXCAPI.H file and change the #define variable SYS_CLOCK_TIME to 0. Save and close RTXCAPI.H file.
6. Right click on Debug on the project window and select OPTIONS. With the Options for Target Debug windows open, click on XLINK. Click on the Include tab and change or verify that "KBD30.XCL" is listed as the "XCL File Name." Verify for each of the tools the "Include" directories. Click OK.
7. On the Menu select Project then click on "Build All."
8. Use the KDB30 Debugger to load the "example.x30" program to the Starter Kit. See section 6 for instructions on running the example program.

Building for 10 MHz M16C Emulator Pod.

Follow steps 1 through 7 of section 5.1. Change or verify that "LNKM16C.XCL" is listed for "XCL File Name." Use the PD30 Debugger to load the "example.x30" file to the Emulator.

Building for the 16 MHz M16C.62 Group Emulator

Follow steps 1 through 7 as listed in section 5.1 with the following exceptions.

1. Change #define variable SYS_CLOCK_TIME to 1.
2. Change or verify the LNKM16C.XCL is listed for the "XCL File Name."

Use the PD30 debugger to load the "example.x30" file to the emulator.

Running the Example Program

To enable RTXC debug on the RTXC_AD/LED demo, connect the Serial Communication port of the MTK7704 I/O board to a PC running HyperTerminal. Set HyperTerminal communications to 4800 Baud, Data, 1 Stop Bit, and No Handshaking. All characters when typed at the HyperTerminal keyboard will be echoed back. Typing a "!" will stop

program execution and cause the system to enter the RTXC debugger.

The RTXC_AD/LED program has several functions dependent on which button is pressed on the MTK7704B I/O board. Pressing both SW1 and SW54 at the same time will cause the LED display to count up starting with 0. Pressing SW1 will cause the system to do a 10-bit A/D read. Pressing SW4 will cause the system to do an 8-bit AD read. The system writes these AD values to the 4 digit LED display.

6. SOURCE CODE

```
/*****************************************************************
*    Copyright,1997, 1998, 1999
*        Mitsubishi Semiconductor America, Inc.
*        MITSUBISHI ELECTRIC CORPORATION AND
*       MITSUBISHI ELECTRIC SEMICONDUCTOR SOFTWARE CORPORATION
*
*****************************************************************
*    M16C 8/10 bit A/D converter program
*
*    This program demonstrates both 8 bit and 10 bit A/D
*    conversions using one shot mode and expanded analog input
*    pin ANEX1. The result of the A/D conversion is displayed on
*    a four digit LED display.
*    An 8 bit sample is taken whenever switch 4 is pressed and a
*    10 bit sample is taken whenever switch 1 is pressed.
*    The result is displayed on LED display and is updated every
*    1 ms.
*
*    written by D. Cocca
*    version 1.00
*
*     Modified for ESP's RTXC(Real Time OS in C)
*     By:    Bruce A. Embry
*     Version 2.00
*     Note:
*        This program was written for the M16C Starter Kit
*         (MSA0600 rev. D) and MTK7704B I/O board.
*
*****************************************************************/
/*    Prototype declarations  */
#include "c:\rtxc\sysopts.h"
#include "intrm16c.h"
#include "iom16c.h"
#include "rtxcapi.h"
#include "enable.h"
#include "periphal.h"
#include "typedef.h"
#include "rtxstruc.h"
#include "cclock.h"   /* CLKTICK  */
#include "cqueue.h"   /* SIO0OQ   */
#include "cres.h"     /* SIO0RES  */
#include "csema.h"    /* DEMOSEM0, DEMOSEM1 */

#include <stdio.h>
#include <math.h>
```

```
#ifdef KS_USER_INTERRUPT  /* { */
extern void isvcksuserint(void);
extern int ksuserintrpt(void *);
extern void setvect( unsigned long far *, unsigned short, unsigned long);
#endif  /* } KS_USER_INTERRUPT */

/************************************************************************
*       Function : main()
*       program section
************************************************************************/


#define AD_C
void adled(void);
void initial(void);
void doit(void);
void display(void);
unsigned int convert(void);
#include "ad.h"
void adled   (void);
void udisplay(void);
void BinToBCD(char *BCD_V, int value);
void echochar(void);

#define SELFTASK ((TASK)0)
/* data structures used by BinToBCD function. */
const char LED_data[10] = {0XC0,0XF9,0XA4,0XB0,0X99,
                    0X92,0X82,0XF8,0X80,0X98};    /* 0 - 9 */
const char LED[4] = {0xEF, 0xDF, 0xBF, 0x7F};

char TURN_LED_ON[4] = {0x01, 0x00, 0x00, 0x00};
char LED_digit[4];

#pragma ROM        LED_data
#pragma ROM        LED
int my_count=0;
char place;
char count, disp_digit;
char demo_state = 0;
#define COUNT_UP     0
#define READ_8BIT    1
#define READ_10BIT   2
int ad_value;        /* stored AD reading */

/*-------------------------------------------------------------------------------**
** Task:               Flash LED                                                 **
** Task name:          fla_led                                                   **
** Desc:               Flashes Led #2 on MSA0600                                 **
** RTXC Servers Used:  KS_delay                                                  **
** Task Time           1000 ms                                                   **
**-------------------------------------------------------------------------------*/
ks_nosaveregs  void fla_led (void)
{
static int LED_STATE;
LED_STATE = 0;
while(1)
    {
    if(LED_STATE){
        LED_STATE = 0;
        p8_0 = 0x1;
        }
    else{
        LED_STATE = 1;
        p8_0 = 0x0;
        }
    KS_delay(SELFTASK,  1000 / CLKTICK);
    }
}
```

```
/*-------------------------------------------------------------------------------------------**
** Task:              Monitor Push Buttons                                                   **
**                                                                                           **
** Task name:  p_button                                                                      **
**                                                                                           **
** Desc:              Monitors Switch 1 and 4 of the I/O board MTK7704B.                     **
**             The state of the system is dependent on the sequance                          **
**             and buttons that are pressed.  If SW1 and SW4 are pushed                      **
**             at the same time, the system will count up.  If SW1 is                        **
**             depressed then the system will perform a 10-Bit anolog to digital             **
**             conversion.  If SW4 is depressed, the system will perform                     **
**             continuous 8-Bit read.                                                        **
**                                                                                           **
** RTXC Services Used: KS_delay                                                              **
** Task Time:  50 ms                                                                         **
**-------------------------------------------------------------------------------------------*/
ks_nosaveregs  void p_button(void)
{
    while (1)
    {
       if  ((p9_7 == 0)&(p9_0 == 0)){
            if(demo_state != COUNT_UP){
                my_count = 0;
                demo_state = COUNT_UP;
                }
            }
        else if (p9_7 == 0)      /*  test for switch 4 pressed */
           demo_state = READ_8BIT;
        else if (p9_0 == 0)      /*  test for switch 1 pressed */
           demo_state = READ_10BIT;
       KS_delay(SELFTASK,  50 / CLKTICK);
    }
}
/*-------------------------------------------------------------------------------------------**
** Task:              Analog to Digital and Calculate LED Display Value                      **
**                                                                                           **
** Task name:  adled                                                                         **
**                                                                                           **
** Desc:              Dependent on state task either perform a count up or analog to         **
**             digital conversion.    After performing one of functions, the task         **
**             convert the value to BCD.  The BCD value is displayed by Update Display    **
**             Task.                                                                         **
**                                                                                           **
** RTXC Servers Used: KS_delay                                                               **
** Task Time                          500 ms or 100 ms (seee code. )                         **
**-------------------------------------------------------------------------------------------*/
void adled(void)
{
     my_count = 0;
    demo_state = COUNT_UP;
    while (1)
       {
       /* For this demo, we shell use RTXC timers to provide task control.  */
       switch(demo_state){
               case COUNT_UP:
                   BinToBCD(&LED_digit[0], my_count);
                   if(my_count++ > 3000)
                       my_count=1;
                   KS_delay(SELFTASK,  500 / CLKTICK);
                   break;
               case READ_8BIT:
                   bits = 0;      /*  8bit A/D*/
                   doit();
                   KS_delay(SELFTASK,  100 / CLKTICK);
                   break;
               case READ_10BIT:
                default:
                   bits = 1;      /*  10 bit A/D */
                   doit();
                   KS_delay(SELFTASK,  100 / CLKTICK);
                   break;
               }
       }
}
```

```
/*-------------------------------------------------------------------------------------------**
** Task:               Update Display                                                        **
** Task name:          udisplay                                                              **
** Desc:               Calls display function. Display function writes BCD values            **
**                     to the LED display of the MTK7704B I/O board.                         **
** RTXC Sevices Used:  KS_delay                                                              **
** Task Time           5 ms                                                                  **
**-------------------------------------------------------------------------------------------*/

void udisplay(void)
{
    while(1){
         KS_delay(SELFTASK,  5 / CLKTICK);
         display();
         }
}

/*-------------------------------------------------------------------------------------------**
** Function:           Do It                                                                 **
**                                                                                           **
** Func, name:         doit                                                                  **
**                                                                                           **
** Desc.               Calls convert functin to perfrom Analog to Digital                    **
**                     conversion and then calls BinToBCD. BinToBCD converts                 **
**                     the AD value to BCD(Binary Coded Decimal).                            **
**                                                                                           **
** Functions Called:   convert                                                               **
**                     BinToBCD                                                              **
**-------------------------------------------------------------------------------------------*/
void doit(void)
{
       ad_value = convert();        /* do A/D conversion */
       BinToBCD(&LED_digit[0], ad_value);
}
/*-------------------------------------------------------------------------------------------**
** Function:           Binary To Binary Coded Decimal                                        **
** Function name:      BinToBCD                                                              **
** Desc:               Converts integer value to Binary Coded                                **
**                     Decimal.                                                              **
** Data Structures:    const char LED_data[10] = {0XC0,0XF9,0XA4,0XB0,0X99,                  **
**                      0X92,0X82,0XF8,0X80,0X98};     0 - 9                                 **
**                     const char LED[4] = {0xEF, 0xDF, 0xBF, 0x7F};                         **
**                                                                                           **
**                     char TURN_LED_ON[4] = {0x01, 0x00, 0x00, 0x00};                       **
**                     const int POW_10[4]={1,10,100,1000};                                  **
**                     TURN_LED_ON                                                           **
**                     char LED_digit[4];                                                    **
**                     LED_data                                                              **
**-------------------------------------------------------------------------------------------*/

void BinToBCD(char *BCD_V, int value)
{
char BOLLEAN_TEST;
int loop, Nloop, N_pow, Bin_value;
const int POW_10[4]={1,10,100,1000};
BOLLEAN_TEST = 1;
Bin_value = value;
 for( loop = 4; loop > 0; loop--) {
    Nloop = loop - 1;
    N_pow = POW_10[Nloop];
    *(BCD_V + Nloop) = Bin_value / N_pow;
    Bin_value = Bin_value %  N_pow;
    if(BOLLEAN_TEST){
        if (*(BCD_V + Nloop) == 0)
            TURN_LED_ON[Nloop] = 0;
        else {
            TURN_LED_ON[Nloop] = 1;
            BOLLEAN_TEST = 0;
            }
        }
    else
         TURN_LED_ON[Nloop] = 1;
    }
    TURN_LED_ON[0] = 1;

}
```

```
/*-------------------------------------------------------------------------------------------**
** Function:   Convert Analog to Digital                                                     **
** Function name:     convert                                                                **
** Desc:              Converts analog input from I/O board to disgital value.                **
**-------------------------------------------------------------------------------------------*/
unsigned int convert(void)
{
unsigned int ad_value;
    vcut = 1;            /*  connect Vref */
    adst = 1;            /* start conversion */
    do{
    }while (adst == 1);  /* wait til conversion done */
    ad_value = ad1;      /* store A/D value */
    vcut = 0;            /* disconnect Vref (save power) */
    return(ad_value);
}
/*-------------------------------------------------------------------------------------------**
** Function:          Display value to LED                                                   **
** Function name:     display                                                                **
** Desc:              Takes value stored in LED_data array and displays it                   **
**                    on the LED display of the I/O board. This is the companion             **
**                    function to BinToBCD.                                                  **
** Data Structures:   TURN_LED_ON                                                            **
**                    LED_digit                                                              **
**                    LED_data                                                               **
**                    LED                                                                    **
**-------------------------------------------------------------------------------------------*/
void display()
{
    char i;

    p6 = 0xFF;               /* turn off LED display */
    ++place;                 /* increment place flag */
    if (place >= 4)
        place = 0;
    if(TURN_LED_ON[place]){
        i = LED_digit[place];
        p7 = LED_data[i];
        p6 = LED[place];
        }
}
/*-------------------------------------------------------------------------------------------**
** Function:          Initialize Hardware                                                    **
** Function name:     initial                                                                **
** Desc:              Initializes hardware for Analog to Digital Conversion.                 **
**                    Function is called from RTXCmain.                                      **
**-------------------------------------------------------------------------------------------*/
void initial(void)
{
/*    Port initialization */
    pd6 = 0xf8;            /* output port
                              P64~P67 control LED digits 1~4 */
    pd7 = 0xff;            /* output port
                              P70~P77 drive 7 segment digit */
    pd8 = 0x00;            /* P8 input */
    pd8_0 = 0x1;           /* change p8_0 to output. */
    pd9 = 0x00;            /* P9 input port */
                         /* ANEX1 on P96 */
                         /* switch 4 on P97 */
/*    A/D initialization */
    adcon2 = 1;            /*sample & hold*/
    adcon0 = 1;            /*select ANEX1*/
/** 10 bit A/D conversions on ANEX are only +/- 7 lsb accurate **/
    adcon1 = 0x98;        /*fAD clk, 10 bit, one shot mode*/
}
```

```
/*-------------------------------------------------------------------------------------**
** Task:               Echo Charactor                                                  **
** Task name:          echochar                                                        **
** Desc.:              Receives charactor from serial port and echos, and transmits    **
**                     it back to the terminal device.                                 **
** RTXC services used: KS_user, KS_dequeuew, KS_lockW, KS_enqueuew, KS_unlock          **
**-------------------------------------------------------------------------------------*/
void echochar(void)
{
char *c;
#ifdef KS_USER_INTERRUPT  /* { */
   setvect( (unsigned long far *)INTB_ADDRESS, KSUSER_TRAPNUM, (unsigned long) isvcksuserint);
   KS_user(ksuserintrpt, (void *)0 );
#endif  /* } KS_USER_INTERRUPT */
 for(;;){
   KS_dequeuew(SIO0IQ,&c);
   KS_lockw(SIO0RES);
   KS_enqueuew(SIO0OQ,&c);
   KS_unlock(SIO0RES);
   }
}
```