E&CE 254 Lab1 Tutorial: Spring 2012
Keil IDE and RL-RTX

Objective

This tutorial is to introduce the Keil μVision4 IDE and Keil RTX RTOS. Students will experiment with inter-process communication by using the RTX task, mailbox and time management library function calls on MCB1768 evaluation board. After this lab, students will have a good understanding of the following:

- How to create a μVision project.
- How to configure μVision for RTX
- How to view and control the RTX operation in debug mode
  - How to use the Watch Window and Memory Window
  - How to set up Hardware breakpoints and Watch points
  - How to use the Performance Analyzer (PA) to see where the program spends its time
  - How to use the Execution Profiling (EP) to display how many times a function has been called and the total time spent in a function.
  - How to use the RTX Task and System Window
  - How to use the RTX Event View window

Creating Your First Hello World Application

The Keil μVision4 IDE getting started guide (posted on the lab web site) is the first documentation you may want to read. The IDE online help contains MDK-ARM Primer and μVision IDE Users Guide. These two documents contain more detailed examples of how to use the IDE. To create a simple application that displays "Hello World!" on the LCD screen, follow the steps below:

1. Create a folder for your new project on your computer.
2. Copy all file under GLCD folder to your project folder
3. Copy `system_LPC17xx.c` under Startup folder to your project folder
4. Creating a New Project in μVision
   - Project $\rightarrow$ New μVision Project
   - NXP(Founded by Philips) $\rightarrow$ LPC1768 $\rightarrow$ Answer "Yes" to copy the startup files
5. Rename "Target 1" to "Hello World".
6. Rename "Source Group 1" to "Startup Code"
7. Add a new source group and name it "Source Code".
8. Add all LCD C supporting files to the "Source Code" source group
9. Create a new `main.c` file (see Listing 1 to print out a string to the LCD
10. Add the `main.c` file to the "Source Code" source group
11. Build by click the "Rebuild All" button
12. Download by clicking the "Load" button

13. Press the “Reset” button to run the application

```c
#include <LPC17xx.h>
#include "GLCD.h"

int main()
{
   SystemInit();
   GLCD_Init();
   GLCD_Clear(Yellow);
   GLCD_DisplayString(0, 0, 1, "Hello World!");
}
```

Listing 1: Hello World main.c

Creating an RL-RTX Application

The RL-RTX is one of the components of RL-ARM, the RealView Real-Time Library (RL-ARM). The RTX kernel is a real time operating system (RTOS) that enables one to create applications that simultaneously perform multiple functions or tasks (statically created processes). Tasks can be assigned execution priorities. The RTX kernel uses the execution priorities to select the next task to run (preemptive scheduling). It provides additional functions for inter-task communication, memory management and peripheral management.

RTX programs are written using standard C constructs and compiled with the RealView Compiler. The `RTL.h` header file defines the RTX functions and macros that allow you to easily declare tasks and access all RTOS features.

To create anapplication that uses RL-RTX library, follow the steps below:

1. Create a new project for NXP LPC1768 processor.
2. Copy the `RTX_Conf_CM.c (C:\Software\Keil\ARM\Startup)` to the project folder and add it to your project.
3. Open the `RTX_Conf_CM.c` under the Configuration Wizard and set the CPU to 100000000. You need to set the total number of tasks the system can manage. The default value is 6 and normally is sufficient.
4. Open up the Target Option window and activate the “Target” tab. Choose “RTX Kernel” as the operating system (default is set to None). Click Use MicroLIB in the code- generation window.
5. Now you can start to develop an RTX application. Start with the `os_tsk_*()` function reference which is within the μ`Vision` Help File.

Listing 2 shows you a simiple RTX application that lights up the first LED when it starts and then displays the “Hello World!” string on the LCD screen. Then the system creates an initialization task named `init` which spawns three tasks `task1`, `task2`, and `task3`. The `task1` displays digit `0-9` in a round robin fashion. The `task2` makes a LED blinks every one second. The `task3` keeps incrementing a global counter, which we can examine under the debug mode.

```c
/**
 * @brief: A Simple RL-RTX application to blink a LED and
 *         display 0-9 in a round robin fashion on LCD
 */

#include <LPC17xx.h>
#include <RTL.h>
#include "GLCD.h"
```

```c
#include "LED.h"

int g_counter=0; // a global counter

/**
 * @brief: displays 0-9 in a round robin fashion.
 */
__task void task1 (void)
{
   int i = 0;
   GLCD_DisplayString(3, 0, 1, "Task␣1:");

   for (;;) {
      GLCD_DisplayChar(3, 7, 1, i+'0' );
      os_dly_wait(100);
      if (i++ == 9) {
         i=0;
      }
   }
}

/**
 * @brief: toggles LED #7 at P2.6 every second
 */
__task void task2 (void)
{
   for (;;) {
      LED_on(7);
      os_dly_wait(60);
      LED_off(7);
      os_dly_wait(40);
   }

}

/**
 * @brief: A dummy task that keeps incrementing a counter
 */

__task void task3 (void)
{
   for(;;) {
      g_counter++;
   }
}

/**
 * @brief: Initialization task that spawns all other tasks.
 */
__task void init(void) {
   os_tsk_create (task1, 1); // task1 at priority 1
   os_tsk_create (task2, 1); // task2 at priority 1
```

```
61    os_tsk_create (task3, 1); // task3 at priority 1
62    os_tsk_delete_self(); // delete itself
63 }
64
65 int main()
66 {
67    SystemInit();          // initialize the LPC17xx MCU
68    LED_init();            // initialize the LED device
69    GLCD_Init();           // initialize the LCD device
70
71    LED_on(0);             // turn on LED #0 at P1.28
72
73    GLCD_Clear(Yellow);    // clear LCD screen with yellow color
74    GLCD_DisplayString(0, 0, 1, "Hello World!");
75                           // display a string at row 0 col 0 with
76                           // font size 1.
77    os_sys_init(init);     // initiate the first task in the RTOS
78 }
```

Listing 2: A Simple RL-RTX Application main.c

View and Control the RTX Operation in Debug Mode [1]

- **Watch window**
  Click View → Watch Windows → Watch 1.
  Highlight the variable you want to watch and drag it to the Name column in the Watch 1 window. You can also just double click or use F2 to add by typing the variable name.
- **Memory window**
  Click View → Memory Windows → Memory 1.
  Highlight the variable and drag it to the Address text field. Add an `&` before the variable name. You can also just type `&variable_name` directly in the text field. Right click in the memory window and pick the data type you want to be displayed in the memory window.
- **Hardware Breakpoints**
  You can set up breakpoints next to the C statement that you want stop at by double clicking the left side of the line numbers
- **Watch Points** (also called Access Breaks)
  You can stop the program when a variable reaches a user defined value.
  Open Debug → Breakpoints (or Ctrl-B). In the Expression text field, enter
  `variable == value` (for example `x==200`)
  and select "Write". Click on "Define" and then "Close" to close the window.
  Run the program and it will stop when the variable reaches the value.
- **Performance Analzyer**
  Click View → Analysis Windows → Performance Analyzer to open it. It is a tool to tell you where your program is spending its time. You can only use this in the simulator with the hardware we have in the lab.
- **Execution Profiling**
  Click Debug → Execution Profiling → Show Time/Calls. It can show how much time has been spent in a function or how many times a function has been called.
- **RTX Tasks and System Window**
  Click Debug → OS Support → RTX Tasks and Systems. This window updates as the RTX runs. You can watch task switching and various system and task related information in this window.

- **RTX Event Viewer**
  Click Debug → OS Support → Event Viewer. Event Viewer provides a graphical representation of how long and when individual tasks run.

Notes

The simulator cannot simulate the LCD.

References

[1] Robert Boys. Keil RTX RTOS: The Easy Way V0.4. 2010.