

|  |  |
| --- | --- |
| **Course Number** | COE 718 |
| **Course Title** | Embedded Systems Design |
| **Semester/Year** | F2020 |
| **Lab No** | 3 |
| **Instructor Name** | Saber Amini |
| **Section No** | 03 |

|  |  |
| --- | --- |
| **Submission Date** | 10/28/2020 |
| **Due Date** | 10/28/2020 |

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **Signature\*** |
| Vatsal Shreekant | 500771363 |  |

*\*By signing above, you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

*www.ryerson.ca/senate/current/pol60.pdf*

**Introduction**

A “thread” in computer science is short for a thread of execution. Threads are a way for a program to divide (termed "split") itself into two or more simultaneously (or pseudo-simultaneously) running tasks.

The osThreadCreate() and osThreadDef() functions will create the threads and set their priorities respectively.

The osKernelInitialize() and osKernelStart() will setup the round-robin scheduling definition for the threads and execute the kernel respectively.

osTimerThread() thread initializes and executes first. This thread is responsible for executing time management functions specified by ARM's RTOS configuration.

The program starts executing from main(), where main() ensures that:

1. The Cortex-M3 system and timers are initialized -SystemInit().
2. The os kernel is initialized for interfacing software to hardware -osKernelInitialize().
3. Creates the threads to execute thread1 and thread2 -Init\_Thread ().
4. Starts the kernel to begin thread switching -osKernelStart().

The Thread\_C thread executes for its round-robin time slice since it is the highest priority. After 15 msec the timer thread forces control to the Thread\_A which has above normal priority and then over to Thread\_D. After 15 msec the timer thread forces control to the Thread\_B which has above below priority and then over to Thread\_E.

**Procedure**

1. Load main.c and Thread.c example project and complete the instructions in the lab manual.
2. Select the following packages under ‘Manage Run-Time Environment’ window and select OK button:

* CMSIS>CORE.
* CMSIS>RTOS(API)>Keil RTX.
* Device > Startup.

1. Modify the ‘Thread.c’ file to implement a round-robin scheduling example using 3 different tasks as listed in the table 1 of the lab manual. The code implemented in Thread.c is listed as follows:

Machine generated alternative text:
C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Thread.c
Page 1
1   
/*----------------------------------------------------------------------------
2   
*      Lab 3: Scheduling Multithreaded Applications with RTX & uVision
3   
* Vatsal Shreekant, student id: 500771363
4   
 *---------------------------------------------------------------------------*/
5   
#include
<stdio.h>
6   
#include
<ctype.h>
7   
#include
<string.h>
8   
#include
<math.h>
9   
#include
"cmsis_os.h"
10   
#include
"LPC17xx.H"
11   
#define
ADDRESS(x)(*((
volatileunsignedlong
*)(x)))
12   
#define
BitBand(x,y)ADDRESS(((
unsignedlong
)(x)&
0xF0000000
)|
0x02000000
|(((
unsigned
long
)(x)&
0x000FFFFF
)<<
5
)|((y)<<
2
))
13   
#define
GPIO1_LED31(*((
volatileunsignedlong
*)
0x233806FC
))
14   
#define
GPIO2_LED2(*((
volatileunsignedlong
*)
0x23380A88
))
15   
#define
__FI
1
16   
17   
18   
19   
double
factor;
20   
double
var_A;
21   
double
var_B;
22   
double
var_C;
23   
double
var_D;
24   
double
var_E;
25   
int
myExponentialCalc(
int
,
int
);
26   
void
delay(
unsignedint
);
27   
volatileunsignedlong
*GPIO2_LED4;
28   
29   
30   
31   
void
threadA(
voidconst
*argument);
// thread function
32   
void
threadB(
voidconst
*argument);
33   
void
threadC(
voidconst
*argument);
34   
void
threadD(
voidconst
*argument);
35   
void
threadE(
voidconst
*argument);
36   
37   
38   
osThreadIdid_Thread_A,id_Thread_B,id_Thread_C,id_Thread_D,id_Thread_E;
// thread id
39   
40   
osThreadDef(threadA,osPriorityAboveNormal,
1
,
0
);
//priority #2
41   
osThreadDef(threadB,osPriorityBelowNormal,
1
,
0
);
//priority #3
42   
osThreadDef(threadC,osPriorityHigh,
1
,
0
);
//priority #1
43   
osThreadDef(threadD,osPriorityAboveNormal,
1
,
0
);
//priority #2
44   
osThreadDef(threadE,osPriorityBelowNormal,
1
,
0
);
//priority #3
45   
46   
47   
int
Init_Thread(
void
){
48   
49   
id_Thread_A=osThreadCreate(osThread(threadA),NULL);
// create 
threads
50   
id_Thread_B=osThreadCreate(osThread(threadB),NULL);
51   
id_Thread_C=osThreadCreate(osThread(threadC),NULL);
52   
id_Thread_D=osThreadCreate(osThread(threadD),NULL);
53   
id_Thread_E=osThreadCreate(osThread(threadE),NULL);
54   
55   
if
(!id_Thread_A)
return
(-
1
);
56   
57   
return
(
0
);
58   
}
59   
60   
61   
void
threadA(
voidconst
*arg){
62   
double
x=
0
;
63   
int
i=
0
;
64   
65   
for
(i=
0
;i<
257
;i++){
66   
x=x+(i+(i+
2
));


Figure 1: Page 1 of Thread.c

Machine generated alternative text:
C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Thread.c
Page 2
67   
var_A=x;
68   
//Function that passes control to the 
next task of the same priority in the ready queue
69   
70   
}
71   
delay(
100
);
72   
73   
}
74   
75   
void
threadB(
voidconst
*arg){
76   
double
x=
0
;
77   
int
i;
78   
int
factor=
1
;
79   
80   
81   
for
(i=
1
;i<
17
;i++){
82   
factor=factor*i;
83   
x=x+((
double
)(myExponentialCalc(
2
,i))/factor);
84   
var_B=x;
85   
86   
osDelay(
1
);
//Function that passes 
control to the next task of the same priority in the ready queue
87   
88   
}
89   
90   
91   
}
92   
93   
void
threadC(
voidconst
*arg){
94   
double
x=
0
;
95   
int
n=
0
;
96   
97   
98   
for
(n=
1
;n<
17
;n++){
99   
x=x+(n+
1
)/n;
100   
var_C=x;
101   
}
102   
103   
}
104   
105   
void
threadD(
voidconst
*arg){
106   
107   
double
x=
0
;
108   
int
m=
0
;
109   
factor=
1
;
110   
111   
for
(m=
0
;m<
6
;m++){
112   
factor=factor*m;
113   
if
(factor==
0
){
114   
factor=
1
;
115   
}
116   
else
{
117   
osDelay(
1
);
//Function that 
passes control to the next task of the same priority in the ready queue
118   
x=x+((
double
)(myExponentialCalc(
5
,m)))/(
double
)factor;
119   
var_D=x;
120   
}
121   
}
122   
123   
}
124   
125   
void
threadE(
voidconst
*arg){
126   
double
x=
0
;
127   
int
p=
0
;
128   
int
radius=
1
;
129   
130   
131   


Figure 2: Page 2 of Thread.c

Machine generated alternative text:
C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Thread.c
Page 3
132   
133   
for
(p=
1
;p<
13
;p++){
134   
x=x+(
3.14
)*((
double
)(myExponentialCalc(radius,
2
)));
135   
var_E=x;
136   
osDelay(
1
);
//Function that passes 
control to the next task of the same priority in the ready queue
137   
138   
}
139   
140   
}
141   
142   
int
myExponentialCalc(
int
x,
int
n)
143   
{
144   
int
i;
145   
int
number=
1
;
146   
147   
for
(i=
0
;i<n;i++)
148   
number*=x;
149   
150   
return
(number);
151   
}
152   
153   
void
delay(
unsignedint
value){
154   
unsignedint
count1=
0
;
155   
unsignedint
count2=
0
;
156   
157   
for
(count1=
0
;count1<value;count1++){
158   
for
(count2=
0
;count2<count1;count2++){
159   
}
160   
}
161   
}


Figure 3: Page 3 of Thread.c

1. Compile project using the build button and start the simulation by selecting the debug button.
2. Select debug mode to analyze performance of the threads using Performance Analyzer and the Event Viewer.

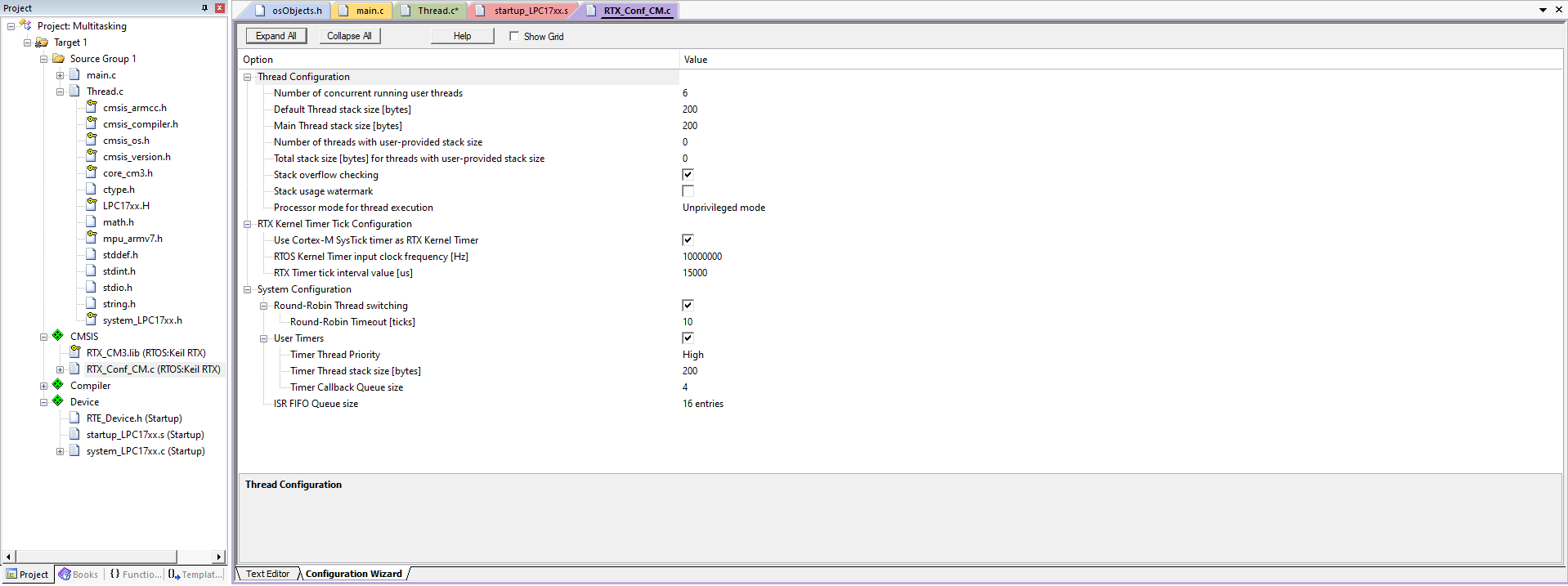


Figure 4: RTX\_Conf\_CM.c Configuration Wizard

**Conclusion**

When comparing and analyzing the results under debug mode, it is evident that the tasks are being prioritized and executed as per the priority thread and this was the initial assumption before implementing the code. Refer to figures 5 and 6 for the results. As per the instructions in the lab manual, the code for the LED was not required. The BitBand() function was used to toggle the pins at Port 2. The results for values A, B, C, D and E in the watch window were also verified against an online calculator and there were no discrepancies.

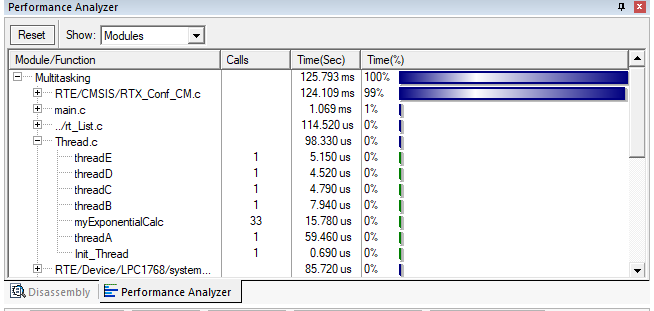


Figure 5: Performance Analyzer Window

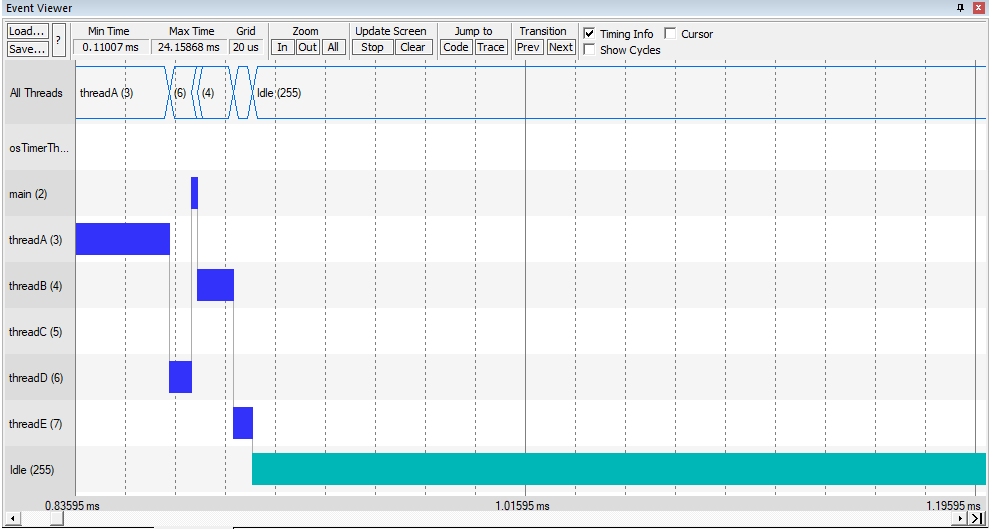


Figure 6: Event Viewer Window

**References**

1. NXP User Manual, https://www.nxp.com/docs/en/user-guide/UM10360.pdf, 2020
2. ARM Keil User Guide, https://www.keil.com/support/man/docs/mcb1700/mcb1700\_intro.htm, 2020