**Introduction**

In this hands-on lab, we will use Kinetis Design Studio to create 4 tasks named Task\_A, Task\_B, Task\_C and Task\_D which enter in an endless loop printing a message in a terminal.

**Resources**

PC running Windows 7 with the following software:

• Kinetis Design Studio (KDS) v2.0.0

• MQX for KSDK v1.1.0

Hardware:

• FRDM-K64F

1. **Implementation**
   1. By now a ‘*New MQX RTOS for KSDK Project Wizard’* does not exists. For this Lab use **‘MQX for KSDK Lab 2 - Base Project’**.

**Note:** If you want to know how to create a base project see the **‘Appendix’** in this document.

* 1. Define Task\_A in the Task\_Template\_Struct. Use the parameters below for this task.
  2. Task Number 2
  3. Task name Task\_A
  4. Stack size 1500
  5. Priority 10
  6. NOT Auto Start.
  7. Task\_Template\_Struct should look like this.

TASK\_TEMPLATE\_STRUCT MQX\_template\_list[] =

{

/\* Task number, Entry point, Stack, Pri, String, Auto? \*/

{MAIN\_TASK, Main\_task, 1500, 9, "main", MQX\_AUTO\_START\_TASK},

{2, Task\_A, 1500, 10, "Task A", 0 },

{0, 0, 0, 0, 0, 0, }

};

* 1. Add a macro in main.c to replace the task number in the Task\_Template\_Struct. For example.

**#define** TASK\_A 2

TASK\_TEMPLATE\_STRUCT MQX\_template\_list[] =

{

/\* Task number, Entry point, Stack, Pri, String, Auto? \*/

{MAIN\_TASK, Main\_task, 1500, 9, "main",MQX\_AUTO\_START\_TASK},

{TASK\_A, Task\_A, 1500, 10, "Task A", 0 },

{0, 0, 0, 0, 0, 0, }

};

* 1. According to the instructions the task’s body should look as shown below. Use function \_**time\_delay()** to make a 500mS delay to be able to see output in the UART and let other tasks run. See **Freescale MQX™ RTOS Reference Manual.pdf** for details about time delay function and its parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc****.*

**void** **Task\_A**(uint32\_t initial\_data)

{

**while**(1){

printf("\n This is Task A \n");

**\_time\_delay**(500);

}

}

* 1. Add the function prototype in main.c.

**void** **Task\_A** (uint32\_t);

* 1. Repeat steps 1 to 5 for Task\_B, Task\_C and Task\_D.
  2. In Main\_Task create one instance of Task\_A, Task\_B, Task\_C and Task\_D using **\_task\_create**().You can see **Freescale MQX™ RTOS Reference Manual.pdf** for details about tasks functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc****.*
  3. The task id number must be saved in a variable of type \_task\_id.
  4. A message to a terminal must be sent whether the task was created successfully or not and show the error number in case it fails.
  5. After creating all tasks, Main\_Task must remain blocked using **\_task\_block**() function.

\_task\_id ta, tb, tc, td;

**void** **Main\_task**(uint\_32 initial\_data)

{

printf("\n Using Tasks \n");

ta = **\_task\_create**(0,TASK\_A,0);

**if** (ta == MQX\_NULL\_TASK\_ID)

{

printf("Task\_A could not be created. Error: 0x%X \n", ta);

}

**else**

{

printf("Task\_A was created\n");

}

tb = **\_task\_create**(0,TASK\_B,0);

**if** (tb == MQX\_NULL\_TASK\_ID)

{

printf("Task\_B could not be created. Error: 0x%X \n", tb);

}

**else**

{

printf("Task\_C was created\n");

}

tc = **\_task\_create**(0,TASK\_C,0);

**if** (tc == MQX\_NULL\_TASK\_ID)

{

printf("Task\_C could not be created. Error: 0x%X \n", tc);

}

**else**

{

printf("Task\_C was created\n");

}

td = **\_task\_create**(0,TASK\_D,0);

**if** (td == MQX\_NULL\_TASK\_ID)

{

printf("Task\_D could not be created. Error: 0x%X \n", td);

}

**else**

{

printf("Task\_D was created\n");

}

**\_task\_block**();

}

* 1. Debug the application. What output do you get in the terminal?
  2. Edit the application.
     1. Main\_Task must create Task\_A and remain blocked using **\_task\_block**() function.
     2. Task\_A must create Task\_B and remain blocked using **\_task\_block**().
     3. Task\_B must create Task\_C and destroy itself using **\_task\_destroy**(MQX\_NULL\_TASK\_ID).
     4. Task\_C must create Task\_D and destroy itself using **\_task\_destroy**(MQX\_NULL\_TASK\_ID).
     5. Task\_D must put Task\_A in ready state using **\_task\_ready**(). You will need to use **\_task\_get\_td**() to get a pointer to the task descriptor which is the parameter that **\_task\_ready**() needs.
     6. Finally Task\_D must destroy itself using **\_task\_destroy**(MQX\_NULL\_TASK\_ID).

See **Freescale MQX™ RTOS Reference Manual.pdf** for details about tasks functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc***.

* 1. Debug the application. What output do you get?

**That now completes Lab 3.**

1. **Code**

**#include** <mqx.h>

**#include** <bsp.h>

**#define** MAIN\_TASK 1

**#define** TASK\_A 2

**#define** TASK\_B 3

**#define** TASK\_C 4

**#define** TASK\_D 5

**void** **Main\_task**(uint\_32);

**void** **Task\_A** (uint\_32);

**void** **Task\_B** (uint\_32);

**void** **Task\_C** (uint\_32);

**void** **Task\_D** (uint\_32);

TASK\_TEMPLATE\_STRUCT MQX\_template\_list[] =

{

/\* Task number, Entry point, Stack, Pri, String, Auto? \*/

{MAIN\_TASK, Main\_task, 1500, 9, "main", MQX\_AUTO\_START\_TASK},

{TASK\_A, Task\_A, 1500, 10, "Task A", 0},

{TASK\_B, Task\_B, 1500, 10, "Task B", 0},

{TASK\_C, Task\_C, 1500, 10, "Task C", 0},

{TASK\_D, Task\_D, 1500, 10, "Task D", 0},

{0, 0, 0, 0, 0, 0, }

};

\_task\_id ta, tb, tc, td;

**void** **Main\_task**(uint\_32 initial\_data)

{

printf("\n Using Tasks \n");

ta = **\_task\_create**(0,TASK\_A,0);

**if** (ta == MQX\_NULL\_TASK\_ID)

{

printf("Task\_A could not be created. Error: 0x%X \n", ta);

}

**else**

{

printf("Task\_A was created\n");

}

**\_task\_block**();

}

**void** **Task\_A**(uint\_32 initial\_data)

{

\_task\_id tb;

**while**(1){

printf("\nThis is Task A \n");

**\_time\_delay**(500);

tb = **\_task\_create**(0,TASK\_B,0);

**if** (tb == MQX\_NULL\_TASK\_ID)

{

printf("Task\_B could not be created. Error: 0x%X \n", tb);

}

**else**

{

printf("Task\_B was created\n");

}

**\_task\_block**();

}

}

**void** **Task\_B**(uint\_32 initial\_data)

{

\_task\_id tc;

**while**(1){

printf("\nThis is Task B \n");

**\_time\_delay**(500);

tc = **\_task\_create**(0,TASK\_C,0);

**if** (tc == MQX\_NULL\_TASK\_ID)

{

printf("Task\_C could not be created. Error: 0x%X \n", tc);

}

**else**

{

printf("Task\_C was created\n");

}

**\_task\_destroy**(MQX\_NULL\_TASK\_ID);

}

}

**void** **Task\_C**(uint\_32 initial\_data)

{

\_task\_id td;

**while**(1){

printf("\nThis is Task C \n");

**\_time\_delay**(500);

td = **\_task\_create**(0,TASK\_D,0);

**if** (td == MQX\_NULL\_TASK\_ID)

{

printf("Task\_D could not be created. Error: 0x%X \n", td);

}

**else**

{

printf("Task\_D was created\n");

}

**\_task\_destroy**(MQX\_NULL\_TASK\_ID);

}

}

**void** **Task\_D**(uint\_32 initial\_data)

{

**while**(1){

printf("\nThis is Task D \n");

**\_time\_delay**(500);

**\_task\_ready**(**\_task\_get\_td**(ta));

**\_task\_destroy**(MQX\_NULL\_TASK\_ID);

}

}

/\* EOF \*/

1. **Appendix**

To create a new project you will have to copy/paste the hello\_frdmk64f (project imported in lab 1) into your workspace and rename it. When you copy/paste a project into your workspace, your sources folder is linked to the original one, so if you change anything on your new project it will be changed in the original one too. To avoid this problem you will have to erase the sources folder of your new project and create a new one with the same name (this one will not be linked to the previous project).

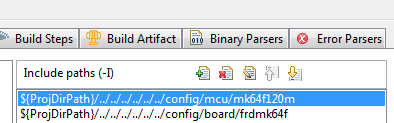
* 1. If your workspace is not the same that the hello\_frdmk64f project you will have to modify the paths of the project. To do this you will have to go to **Project->Properties->C/C++ Build ->Settings**
  2. You will have to change the every single path of the program, to do this you will have to look for the right path the program is referring. For example, in the

**Cross ARM GNU Assembler -> Includes** the first path will be **${ProjDirPath}/../../../../../../config/mcu/mk64f120m**

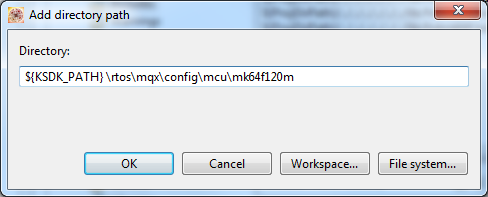
You will have to look into the installation folder **(C:\Freescale\KSDK\_1.1.0)** the last directories of this path. In this case you will find it at

**C:\Freescale\KSDK\_1.1.0\rtos\mqx\config\mcu\mk64f120m**.

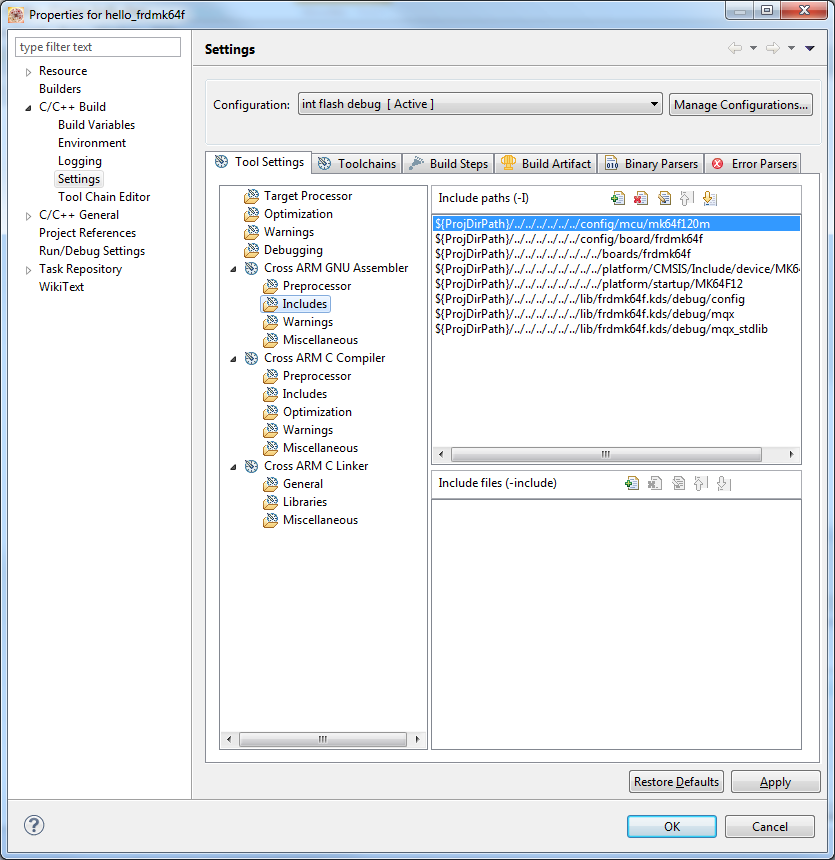
* 1. Once you get the correct path you will have to click the add option.



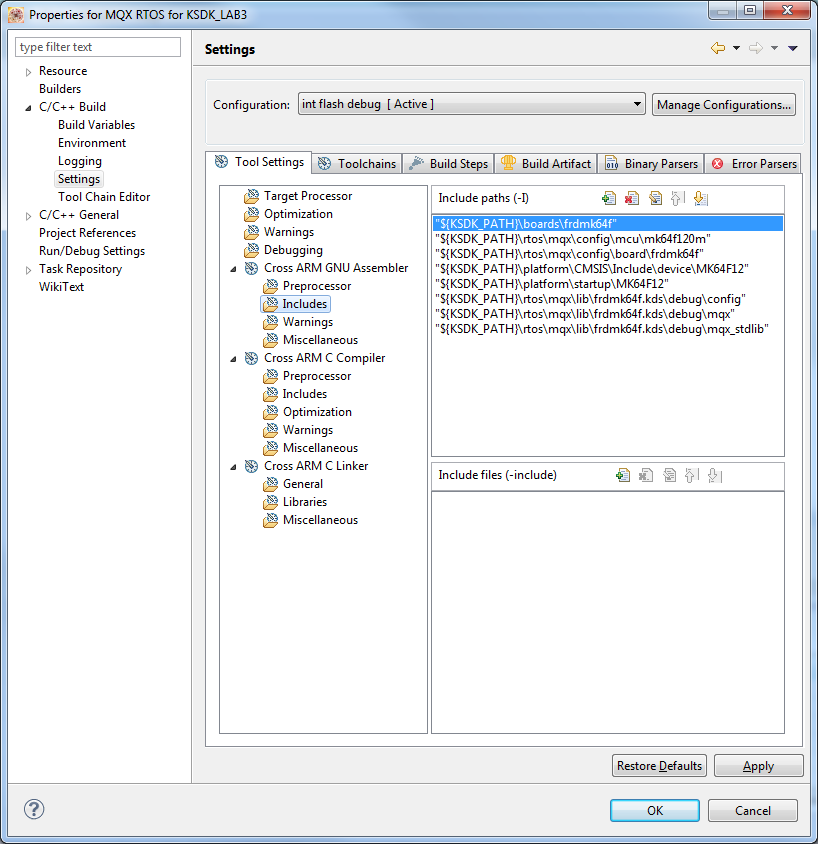
* 1. This window will appear, here you will have to add the correct path and substitute the **C:\Freescale\KSDK\_1.1.0** with **${KSDK\_PATH}** and then click ok.(Note: is important to have defined KSDK\_PATH as an environment variable)



* 1. Once you have added this new path you can erase the other one. You will have to repeat this steps in the next folders.



* 1. Your new paths will have to see as follows.



* 1. Once you change every single path you will be able to compile and debug your new project. All this process is necessary because the MQX RTOS for KDS wizard is still in development.