Freescale Semiconductor

Creating a New USB project with KSDK and Processor Expert support in KDS

By: Technical Information Center

Developing an USB application can involve to include some extra features besides the ones shown in USB examples for Kinetis SDK.

You could use Kinetis SDK API functions to add new features to your USB project but there is also an option on adding processor expert support or combine both of them.

This documents explains how to create a new USB example project for KSDK 1.2.0 and add Processor Expert to it.

For this document, Freedom K64F and USB HID bare metal example will be used.

1 Requirements.

1.1 Install KDS 3.0.0 (Kinetis Design Studio), you can download from www.freescale.com/kds

1.2 Install KSDK 1.2.0 (Kinetis Software Development Kit), you can download from www.freescale.com/ksdk

1.3 Install ***'KSDK_1.2.0_Eclipse_Update'*** you can find the update in ***'C:\Freescale\KSDK_1.2.0\tools\eclipse_update'***. The instruction to make the updates are described in chapter 2 and 2.1 of ***'C:\Freescale\KSDK_1.2.0\doc\rtos\mqx\MQX RTOS IDE Guides\MQX-KSDK-KDS-Getting-Started.pdf'***

1.4 It is necessary to build the following libraries:

1.4.1 Platform Library for MQX, '***ksdk_platform_lib_K64F12.a'***

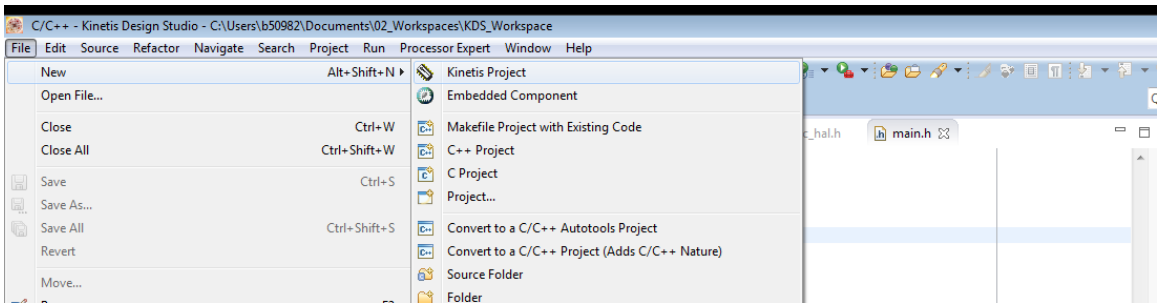
- This project is located in ***${KSDK_PATH}/lib/ksdk_platform_lib/kds/K64F12***

1.4.2 MQX Library, ***'usbd_sdk_frdmk64f_bm_frdmk64f.a'***

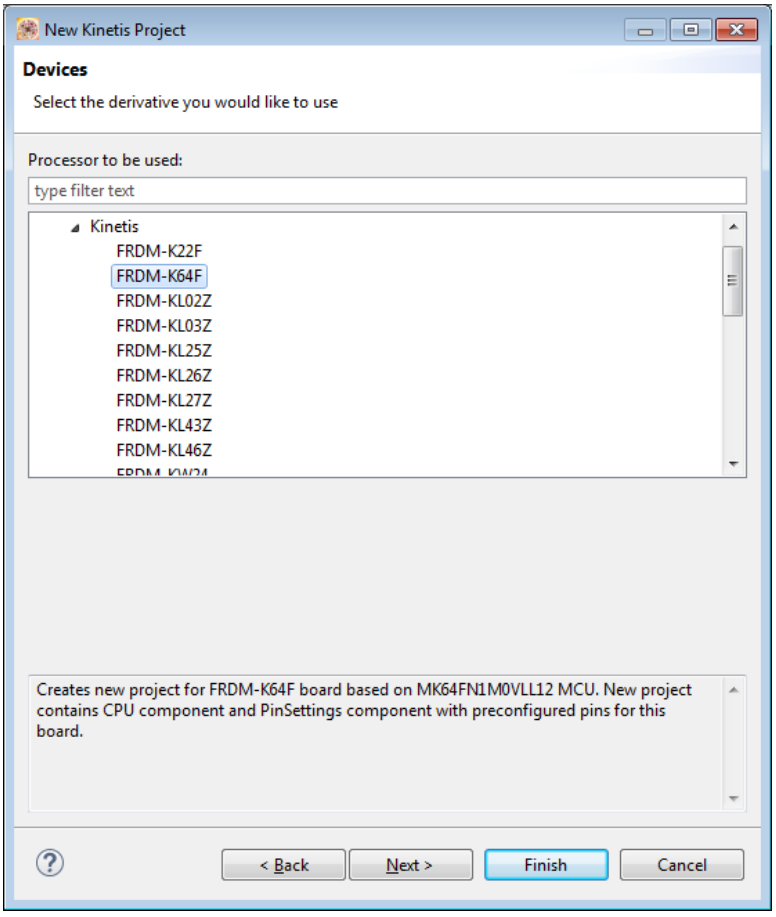
- This project is located in ***"${KSDK_PATH}/usb/usb_core/device/build/kds/usbd_sdk_frdmk64f_bm***

2 Create a New Kinetis Project

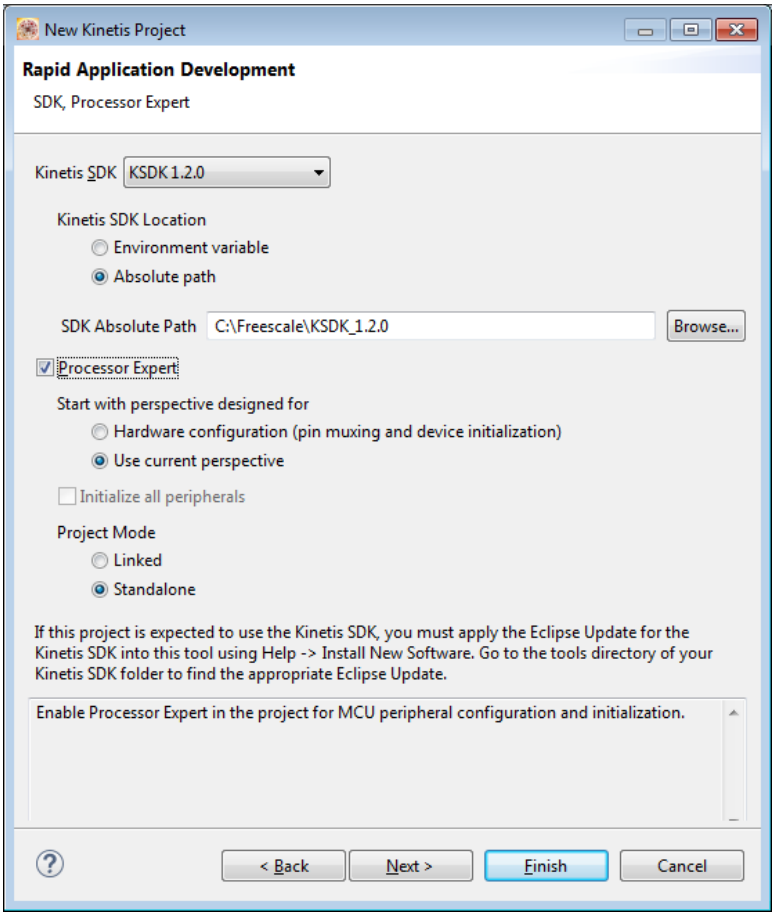
2.1 Go to ***menu File > New > Kinetis Project*** and wizard for new project will appear. Write a name for your project and click ***'Next'***



2.2 On ***Devices*** window, select your target and click ***Next >***. In this case, FRDMK64 is chosen.



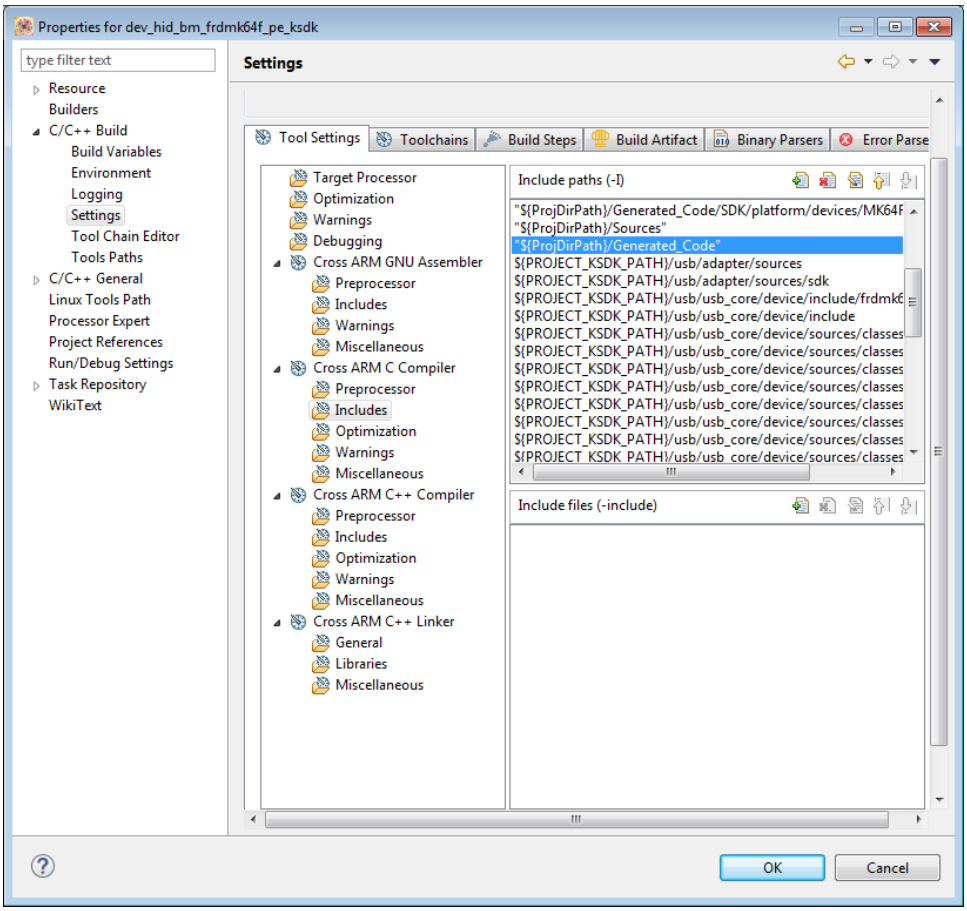
2.3 Next window allows user to select KSDK location along with Processor Expert support. Check Processor Expert option and click ***Finish*** button.



2.4 Once project is created, go to **menu Project > Properties > C/C++ Build > Settings > Cross ARM C Compiler > Includes** and add following USB includes. You can copy and paste the paths below.

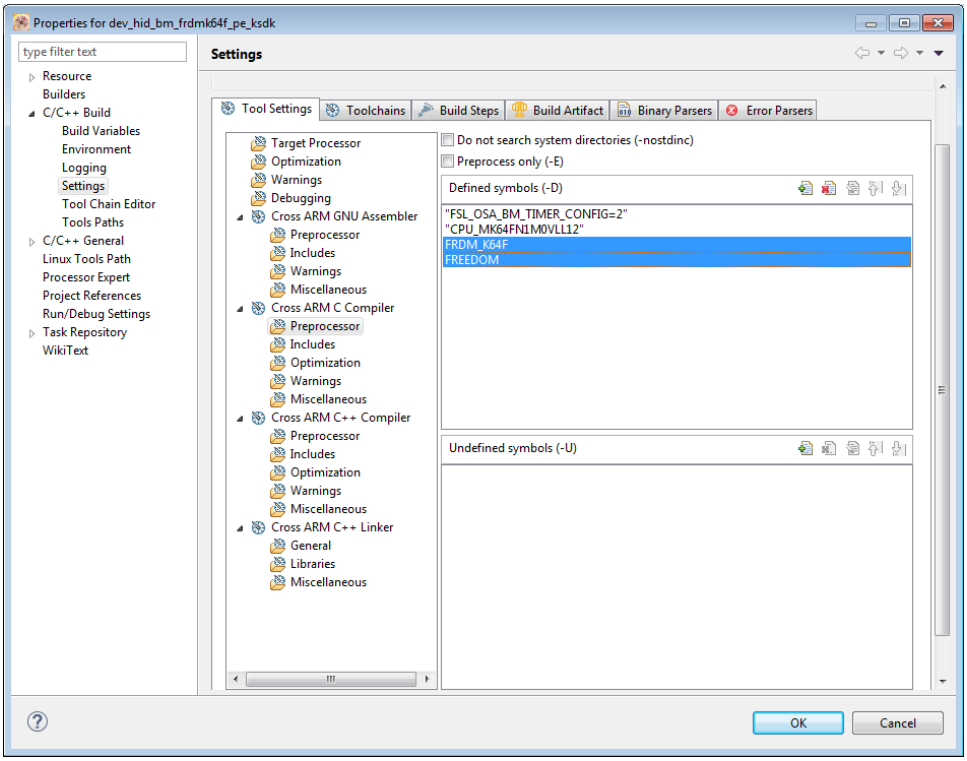
```
${PROJECT_KSDK_PATH}/usb/adapter/sources
${PROJECT_KSDK_PATH}/usb/adapter/sources/sdk
${PROJECT_KSDK_PATH}/usb/usb_core/device/include/frdmk64f
${PROJECT_KSDK_PATH}/usb/usb_core/device/include
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/audio
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/cdc
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/common
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/composite
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/hid
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/include
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/include/config
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/msd
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes/phdc
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/controller/khci
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/controller
${PROJECT_KSDK_PATH}/usb/usb_core/hal
${PROJECT_KSDK_PATH}/usb/usb_core/host/include
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/audio
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/cdc
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/hid
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/hub
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/msd
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/phdc
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/classes/printer
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/controller/khci
${PROJECT_KSDK_PATH}/usb/usb_core/host/sources/controller
${PROJECT_KSDK_PATH}/usb/usb_core/include
${PROJECT_KSDK_PATH}/usb/usb_core/otg/sources/driver/khci
${PROJECT_KSDK_PATH}/usb/usb_core/otg/sources/driver/max3353/sdk
${PROJECT_KSDK_PATH}/usb/usb_core/otg/sources/include
${PROJECT_KSDK_PATH}/usb/usb_core/otg/sources/otg
```

Note: These paths include all supported classes either for Host, Device or OTG controller. This way, you can create your own application based on a specific class or mode and all needed files are already included. (For this example, all paths related to Host, OTG, and device classes different to HID could be omitted)


Properties for dev_hid_bm_frdmk64f_pe_ksdk
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Tool Chain Editor
Tools Paths
C/C++ General
Linux Tools Path
Processor Expert
Project References
Run/Debug Settings
Task Repository
WikiText
Settings
Tool Settings
Toolchains
Build Steps
Build Artifact
Binary Parsers
Error Parse
Target Processor
Optimization
Warnings
Debugging
Cross ARM GNU Assembler
Preprocessor
Includes
Warnings
Miscellaneous
Cross ARM C Compiler
Preprocessor
Includes
Optimization
Warnings
Miscellaneous
Cross ARM C++ Compiler
Preprocessor
Includes
Optimization
Warnings
Miscellaneous
Cross ARM C++ Linker
General
Libraries
Miscellaneous
Include paths (-I)
"${ProjDirPath}/Generated_Code/SDK/platform/devices/MK64F
"${ProjDirPath}/Sources"
"${ProjDirPath}/Generated_Code"
${PROJECT_KSDK_PATH}/usb/adapter/sources
${PROJECT_KSDK_PATH}/usb/adapter/sources/sdk
${PROJECT_KSDK_PATH}/usb/usb_core/device/include/frdmk6
${PROJECT_KSDK_PATH}/usb/usb_core/device/include
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
${PROJECT_KSDK_PATH}/usb/usb_core/device/sources/classes
Include files (-include)
OK
Cancel

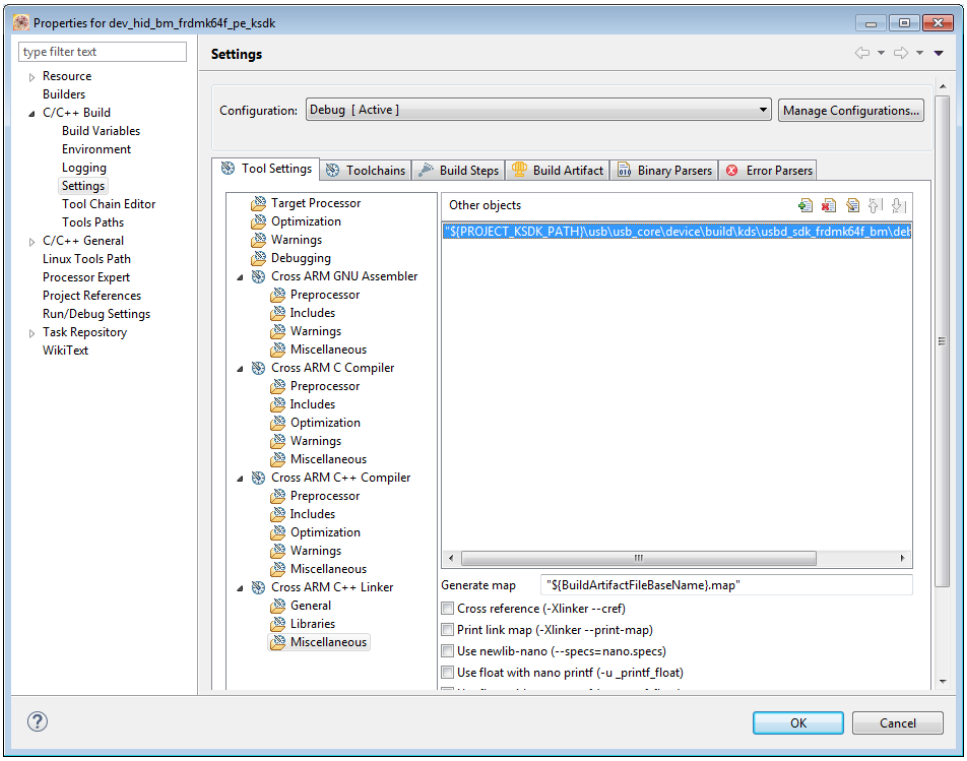
2.5 Go to menu **Project > Properties > C/C++ Build > Settings > Cross ARM C Compiler > Preprocessor** and set next defines:

```
FRDM_K64F
FREEDOM
```

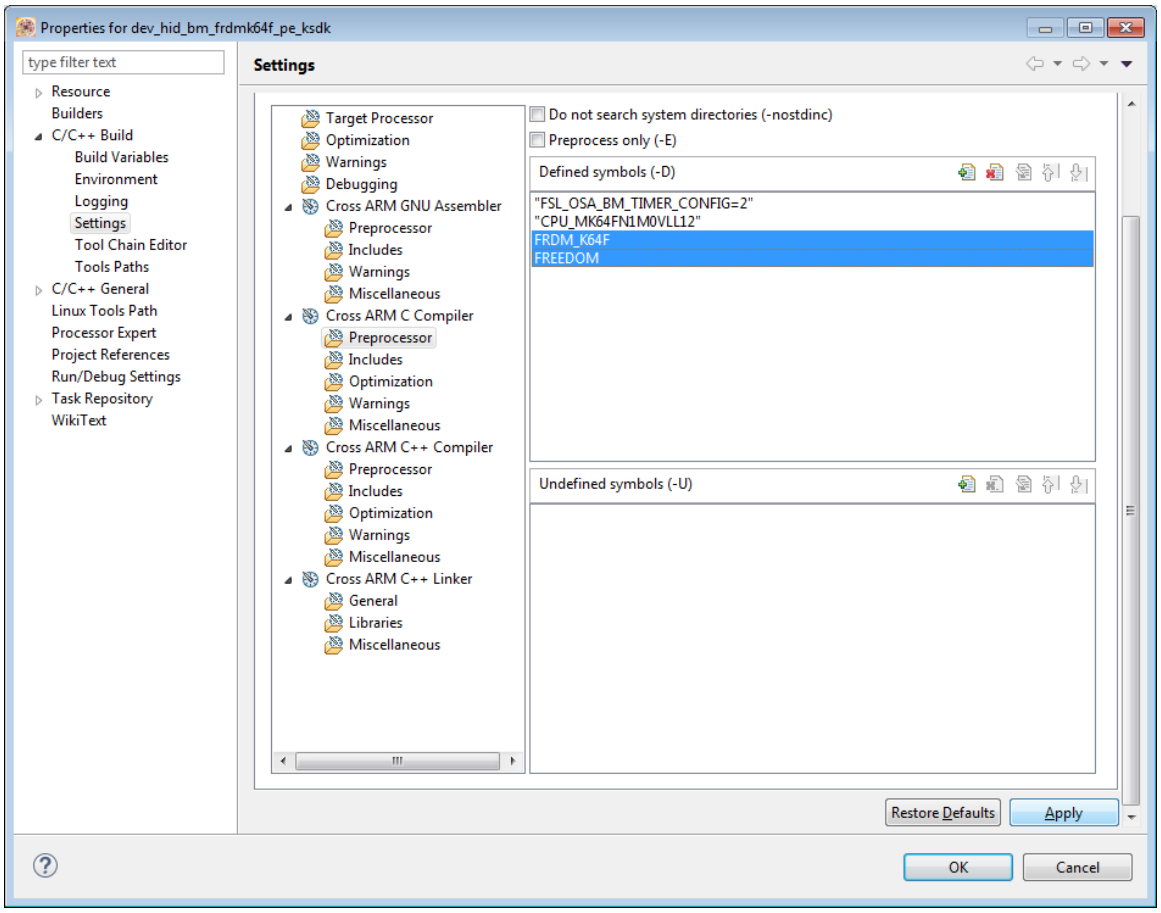


2.6 Go to menu **Project > Properties > C/C++ Build > Settings > Cross ARM C++ Linker > Miscellaneous**, add next USB library:

```
"${PROJECT_KSDK_PATH}\usb\usb_core\device\build\kds\usbd_sdk_frdmk64f_bm\debug\libusbd_bm.a"
```

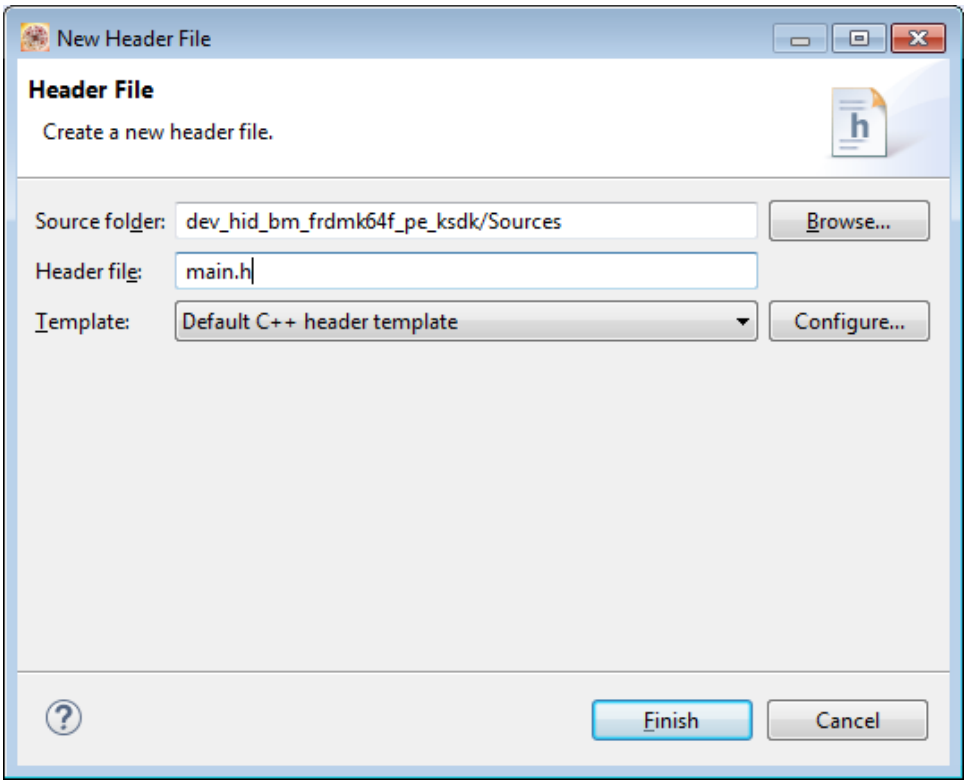


2.7 Now the new project is completely self-dependent and ready to run. Press **Apply** to save changes and then **OK**.



3 Adding USB Files and configuring PE components.

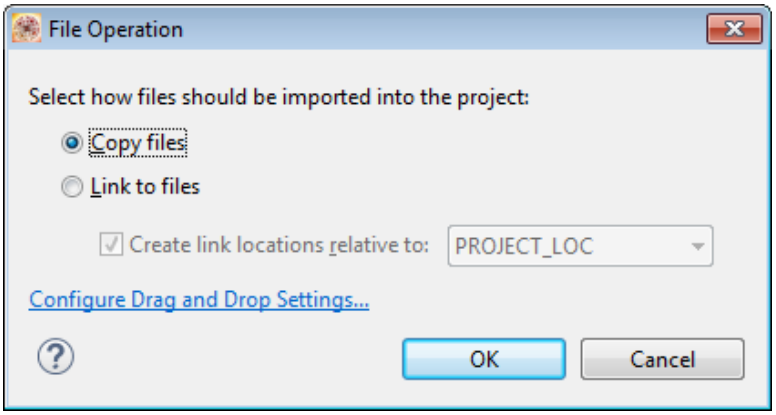
3.1 Once project settings are configured correctly, open **Sources** folder and add a new main header file. This is done by right-clicking on **Sources** folder and select ***New > Header File.***



3.2 Add next code to main.h file (It is based on dev_hid_mouse_frdmk64f_bm example):

```
/*****************************************************************************
 * Constants - None
 *****************************************************************************/

/*****************************************************************************
 * Macro's
 *****************************************************************************/
#define  MOUSE_BUFF_SIZE      (4)   /* report buffer size */
#define  REQ_DATA_SIZE        (1)

#define  HIGH_SPEED           (0)

#if HIGH_SPEED
#define CONTROLLER_ID         USB_CONTROLLER_EHCI_0
#else
#define CONTROLLER_ID         USB_CONTROLLER_KHCI_0
#endif
/*****************************************************************************
 * Types
 *****************************************************************************/
typedef struct _mouse_variable_struct
{
    hid_handle_t app_handle;
    uint16_t app_speed;
    uint8_t rpt_buf[MOUSE_BUFF_SIZE];/*report/data buff for mouse application*/
    uint8_t app_request_params[2]; /* for get/set idle and protocol requests*/
    uint8_t mouse_init;/* flag to check lower layer status*/
} mouse_global_variable_struct_t;

/*****************************************************************************
 * Global variables
 *****************************************************************************/

/*****************************************************************************
 * Global Functions
 *****************************************************************************/
extern void TestApp_Init(void);
```

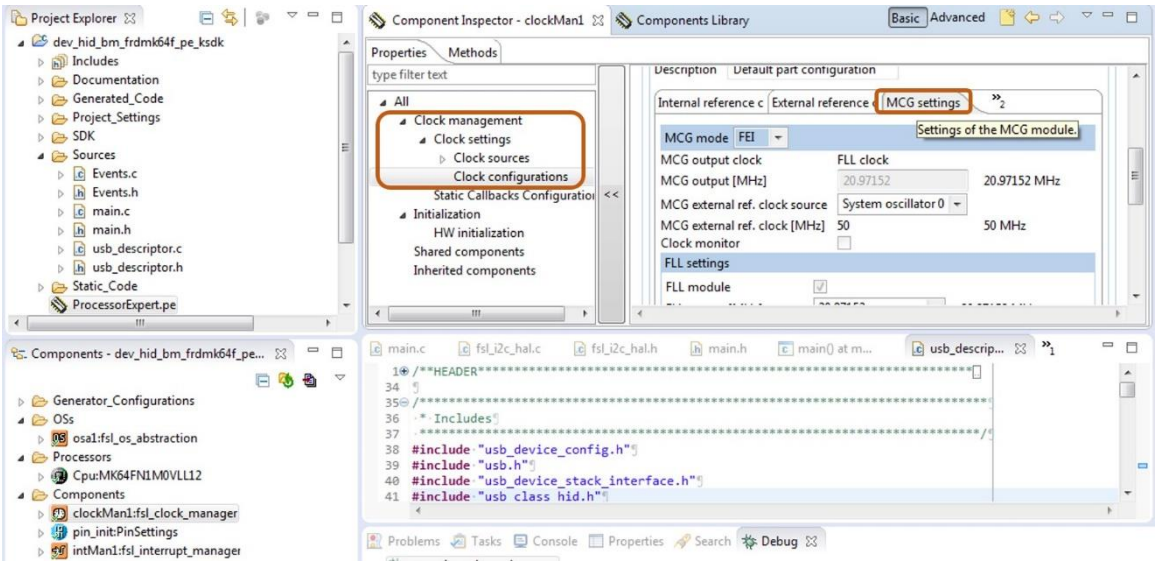
3.3 Copy usb_descriptor.c and usb_descriptor.h files from ***C:\Freescale\KSDK_1.2.0\examples\frdmk64f\demo_apps\usb\device\hid\hid_mouse*** to ***Sources*** Folder, You can drag and drop these files and select ***copy files***.



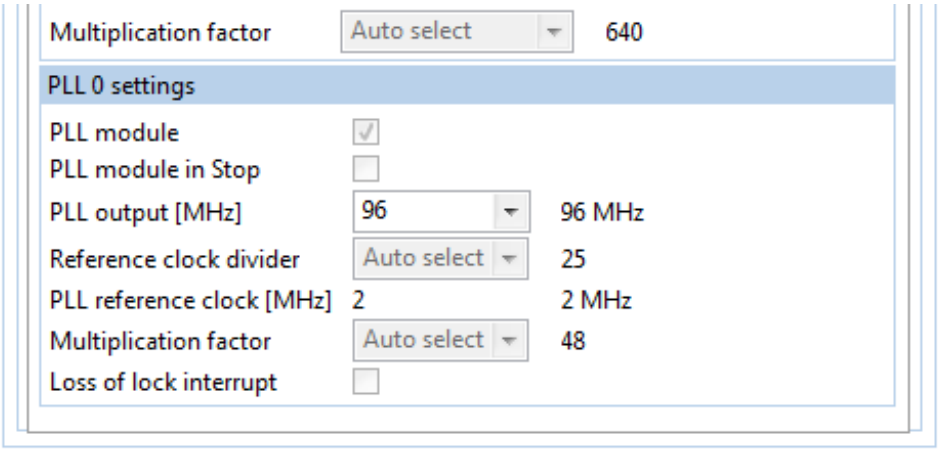
3.4 In usb_descriptor.c replace include file “mouse.h” by “main.h”

```
/**HEADER*******************************************************************
 *
 *****************************************************************************
 * Includes
 *****************************************************************************/
#include "usb_device_config.h"
#include "usb.h"
#include "usb_device_stack_interface.h"
#include "usb_class_hid.h"
#include "main.h"
#include "usb_descriptor.h"

/*****************************************************************************
 * Constant and Macro's
 *****************************************************************************/
usb_ep_struct_t g_ep[HID_DESC_ENDPOINT_COUNT] =
{
    {
        HID_ENDPOINT,
        USB_INTERRUPT_PIPE,
        USB_SEND,
        FS_INTERRUPT_OUT_ENDP_PACKET_SIZE,
    }
};

/* structure containing details of all the endpoints used by this device */
usb_endpoints_t g_usb_desc_ep =
{
```

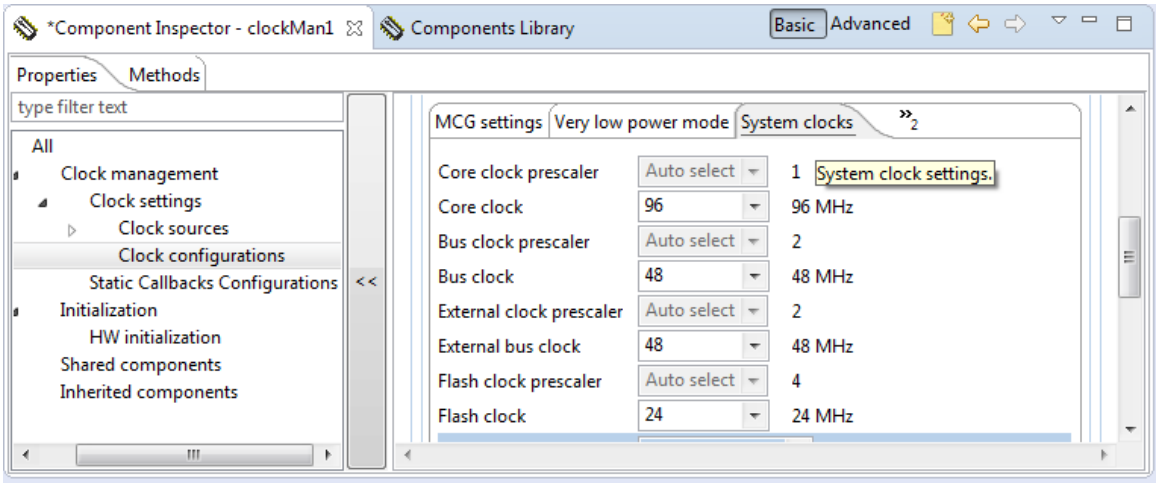
3.5 Now, Double click over ProecessorExpert.pe icon and configure clock settings for this board. Core Clock should be at least 20MHz for correct USB performance. Click over ***clockMan1:fsl_clock_manager*** component and go to ***Clock management > Clock settings > Clock configurations*** and select ***MCG settings*** tab.



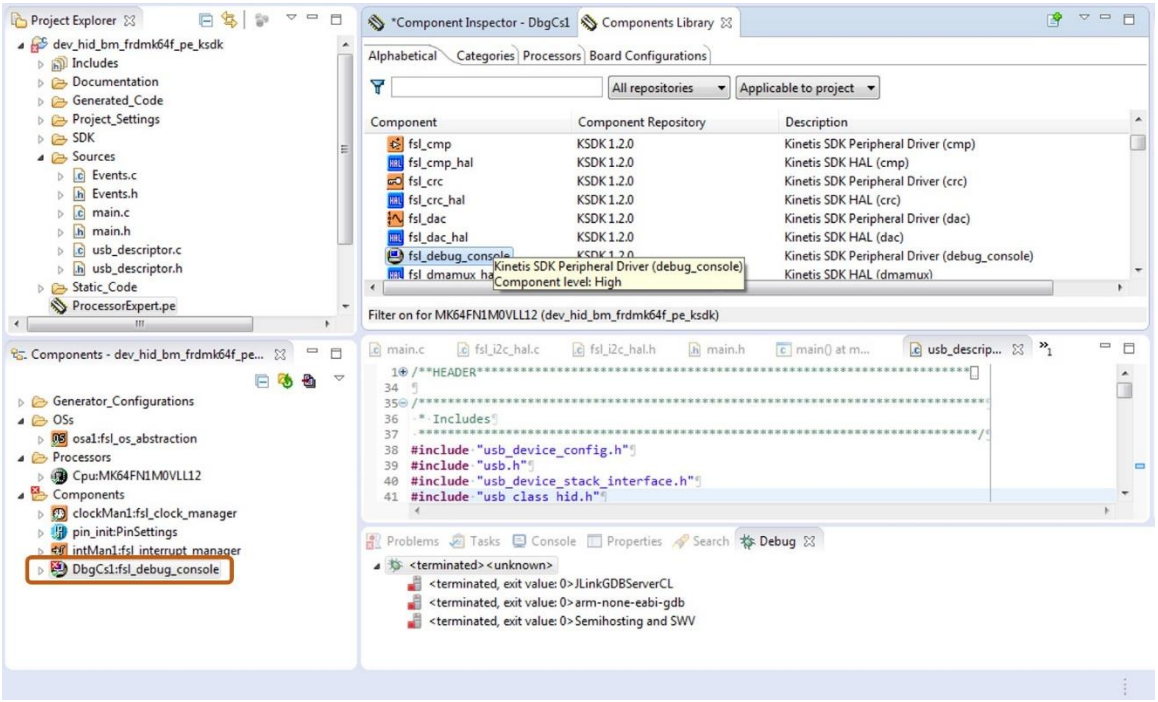
3.6 In ***MCG Settings tab***, select ***PEE*** as ***MCG mode*** and select PLL output frequency to 96 MHz.



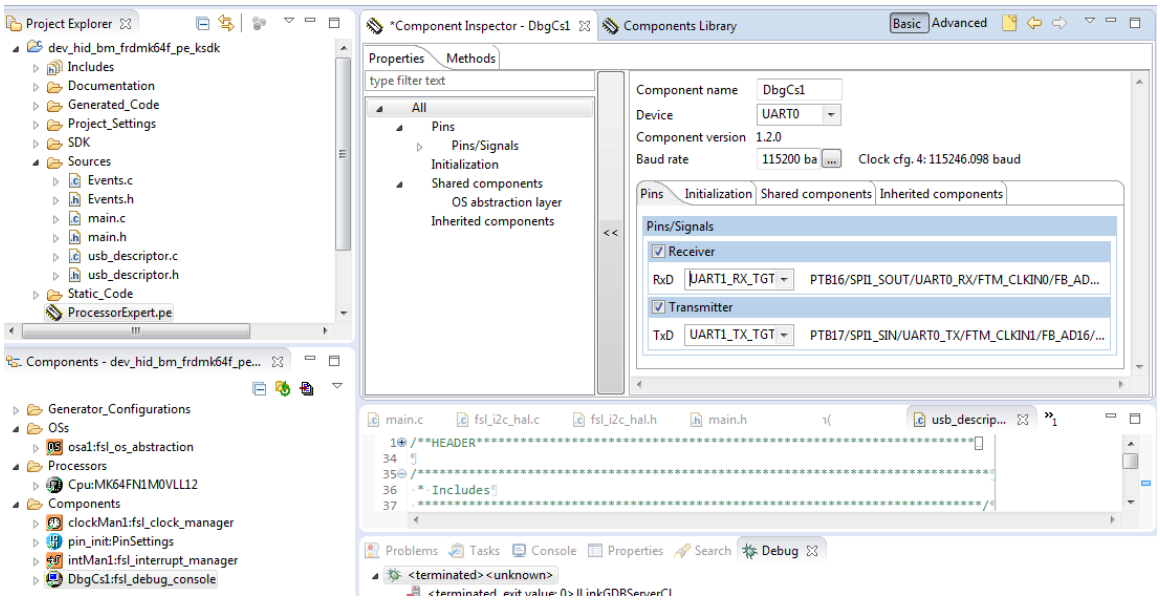
3.7 Go to ***System clocks*** tabs and configure Core, Bus, External and Flash clock according to your requirements. In this case, configurations is done as follows:



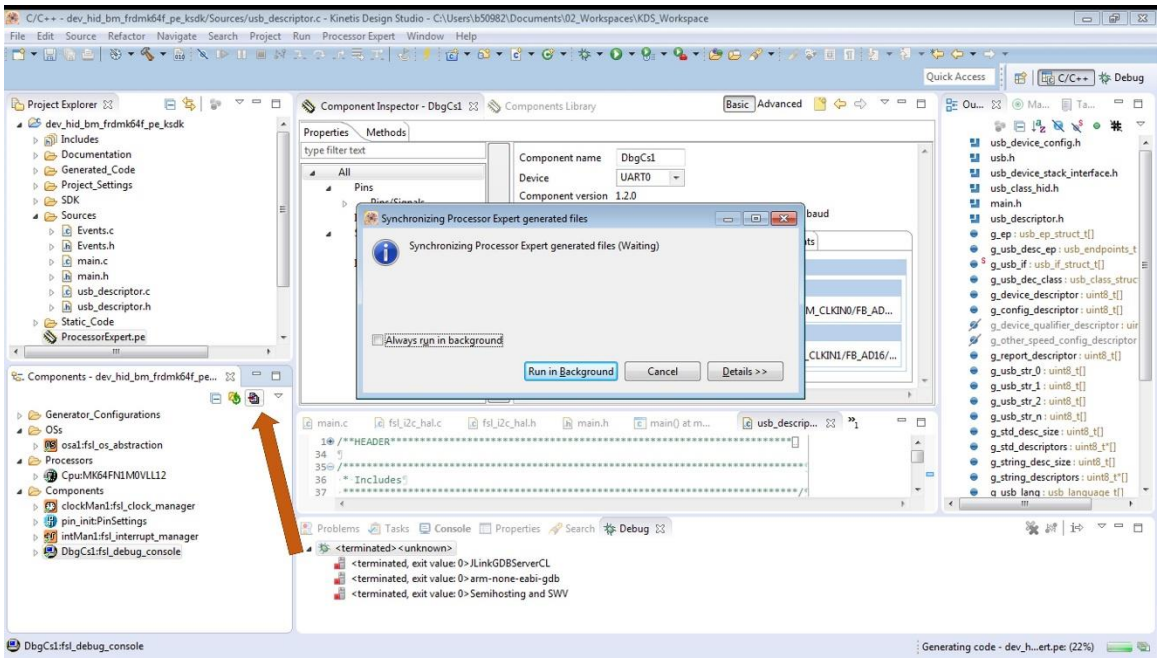
3.8 Now, it is necessary to add console component to current project. Go to ***Components Library*** and double-click on ***fsl_debug_console***. You will notice how DbgCs1 component has been added to your project.



3.9 DbgCs1 Components needs to be configured. Double click over ***DbgCs1:fsl_debug_console*** and configure UART modules as needed. For FRDMK64F, UART0 is connected to OpenSDA Interface and uses PTB16 and PTB17 pins for RxD and TxD respectively.



3.10 Save Processor Expert configuration and click ***Generate Processor Expert Code***.



3.11 Once Processor Expert Code has been generated, hide processor expert view (Go to ***menu Processor Expert > Hide Views***) and start editing application code on main.c.

4 Create Application

4.1 Now, we are ready to add our application code (either using KSDK or Processor Expert). In this case, dev_hid_mouse_frdmk64f_bm_frdmk64f was taken as reference and main.c file was modified as follows:

```
/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "clockMan1.h"
#include "pin_init.h"
#include "osa1.h"
#include "DbgCs1.h"
#if CPU_INIT_CONFIG
  #include "Init_Config.h"
#endif
/* User includes (#include below this line is not maintained by Processor Expert) */

#include "usb_device_config.h"
#include "usb.h"
#include "usb_device_stack_interface.h"
#include "usb_class_hid.h"
#include "main.h"
#include "usb_descriptor.h"

extern usb_desc_request_notify_struct_t g_desc_callback;
static mouse_global_variable_struct_t g_mouse;

/*****************************************************************************
 *      @name     move_mouse
 *      @brief    This function gets makes the cursor on screen move left,right
 *                up and down
 *      @param    None
 *      @return   None
 *****************************************************************************/
static void move_mouse(void)
{
    static int8_t x = 0, y = 0;
    enum { RIGHT, DOWN, LEFT, UP };
    static uint8_t dir = (uint8_t) RIGHT;

    switch(dir)
    {
    case RIGHT:
        g_mouse.rpt_buf[1] = 2;
        g_mouse.rpt_buf[2] = 0;
        x++;
        if (x > 100) {
            dir++;
        }
        break;
    case DOWN:
        g_mouse.rpt_buf[1] = 0;
```

```
        g_mouse.rpt_buf[2] = 2;
        y++;
        if (y > 100) {
            dir++;
        }
        break;
    case LEFT:
        g_mouse.rpt_buf[1] = (uint8_t)(-2);
        g_mouse.rpt_buf[2] = 0;
        x--;
        if (x < 0) {
            dir++;
        }
        break;
    case UP:
        g_mouse.rpt_buf[1] = 0;
        g_mouse.rpt_buf[2] = (uint8_t)(-2);
        y--;
        if (y < 0) {
            dir = RIGHT;
        }
        break;
    }
    (void) USB_Class_HID_Send_Data(g_mouse.app_handle, HID_ENDPOINT,
        g_mouse.rpt_buf, MOUSE_BUFF_SIZE);
}

void USB_App_Device_Callback(uint8_t event_type, void* val, void* arg)
{
    UNUSED_ARGUMENT (arg)
    UNUSED_ARGUMENT (val)

    switch(event_type)
    {
    case USB_DEV_EVENT_BUS_RESET:
        g_mouse.mouse_init = FALSE;
        if (USB_OK == USB_Class_HID_Get_Speed(g_mouse.app_handle, &g_mouse.app_speed))
        {
            USB_Desc_Set_Speed(g_mouse.app_handle, g_mouse.app_speed);
        }
        break;
    case USB_DEV_EVENT_ENUM_COMPLETE:
        g_mouse.mouse_init = TRUE;
        move_mouse();/* run the cursor movement code */
        break;
    case USB_DEV_EVENT_ERROR:
        /* user may add code here for error handling
         NOTE : val has the value of error from h/w*/
        break;
    default:
        break;
    }
    return;
}
```

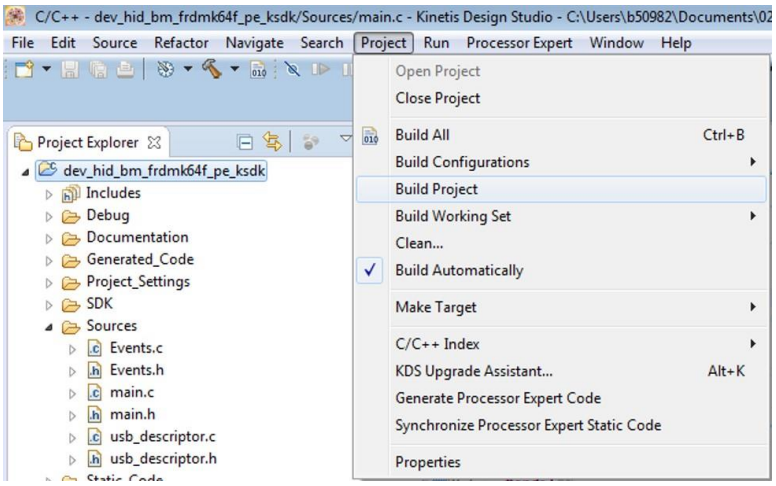
```
/*****************************************************************************
 *    @name        USB_App_Class_Callback
 *
 *    @brief       This function handles the callback for Get/Set report req
 *
 *    @param       request  :  request type
 *    @param       value    :  give report type and id
 *    @param       data     :  pointer to the data
 *    @param       size     :  size of the transfer
 *
 *    @return      status
 *                  USB_OK  :  if successful
 *                  else return error
 *****************************************************************************/
uint8_t USB_App_Class_Callback
(
    uint8_t request,
    uint16_t value,
    uint8_t ** data,
    uint32_t* size,
    void* arg
)
{
    uint8_t error = USB_OK;
    uint8_t index = (uint8_t)((request - 2) & USB_HID_REQUEST_TYPE_MASK);

    if ((request == USB_DEV_EVENT_SEND_COMPLETE) && (value == USB_REQ_VAL_INVALID)) {
        if ((g_mouse.mouse_init) && (arg != NULL)) {
            move_mouse();/* run the cursor movement code */
        }
        return error;
    }
    /* index == 0 for get/set idle, index == 1 for get/set protocol */
    *size = 0;
    /* handle the class request */
    switch(request)
    {
    case USB_HID_GET_REPORT_REQUEST:
        *data = &g_mouse.rpt_buf[0]; /* point to the report to send */
        *size = MOUSE_BUFF_SIZE; /* report size */
        break;

    case USB_HID_SET_REPORT_REQUEST:
        for (index = 0; index < MOUSE_BUFF_SIZE; index++)
        { /* copy the report sent by the host */
            g_mouse.rpt_buf[index] = *(*data + index);
        }
        break;

    case USB_HID_GET_IDLE_REQUEST:
        /* point to the current idle rate */
        *data = &g_mouse.app_request_params[index];
        *size = REQ_DATA_SIZE;
        break;
```

```
    case USB_HID_SET_IDLE_REQUEST:
        /* set the idle rate sent by the host */
        g_mouse.app_request_params[index] = (uint8_t)((value & MSB_MASK) >>
        HIGH_BYTE_SHIFT);
        break;

    case USB_HID_GET_PROTOCOL_REQUEST:
        /* point to the current protocol code
         0 = Boot Protocol
         1 = Report Protocol*/
        *data = &g_mouse.app_request_params[index];
        *size = REQ_DATA_SIZE;
        break;

    case USB_HID_SET_PROTOCOL_REQUEST:
        /* set the protocol sent by the host
         0 = Boot Protocol
         1 = Report Protocol*/
        g_mouse.app_request_params[index] = (uint8_t)(value);
        break;
    }
    return error;
}

/*****************************************************************************
 *
 *   @name        TestApp_Init
 *
 *   @brief       This function is the entry for mouse (or other usage)
 *
 *   @param       None
 *
 *   @return      None
 **
 *****************************************************************************/
void TestApp_Init(void)
{
    hid_config_struct_t config_struct;

    /* initialize the Global Variable Structure */
    OS_Mem_zero(&g_mouse, sizeof(mouse_global_variable_struct_t));
    OS_Mem_zero(&config_struct, sizeof(hid_config_struct_t));

    /* Initialize the USB interface */
    USB_PRINTF("begin to test mouse\r\n");
    config_struct.hid_application_callback.callback = USB_App_Device_Callback;
    config_struct.hid_application_callback.arg = &g_mouse.app_handle;
    config_struct.class_specific_callback.callback = USB_App_Class_Callback;
    config_struct.class_specific_callback.arg = &g_mouse.app_handle;
    config_struct.desc_callback_ptr = &g_desc_callback;

    g_mouse.app_speed = USB_SPEED_FULL;
    USB_Class_HID_Init(CONTROLLER_ID, &config_struct, &g_mouse.app_handle);
}
```

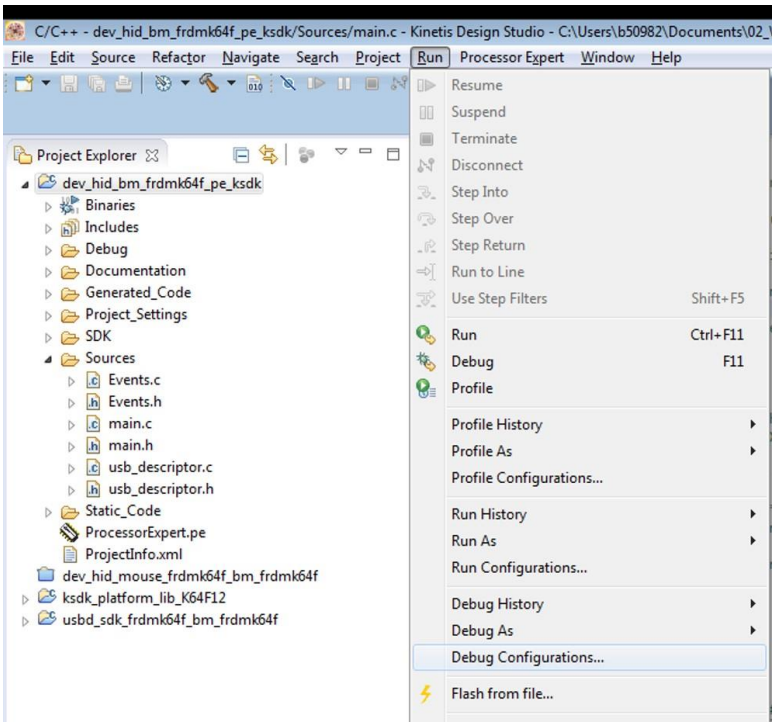
```
void APP_task()
{
    USB_HID_Periodic_Task();
}

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  TestApp_Init();
  /*** Don't write any code pass this line, or it will be deleted during code
generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T
MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is
defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
```

5 Run the application

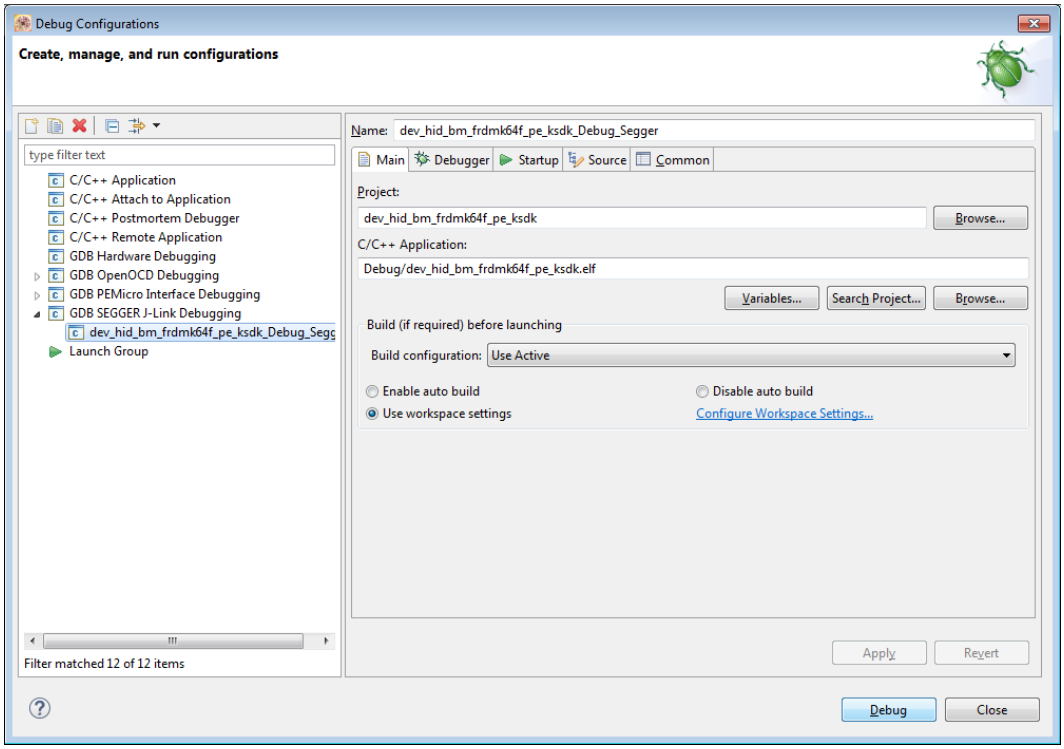
5.1 Build your application, go to ***menu Project > Build Project***. Alternately click the hammer button.



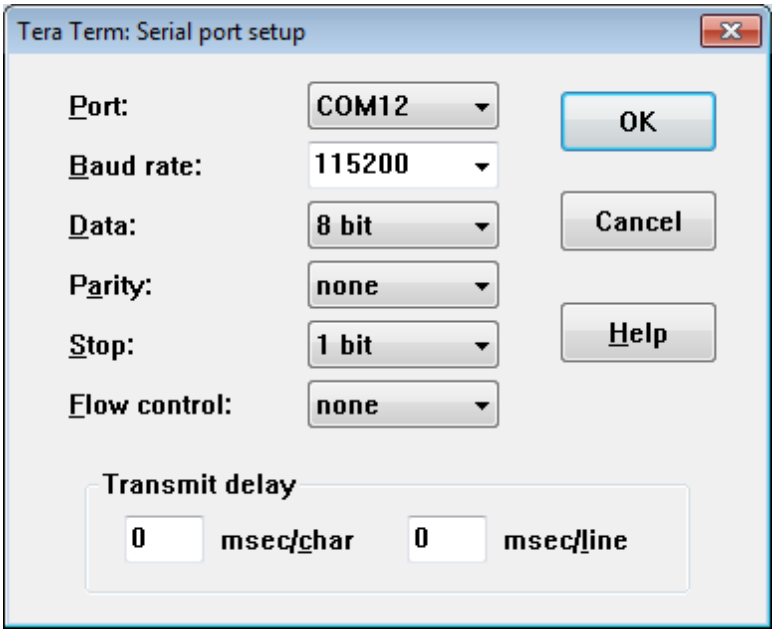
5.2 Go to ***menu Run > Debug Configurations***…



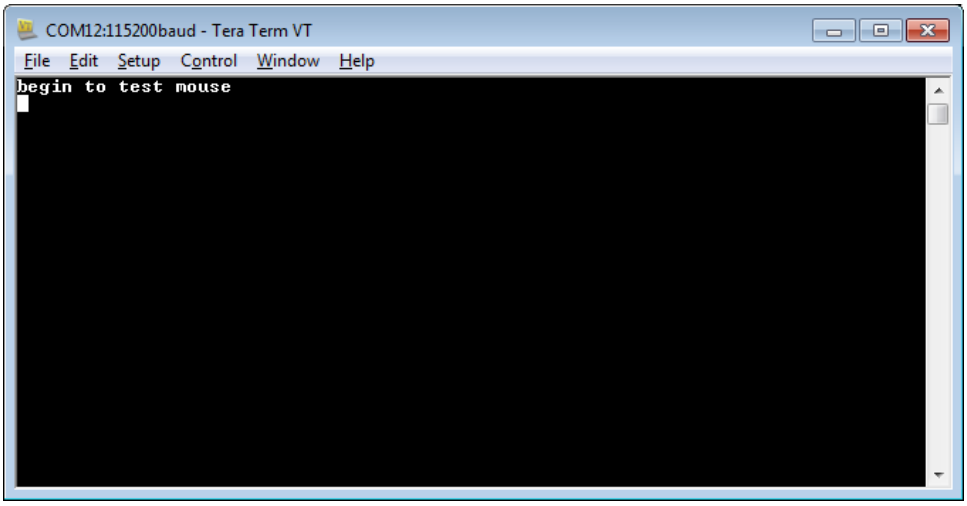
5.3 Select the ***'Debug Configuration'*** that matches your connection type, in this example **GDB SEGGER J-Link** connection is used.



5.4 Open a terminal, select the appropriate port and set baud rate to 115200.



5.5 Run the application, you will see ***"begin to test mouse"*** in terminal.



5.6 Connect an USB Micro-B cable on J22 from FRDMK64F board to a PC-Host.

5.7 After successful driver installation process on PC host, you can see how cursor is moving right, down, left and up according ***move_mouse()*** function.

Now you have an USB project with KSDK and Processor Expert compatibility to develop your application.

Other useful links

KSDK GPIO driver with Processor Expert
https://community.freescale.com/videos/3195