Lab 2 – Getting Started with MCUXpresso

- Creating new IDE project
- Working with Config Tools
- Working with SDK APIs

Creating a new project

This section covers creating a blank MCUXpresso SDK project using the IDE New Project Wizard

1. Use the Quickstart Panel to create a "New project…" 

   1.1. Select the FRDM-K64F board.

      1.1.1. If needed to help filter the selection of SDK, select "K6x" from the list of "SDK MCUs"

      1.1.2. Locate and select the picture of the board (the text below the picture is a hyperlink to learn more about the board on http://nxp.com

      1.1.3. Select the board selected (highlighted) select "Next"

   1.2. Review default project configuration settings

   1.3. Select **Finish**

   1.4. Review resulting project

Default Config Tool setting and initial Function Groups

This section covers the default board configuration templates that are predefined in MCUXpresso Config Tools. Details will cover default settings and function groups.

2. Launch/Enable Config Tools

   2.1. Use the Project Explorer menu bar  to open the Tool Overview

   2.2. Review overview information

   2.3. Config Initial Function Groups (that will be called at boot) by selecting the Flags  next to the names

      2.3.1. Enable default Pin Functional Groups (disabling others if needed):

         - **BOARD_InitPins**
         - **BOARD_InitButtons**
         - **BOARD_InitLEDs**

      2.3.2. Enable the Peripheral **BOARD_InitBUTTONsPeripheral** Functional group

2.4. Launch the Pin Tool by clicking on the **BOARD_InitLEDs** functional group name

2.4.1. Review configuration of GPIO output pins in the functional group

2.5. Switch to the Clock Tool using the IDE Perspective links in the upper right corner 

2.5.1. Review default settings, currently clock configuration is setup to chip defaults 20.971 MHz clock derived from an internal 32.768 kHz internal reference clock.

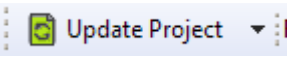
2.6. Switch to the Peripheral Tool using the IDE Perspective links in the upper right corner 

2.6.1. Select the **BOARD_InitBUTTONsPeripheral** functional group from the drop-down menu in the toolbar

2.6.2. Double-click the SW2 and SW3 rows in the Peripheral table to open the respective configuration windows

2.6.3. Take note of Interrupt handler names associates with the two peripheral components

- **BOARD_SW2_IRQHANDLER** (*for Port C*)
- **BOARD_SW3_IRQHANDLER** (*for Port A*)

2.7. Update Project with the Config Tool settings (default functional groups) 

Interrupt Callbacks and GPIO API

This section will cover interrupt callback functions for GPIOs and GPIO SDK Driver API for toggling the development board LEDs.

3. Add interrupt callback functions for SW2 and SW3

3.1. Project files to reference:

- Complete these steps twice for SW2 and SW3 selecting a **different** LED color
- Refer to **fsl_gpio.h** for functions
- Refer to **pin_mux.h** for input parameter defines

3.2. Example callback prototype:

- `void BOARD_SW2_IRQHANDLER(void) { }`

3.3. Contents of interrupt callback functions:

3.3.1. Get triggered interrupt flags using:

- `uint32_t flags = GPIO_PortGetInterruptFlags(BOARD_SW2_GPIO);`

3.3.2. Clear triggered interrupt flags using:

- `GPIO_PortClearInterruptFlags(BOARD_SW2_GPIO, flags);`

3.3.3. Toggle GPIO using:

- `GPIO_PortToggle(BOARD_LED_BLUE_GPIO, 1 << BOARD_LED_BLUE_PIN);`

3.4. Build, Debug, and Run the application

3.4.1. Pressing SW2 or SW3 should toggle a different LED color
Note: when both LEDs are on the TriColor LED will appear as a blend of the colors

3.5. End the debug session using the Terminate (red stop) button.

Add Programmable Interrupt Timer (PIT) using Peripheral Tool

This section covers configuring the Programable Interrupt Timer (PIT) using the Peripheral Tool, specifying the frequency and calculated clock frequency, and finally adding the corresponding interrupt callback functions.

4. Launch back into the Config Tools, specifically to the peripheral tool
*Note: You can launch the peripheral tool by Right-Clicking the **board/peripheral.c** file from the desired project.*

4.1. Ensure that the **BOARD_InitBUTTONsPeripheral** functional group is selected

4.2. Add a **PIT** Peripheral configuration by placing a checkmark next to PIT and confirming the default selection of driver

4.2.1. Review Clock Settings: Clock Source and Frequency determined by **CLOCK_GetFreq()**

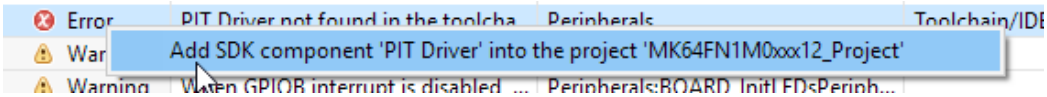
4.2.2. Change Timer period/frequency to **500ms**

4.2.3. Take note of Interrupt settings:

- Interrupt enabled
- Default Interrupt handler name: **BOARD_PIT_1_0_IRQHANDLER**

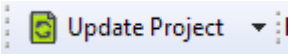
4.3. Error in Problems View
PIT Driver not found in the toolchain/IDE project

4.3.1. Right-click Error row in Problems panel (lower right corner) and select resolution



4.3.2. Confirm **YES** to add **fsl_pit** driver to IDE project *(this can also be done within the IDE settings)*

4.4. With **errors resolved**, update the project with the Config Tool settings (with added PIT configuration)



5. Add PIT API function call inside of **main()** to start Timer

5.1. Project files to reference:

- Refer to **fsl_pit.h** for functions

5.2. Start PIT Timer using (place function call just before while loop):

- `PIT_StartTimer(BOARD_PIT_1_PERIPHERAL, kPIT_Chnl_0);`

6. Add interrupt callback functions for PIT

   6.1. Project files to reference:

   - Refer to **fsl_pit.h** for functions
   - Refer to **peripherals.h** for input parameter defines

   6.2. Example callback prototype:

   - `void BOARD_PIT_1_0_IRQHANDLER(void) { }`

   6.3. Contents of interrupt callback functions:

   6.3.1. Get interrupt status flags using:

   - `uint32_t flags = PIT_GetStatusFlags(BOARD_PIT_1_PERIPHERAL, kPIT_Chnl_0);`

   6.3.2. Clear status flags using:

   - `PIT_ClearStatusFlags(BOARD_PIT_1_PERIPHERAL, kPIT_Chnl_0, flags);`

   6.3.3. Toggle GPIO using *(using the third color not already used with the pushbuttons)*:

   - `GPIO_PortToggle(BOARD_LED_GREEN_GPIO, 1 << BOARD_LED_GREEN_PIN);`

   6.4. Build, Debug, and Run the application

   6.4.1. One of the LEDs should now toggle on and off every second, while SW2 and SW3 will continue to function.
   Note: when multiple LEDs are on the TriColor LED will appear as a blend of the colors

   6.5. End the debug session using the Terminate (red stop) button.

Add a print statement in response to the PIT timer

This section will cover adding a printf statement to the main "while(1)" loop that is triggered from the PIT interrupt callback.

7. Add a **PRINTF** statement that will periodically print the value of the free running counter variable "i"

   7.1. Create a global Boolean variable (added above interrupt function declarations

   - `bool tick = false;`

   7.2. Add statement to PIT interrupt callback function to set tick to true

   - `tick = true;`

   7.3. Add conditional block of code inside the "while(1)" loop that will print the value of **i/10000** whenever tick is *true* and then reset tick to *false*

   - ```
     if (tick) {
         PRINTF( "%d\n", i/10000 );
         tick = false;
     }
     ```
     ```

7.4. Build, Debug, and Run the application

7.4.1. A numerical value will print to the IDE console twice a second, note the typical difference between values

7.5. End the debug session using the Terminate (red stop) button.

Modifying Clock Settings

This section will explore high level settings within the Clock Tool and preconfigured functional groups.

8. Launch back into the Config Tools, specifically to the clock tool
   *Note: You can launch the peripheral tool by Right-Clicking the **board/clock_config.c** file from the desired project.*

   8.1. Review the current clock settings (reset conditions with system clock at 20.972 MHz)

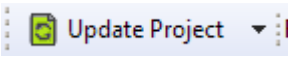
   8.2. Open the Clock Functional Group Property window using the IDE menu **Clocks -> Functional Groups** or by using the icon on the tool icon 

   8.2.1. Use the  symbol to add a new functional group and specify a unique name like "**Board_Default**"
   Note: this name will be used in code, so it cannot contain whitespace or special characters

   8.2.2. Place a checkmark in the "Called from default initialization function" to switch the project to use this configuration by default.

   8.2.3. Change the configuration for this group to use the "Board Defaults" using the IDE menu **Clocks -> Reset to Board Defaults**, and confirm the reset.
   Note: Board Default is different that the out of reset conditions. The board default is aware of the clock input sources available on the development board and will run the device at the highest recommended frequency for run mode.

   - Core clock should reset to 120 MHz derived from a 50 MHz external reference clock

   8.3. Update Project with the Config Tool settings (updated clock settings)
   

   8.4. Build, Debug, and Run the application

   8.4.1. A numerical value will print to the IDE console twice a second, note the typical difference between values
   Note: difference between values should be much higher. Our PIT timer is still firing twice a second, but the core frequency is much higher so our free incrementing counter is now running much faster.

   8.5. End the debug session using the Terminate (red stop) button.

9. *Optional Step*: Clock Settings for **Very Low Power Run Mode (VLPR)**

   9.1. Open the clock tool, change the initial clock functional group to **BOARD_BootClockVPLR**

   9.2. Build, Debug, and Run the application

      9.2.1. A numerical value will print to the IDE console twice a second, note the typical difference between values
      Note: difference between values should be much lower. Our PIT timer is still firing twice a second, but the core frequency is much lower so our free incrementing counter is now running much slower.

   9.3. End the debug session using the Terminate (red stop) button.

10. *Optional Step*: Clock Settings for custom clock settings

   10.1. Open the clock tool, change the initial clock functional group back to the "**Board_Default**" group

   10.2. Try modifying the resulting Core Clock to various values

      - Pay attention to locked settings
      - Some clock frequencies are limited due to USB clock, disable USB clock output for more flexibility

   10.3. Build, Debug, and Run the application

      10.3.1. A numerical value will print to the IDE console twice a second, note the typical difference between values
      Note: difference between values should be much lower. Our PIT timer is still firing twice a second, but the core frequency is much lower so our free incrementing counter is now running much slower.

   10.4. End the debug session using the Terminate (red stop) button.