



Summer School Guide

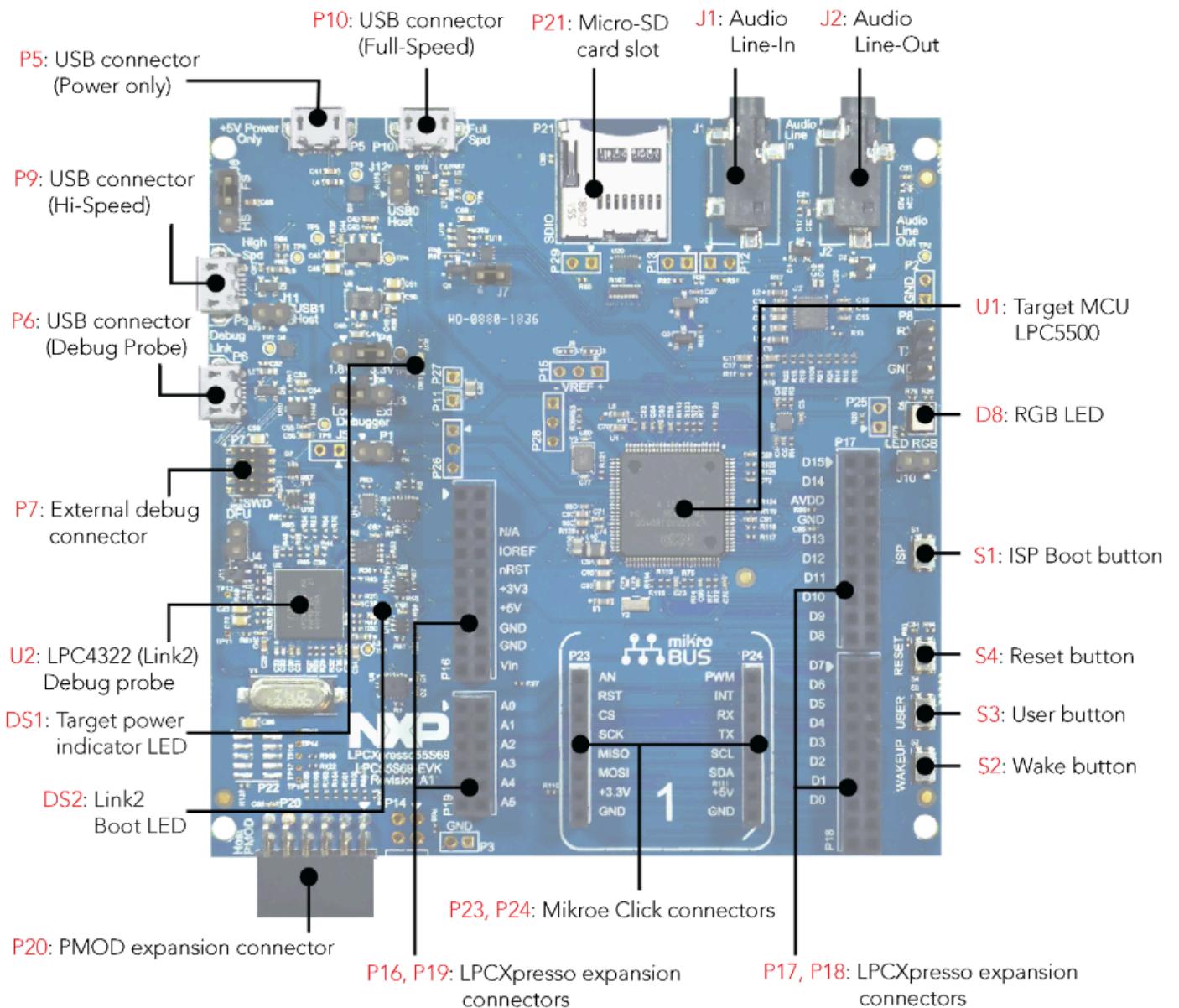
June 22, 2024

Tools

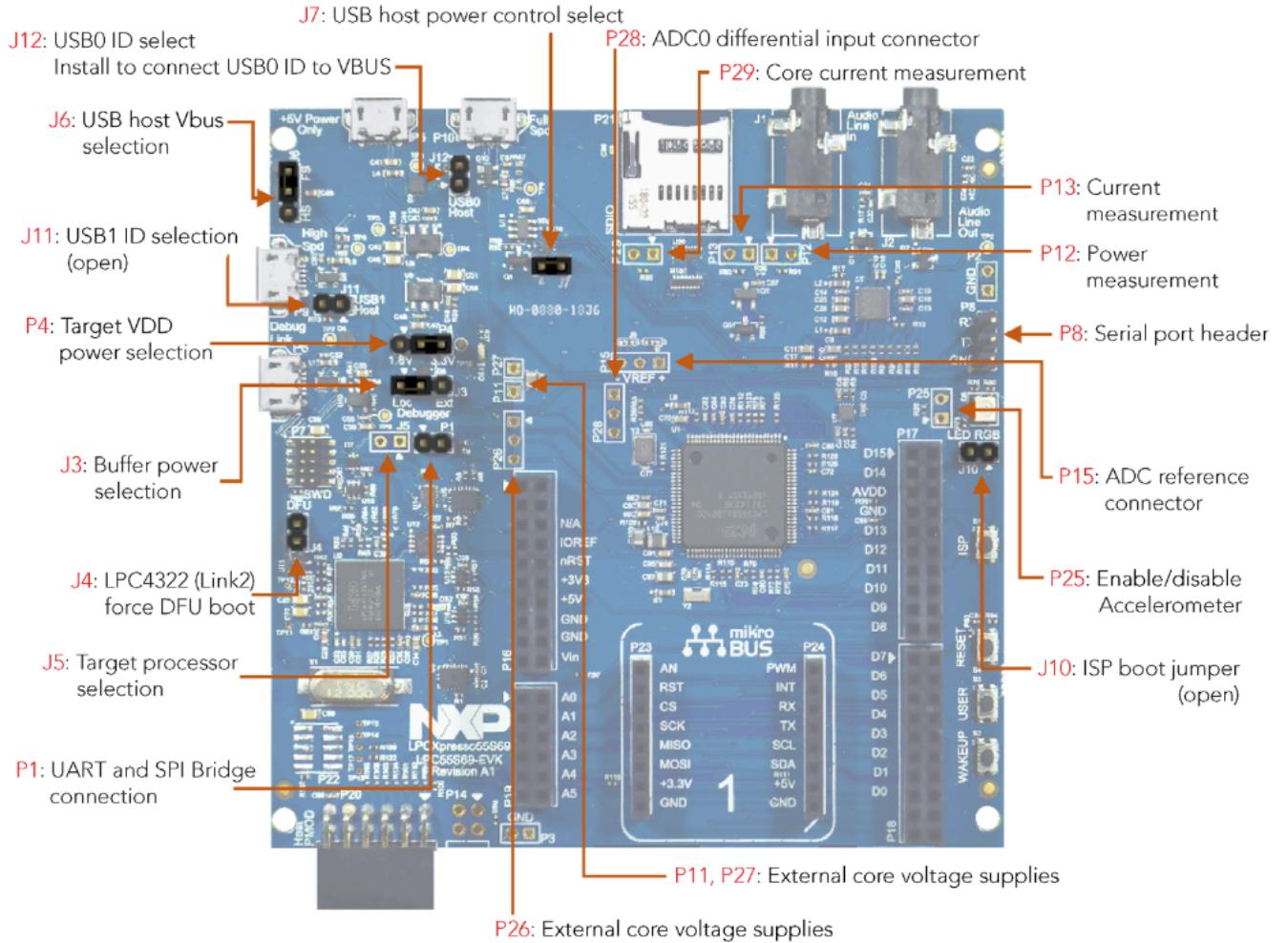
LPCXpresso55S69 Development Board

The LPCXpresso55S69 and LPCXpresso55S28 boards include the following features:

- LPC55S69 Dual Cortex-M33 core processor or LPC55S28 Cortex-M33 processor
- Onboard, high-speed USB, Link2 debug probe with CMSIS-DAP and SEGGER J-Link protocol options
- UART and SPI port bridging from LPC55Sxx target to USB via the onboard debug probe
- Optional external debug probes with trace option (10 or 20 pin Cortex-M connectors)
- RGB user LED
- Reset, ISP, User/Wakeup and user buttons
- Multiple Expansion options, including Arduino UNO, Mikroe Click and PMod
- Micro SD card slot
- NXP MMA8652FCR1 accelerometer
- Stereo audio codec with line in/out
- High / full speed USB port with micro A/B connector for the host or device functionality
- Reset button



Board Elements Overview 1



Board Elements Overview 2

Why do we need SDKs?

- An SDK (Software Development Kit) is a collection of software tools, libraries, documentation, and sample code that developers use to create applications for specific hardware or software platforms. SDKs streamline the development process by providing the necessary components and resources in one package.
 - Hardware Abstraction - it simplifies access to hardware by offering drivers and libraries.
 - Sample Code and Documentation - provides a lot of examples that showcase the functionalities on the specific board/development kit.
 - Consistency and Efficiency - provides standardized interfaces and already follows coding guidelines.

- Support and Updates - ongoing support provided by the manufacturer/community.

An SDK can be downloaded by searching for the development board name and accessing the development board main page (provided below) or by accessing the NXP portal, MCUXpresso SDK Builder and searching for the board we need the toolkit for.

Useful Links:

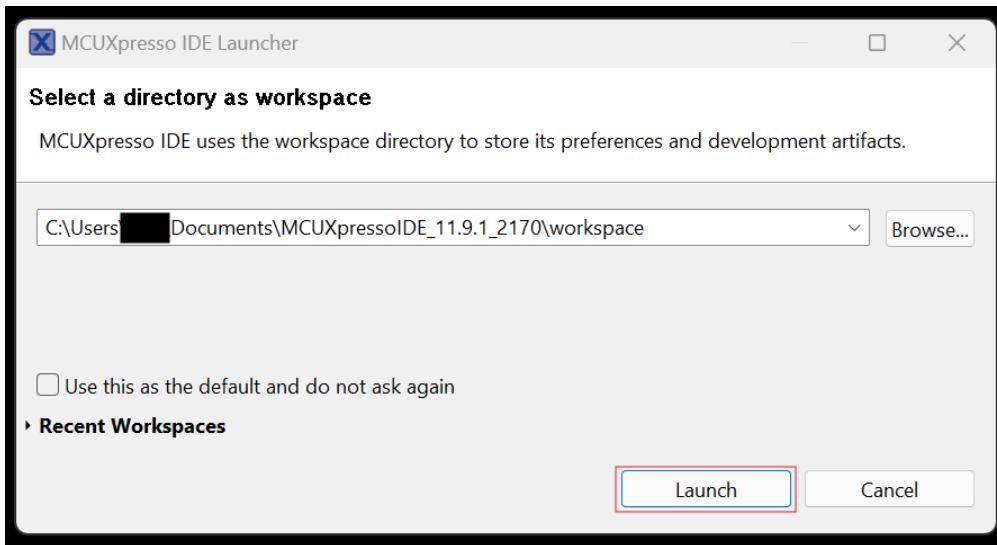
- [Development Board Purchase details, documentation and SDK](#)
- [MCUXpresso SDK Builder](#)

MCUXpresso IDE

The MCUXpresso IDE offers advanced editing, compiling, and debugging features with the addition of MCU-specific debugging views, code trace and profiling, multicore debugging, and integrated configuration tools.

Getting started with the MCUXpresso IDE

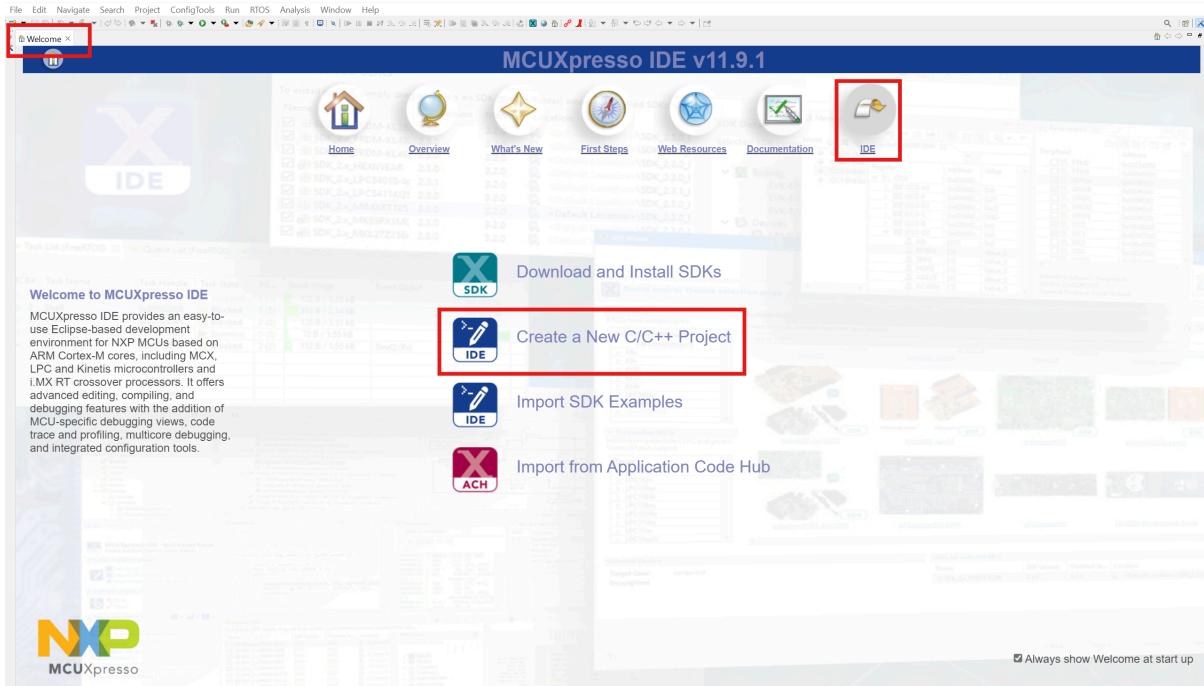
1. Select Working Directory



Select Workspace

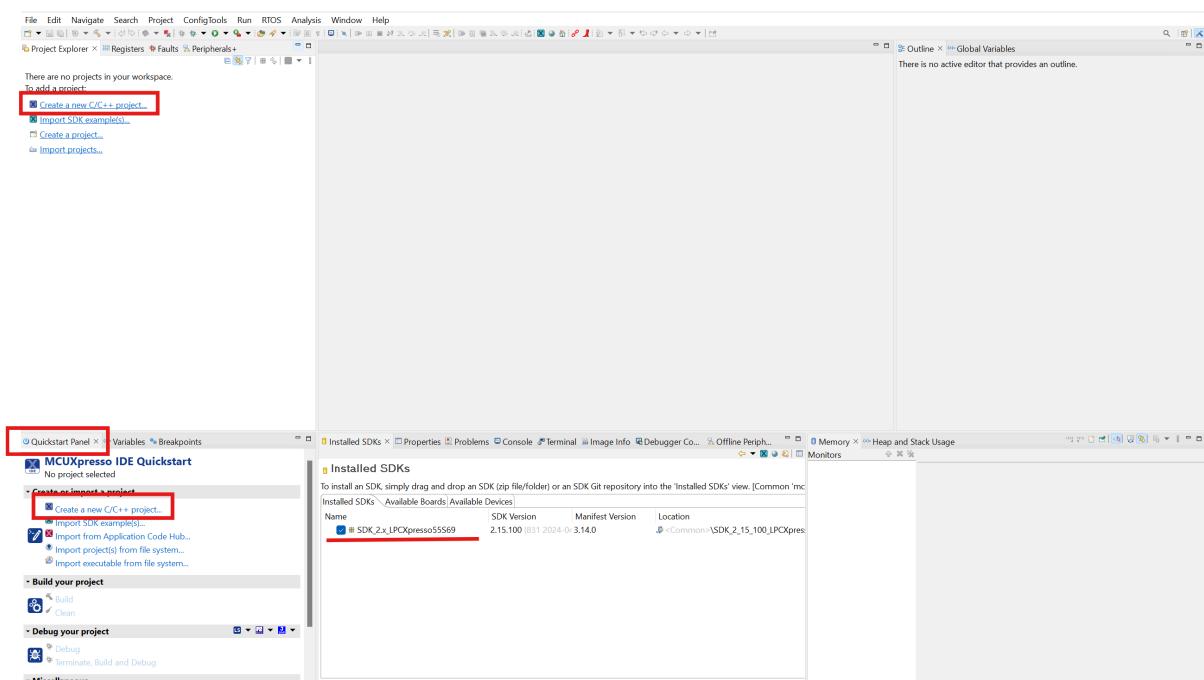
2. Welcome Page

Either use this page to create/import projects, or dismiss it and go to the main IDE view.



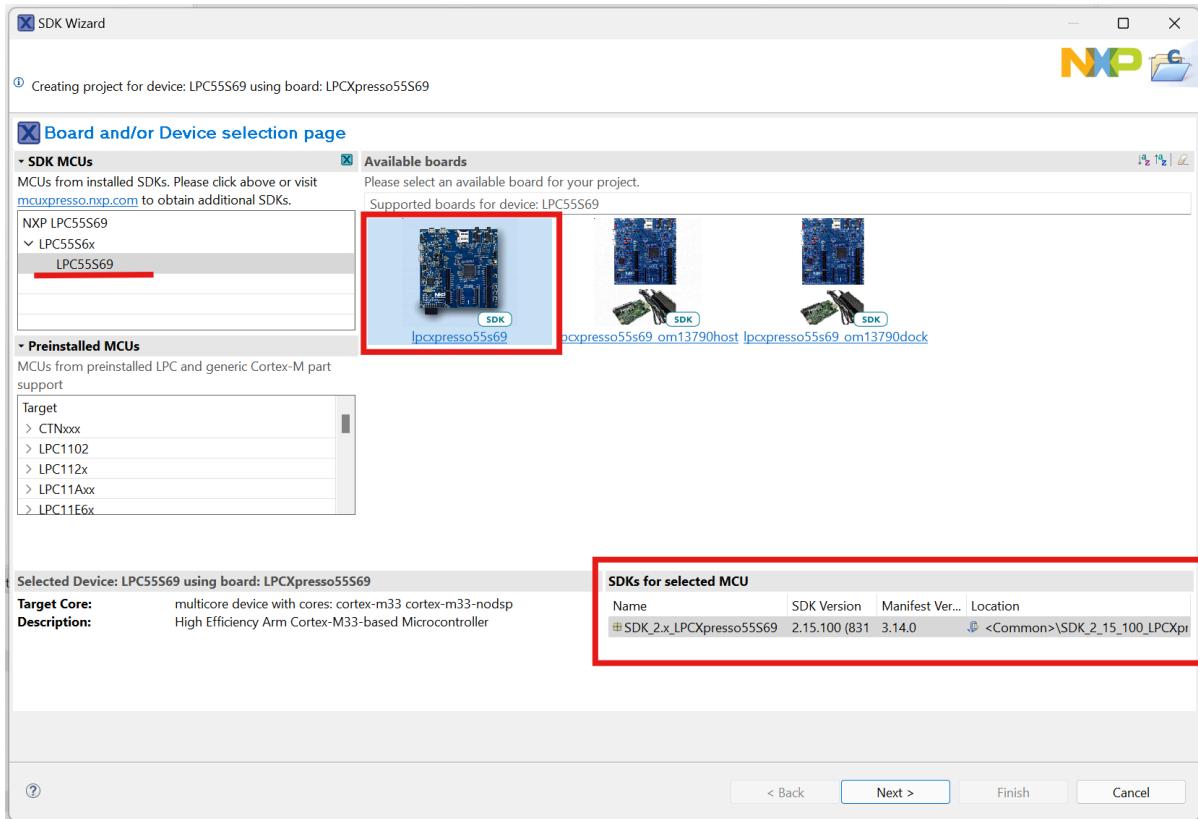
Welcome Page

3. Create New Project



Create new project

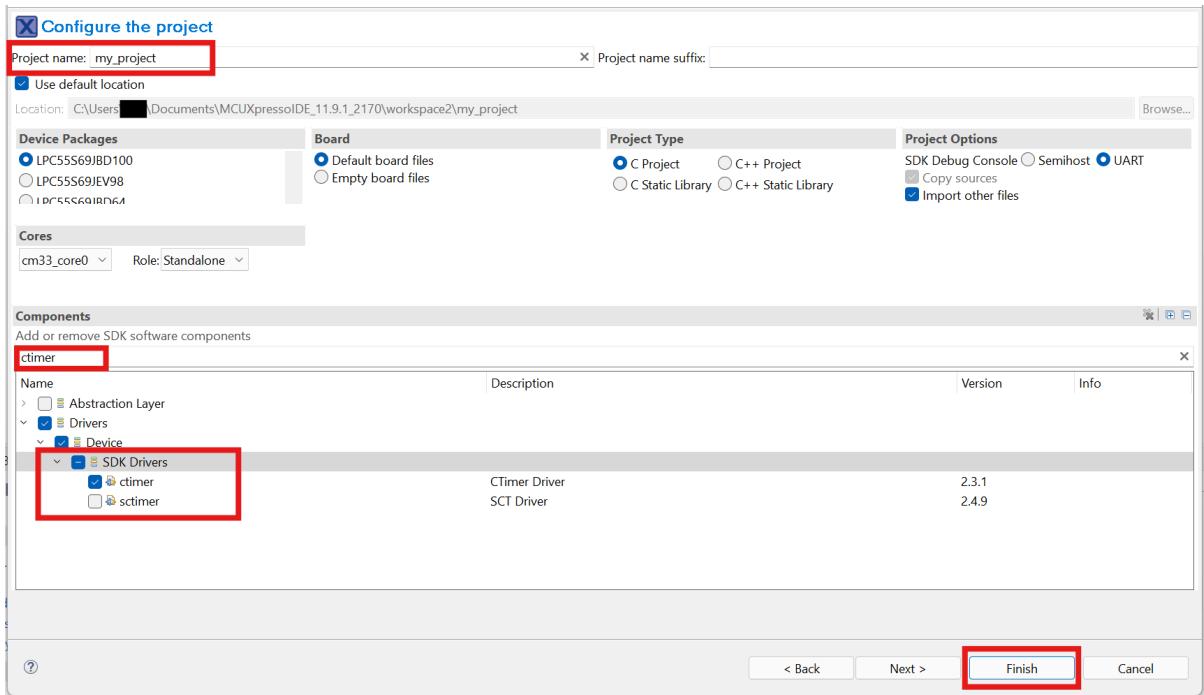
4. Select Development Board



SDK Wizard

5. Project Name, Desired APIs

The next step will allow us to give our project a representative name, and include the desired software components (e.g. the driver for ctimer).



SDK Wizard - Project Config

The project structure is presented below.

```

my.project <Debug>
  Project Settings
  Includes
  CMSIS
  board
    board.c
    board.h
    clock_config.c
    peripherals.c
    peripherals.h
    pin_mux.c
    pin_mux.h
  component
  device
  drivers
  source
    my_project.c
    semihost_hardfault.c
  startup
  utilities
  middleware

my.project.c
1 /*
2  * Copyright 2016-2024 NXP
3  * All rights reserved.
4  *
5  * SPDX-License-Identifier: BSD-3-Clause
6  */
7
8 /**
9  * @file my_project.c
10 * @brief Application entry point.
11 */
12 #include <stdio.h>
13 #include "board.h"
14 #include "peripherals.h"
15 #include "pin_mux.h"
16 #include "clock_config.h"
17 #include "LPC55S69_cm33_core0.h"
18 #include "fsl_debug_console.h"
19 /* TODO: insert other include files here. */
20
21 /* TODO: insert other definitions and declarations here. */
22
23 */
24 * @brief Application entry point.
25
26 int main(void) {
27     /* Init board hardware. */
28     BOARD_InitBootPins();
29     BOARD_InitBootClocks();
30     BOARD_InitBootPeripherals();
31
32 #ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
33     /* Init FSL debug console. */
34     BOARD_InitDebugConsole();
35 #endif
36
37     PRINTF("Hello World\r\n");
38
39     /* Force the counter to be placed into memory. */
40     volatile static int i = 0;
41     /* Enter an infinite loop, just incrementing a counter. */
42     while(1) {
43         i++;
44         /* Dummy' NOP to allow source level single stepping of
45          tight while() loop */
46         __asm volatile ("nop");
47     }
48
49 }
50

```

The application can now be built and executed.

The screenshot shows the NXP Studio IDE interface. The Project Explorer on the left lists the project structure under 'my_project <Debug>'. The Source Editor on the right displays the file 'my_project.c'. A red box highlights the line 'BOARD_InitBootBins();' in the code.

```

12 #include <stdio.h>
13 #include "board.h"
14 #include "peripherals.h"
15 #include "pin_mux.h"
16 #include "clock_config.h"
17 #include "LPC55S69_cm33_core0.h"
18 #include "fsl_debug_console.h"
19 /* TODO: insert other include files here. */
20
21 /* TODO: insert other definitions and declarations here. */
22
23/*
24 * @brief Application entry point.
25 */
26int main(void) {
27
28    /* Init board hardware. */
29    BOARD_InitBootBins();
30    BOARD_InitBootClocks();
31    BOARD_InitBootPeripherals();
32    #ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
33        /* Init FSL debug console. */
34    BOARD_InitDebugConsole();

```

Note: Remember to TERMINATE the program before rebuilding, or triggering another debug session.

Some useful tips:

- the `main` function (our application entry point) will be inside `source/<project_name>.c`.
- the `board` folder provides information and API interfaces for board-specific components, such as on-board LEDs, buttons, and communication interfaces. The macro definitions within these files map out the connections to these components, which can also be verified by referring to the board's schematic.

The screenshot shows the NXP Studio IDE interface. The Project Explorer on the left lists the project structure under 'my_project <Debug>'. The Source Editor on the right displays the file 'my_project.c'. A red box highlights the line 'void BOARD_InitLEDSPins(void);' in the code.

```

382 #define BOARD_INITLEDSPINS_LED_GREEN_GPIO_PIN_MASK (1U << 7U)
383 */
384 * @brief PORT peripheral base pointer */
385 #define BOARD_INITLEDSPINS_LED_GREEN_PORT iU
386 */
387 * @brief PORT pin number */
388 #define BOARD_INITLEDSPINS_LED_GREEN_PIN 7U
389 */
390 * @brief PORT pin mask */
391 #define BOARD_INITLEDSPINS_LED_GREEN_PIN_MASK (1U << 7U)
392 /* */
393
394 */
395 * @brief Configures pin routing and optionally pin electrical features.
396
397 */
398 void BOARD_InitLEDSPins(void); /* Function assigned for the Cortex-M33 (Core #0) */
399
400 #define ICON_PIO_ASW_DIS_EN 0x00u /*!<@brief Analog switch is closed (enabled), only for A0 version */
401 #define ICON_PIO_ASW_EN 0x0400u /*!<@brief Analog switch is closed (enabled) */
402 #define ICON_PIO_DIGITAL_EN 0x0100u /*!<@brief Enables digital function */
403 #define ICON_PIO_FUNC0 0x00u /*!<@brief Selects pin function 0 */
404 #define ICON_PIO_INV_DI 0x00u /*!<@brief Input function is not inverted */
405 #define ICON_PIO_MODE_PULLUP 0x20u /*!<@brief Selects pull-up function */
406 #define ICON_PIO_OPENDRAIN_DI 0x00u /*!<@brief Open drain is disabled */
407 #define ICON_PIO_SLEW_STANDARD 0x00u /*!<@brief Standard mode, output slew rate control is enabled */
408
409 */
410

```

- the `drivers` folder provides the drivers for all included components (on step 5), such as the USART driver.

- **header:** drivers/fsl_usart.h - contains macro definitions and functions prototypes.
- **source:** drivers/fsl_usart.c - contains C implementation for internal and external functionalities.
- **useful C functions:**

```
status_t USART_ReadBlocking(USART_Type *base,
                           uint8_t*data, size_t length);
status_t USART_WriteBlocking(USART_Type *base, const
                           uint8_t*data, size_t length);
```

Useful Links:

- [Download link & User Guide](#)

Hercules

Hercules SETUP utility is a useful serial port terminal (RS-485 or RS-232 terminal). This terminal application will be used for serial communication through UART.

This is the output we will receive when running the default `main` function generated when creating new projects. Additionally, we can send ASCII characters. By checking the `HEX` box, we can send hexadecimal strings to the device.



Hercules SETUP utility by HW-group.com

UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received/Sent data

Serial port COM3 opened

Hello World

Serial

Name

COM3

Baud

115200

Data size

8

Parity

none

Handshake

OFF

Mode

Free

Close

HWg FW update

Modem lines

CD

RI

DSR

CTS

DTR

RTS

Send

ABABAB

HEX

HEX

HEX

HW group

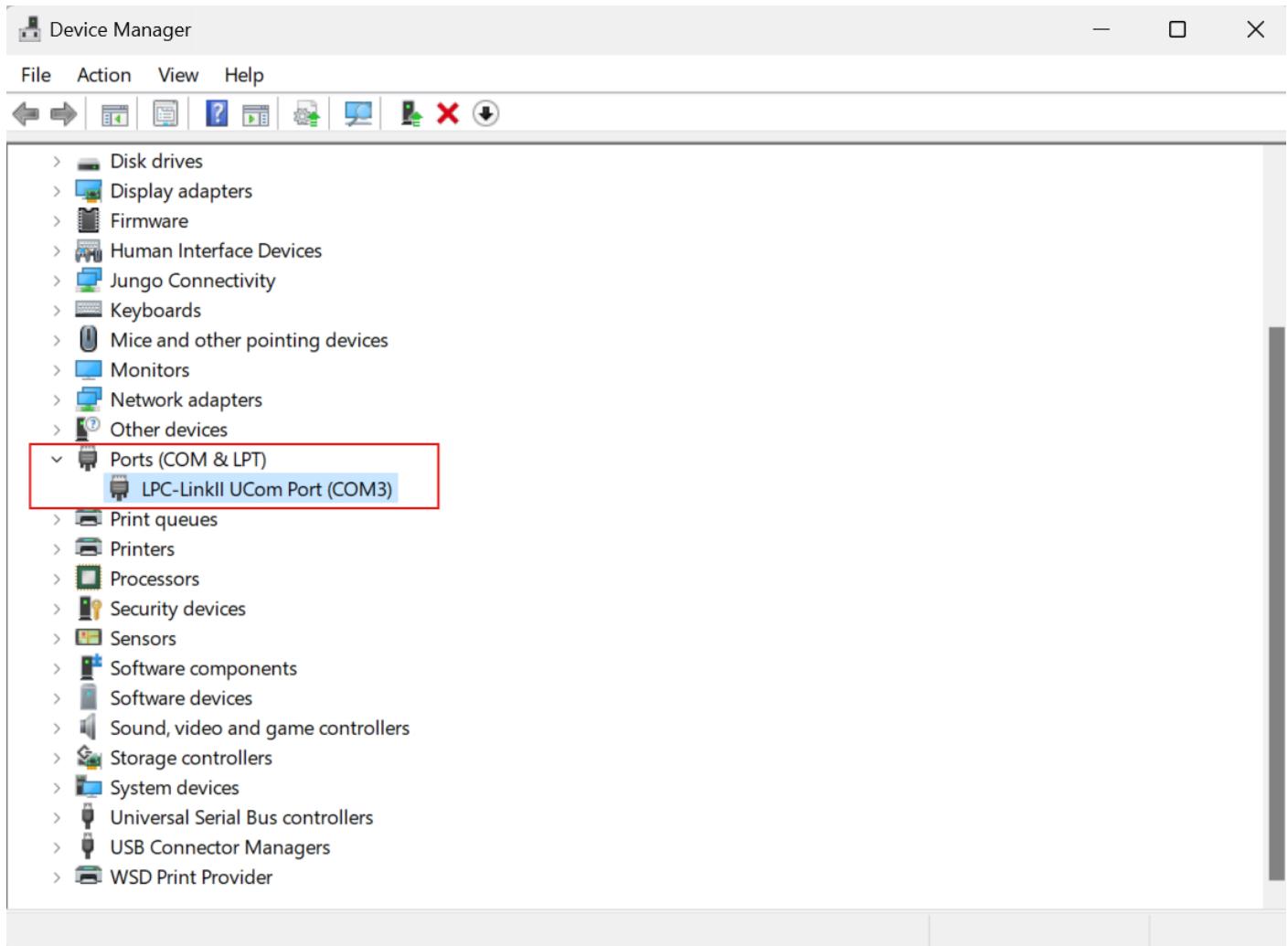
www.HW-group.com

Hercules SETUP utility

Version 3.2.8

Hello World

To identify the COM port of the device, open Device Manager.



Device Manager

Useful Links:

- [Download Hercules](#)